Actively Secure Two-Party Function Secret Sharing with Dynamic Cross-Phase Verification

Yujie Xue[1,3], Lin Liu[1,3⋆], Rongmao Chen[1,3], Yizhen Jiang[1,3], Yuchuan Luo[1,3], Bing Sun[2,3], and Shaojing Fu[1,3]

[1] College of Computer Science and Technology, National University of Defense Technology, Changsha, China
[2] College of Science, National University of Defense Technology, Changsha, China
[3] Center for Cryptologic Research, National University of Defense Technology, Changsha, China

Abstract. Function secret sharing (FSS) gives two parties succinct keys whose local evaluations add up to a hidden function value. Removing the dealer is the natural next step for preprocessing-based secure computation, but malicious security then requires more than checking a finished key: the generation transcript, the function the key actually computes, and every released evaluation share must all be bound to one execution. We call this the cross-phase binding problem, and we solve it.

We present *VeriFSS*, a dealer-free two-party FSS scheme with active security, proved in the standard simulation-based framework against one static malicious corruption. The construction rests on a two-plane key: the distributed generator produces a key for the paired function $x \mapsto (f_\theta(x), \Lambda f_\theta(x))$, where Λ is a secret global authentication scalar. The second plane makes every local evaluation an authenticated sharing for free, which yields online report binding without vector commitments, extractable hashing, or per-point interaction. It also supplies the ingredient the audit was missing: we prove a *moment-fidelity* lemma showing that a corrupted party cannot misreport the full-domain moments of its own sealed key except with probability $3/|E|$ over the challenge set E, and a *shape-identification* lemma showing that three moment equations pin a point function down exactly, with error $2n/|E|$ over any field. The proof isolates the coefficient of the square monomial in the support polynomial and is therefore characteristic-free; in characteristic two an odd-size equal-weight support defeats the support equation on its own, and we show the position equation takes over. A discrete-derivative reduction carries both lemmas to comparison functions, and a Galois-ring variant covers fixed-point payloads over $\mathbb{Z}_{2^k}$ by a 2-adic valuation argument, so no non-additive lift into a binary field is ever needed.

Generation costs two rounds and five field elements per party per level; certification adds $O(n)$ elements with no dependence on the domain size, and a dynamic cross-domain aggregation certifies arbitrarily many heterogeneous instances under one challenge and a constant number of rounds. We then lift the general-purpose FSS gates of Boyle et al.

⋆ Corresponding author: liulin16@nudt.edu.cn

(EUROCRYPT 2021): every gate that is a public affine post-processing of a constant number of DPF/DCF evaluations—interval containment, splines, ReLU, arithmetic shift, bit decomposition, zero test, table lookup—inherits active security at no additional online cost.
We evaluate our construction using a C++ implementation that certifies keys over $\mathbb{F}_{p^2}$ ($p = 2^{61} - 1$) and GF(2^{128}), realises the gate layer, and agrees bit-for-bit on every exported test vector. A certified DPF key at $n = 16$ takes 5.7 ms to generate and 3.3 ms to certify. Certified DPF keys are 5.4% larger than the semi-honest dealer-free baseline at $n = 16$, and aggregated certification traffic converges to 1 282 bytes per instance while the round count stays constant. Replaying the certified execution between two processes over TCP shows what that constancy is worth: at a 50 ms round trip a session takes 1.15 s, and the figure is unchanged whether one key or sixty-four are certified together, so wide-area certification of a whole preprocessing session is latency-bound by a single key. A campaign of 1,400 injected generation deviations plus shape, moment and release forgeries is rejected without exception in six of seven deviation classes; the seventh is rejected in exactly the 105 of 200 trials in which the attacked index bit is one, which is a quantitatively exact confirmation of the single-bit selective-failure predicate that we model explicitly in the ideal functionality, matching the leakage profile of the best known actively secure distributed DPF.

Keywords: Function secret sharing · Distributed point function · Secure two-party computation · Malicious security.

Table of Contents

1 Introduction 5
1.1 The cross-phase binding problem 5
1.2 Our contributions 6
1.3 Technical overview 8
1.4 Related work 9
1.5 Organisation 10
2 Preliminaries 11
2.1 Notation 11
2.2 Function secret sharing 11
2.3 Authenticated secret sharing 11
2.4 The audit algebra 12
2.5 Succinct challenge weights 13
2.6 The tree expander 14
2.7 Threat model 14
3 Auditable Dealer-Free FSS 14
3.1 Syntax 14
3.2 The ideal functionality 15
3.3 Binding properties 16
3.4 Two major bottlenecks hinder 17
4 The VeriFSS Construction 18
4.1 Overview 18
4.2 Phase I: PreCommit 20
4.3 Phase II: GenTrace for DPF 20
4.4 Phase II′: GenTrace for DCF 22
4.5 Phase III: GenCheck 22
4.6 Phase IV–V: evaluation and release 23
4.7 Cost 24
5 Security Analysis 25
5.1 The generation view is independent of Λ 25
5.2 Moment fidelity 26
5.3 Shape identification 26
5.4 Coverage of adversarial deviations 29
5.5 Generation soundness and report binding 30
5.6 Simulation 31
5.7 The residual leakage, quantified 31
5.8 Fiat–Shamir and aggregation 33

6 From Certified DPF/DCF to General FSS Gates 34
6.1 Authenticated offset gates 34
6.2 Deriving the offsets 36
6.3 Interval containment and its descendants 36
6.4 The table gate 37
6.5 Dynamic cross-domain aggregation 38
6.6 Composing with an authenticated arithmetic layer 38
7 Implementation and Evaluation 39
7.1 Artifact and methodology 39
7.2 Correctness and primitive costs 40
7.3 Key sizes 41
7.4 Communication, rounds and preprocessing 42
7.5 Dynamic cross-domain aggregation 43
7.6 The gate layer and privacy-computing workloads 45
7.7 The whole catalogue, semi-honest against certified 46
7.8 Cross-implementation check 51
7.9 Execution over a real transport 51
7.10 Adversarial campaign 52
7.11 Comparison with prior work 53
7.12 Head-to-head against every baseline of Table 17 55
7.13 Deployment note: PPML and secure model inference 58
7.14 Threats to validity 59
8 Conclusion 60
A Authenticated Sub-Protocols 65
B Proof of Theorem 2 66
C The Ring Instantiation 67
C.1 Parameters 67
C.2 Why $s \geq k$ is necessary and not merely convenient 68
C.3 Schwartz–Zippel over the exceptional set 68
C.4 Interaction with the tag plane 68
C.5 What does *not* work 69
D Gate Specifications 69
E Pseudocode for the Gate Catalogue 70
E.1 The two families 71
E.2 Threshold family 72
E.3 Table family 74

1 Introduction

Function secret sharing (FSS) splits a function f from a family $\mathcal{F}$ into two succinct keys such that each party can evaluate its key locally at any input and the two results add up to $f(x)$, while a single key hides f entirely. Distributed point functions (DPFs) were introduced by Gilboa and Ishai [23], and Boyle, Gilboa and Ishai developed the general abstraction together with the tree-based constructions that made it practical [8,9]. The primitive now underlies private information retrieval, anonymous messaging, private aggregation, oblivious RAM, and—since the introduction of *FSS gates* [10,3]—a large part of the preprocessing-model secure-inference stack [29,28,42,44].

The efficiency of that stack comes from an asymmetry: an expensive key phase followed by essentially free local evaluation. In almost all deployments the key phase is run by a trusted dealer. Removing the dealer is desirable for obvious reasons, and semi-honest distributed key generation has been known since Doerner and shelat [20]. What remains open is a *dealer-free two-party FSS scheme whose keys are certified against a malicious generator on the whole domain and whose evaluation shares remain bound to those keys*, together with a treatment of the general-purpose gates that make FSS useful.

1.1 The cross-phase binding problem

Malicious security here is not a single check but a chain of three bindings that are easy to conflate. They are *cross-phase*: each one couples two different phases of the protocol—generation to certification, certification to evaluation, evaluation to release—and it is the coupling, not any single phase, that the literature has left unaddressed.

Key binding.
An accepting generation transcript must determine one key pair; no party may later evaluate a different key.

Function binding.
The determined pair must realise the *intended* f_θ on the whole domain, not merely some well-formed member of the family. A tampered correction word can produce a perfectly well-formed DPF for the wrong point.

Report binding.
Every released evaluation share must be the deterministic output of Eval on the certified key at the queried point.

These are logically independent. Verifiable FSS in the client–server model [2,16] attacks function binding for a *dealer-supplied* key and leaves the other two aside, because the servers are honest there. Actively secure distributed DPF generation [50] attacks key and function binding but pays one bit of leakage and says nothing about report binding. Authenticating an arbitrary field element gives report binding for that element but no link to an FSS computation.

Two obstacles have kept the three bindings apart. The first is *moment fidelity*. Full-domain audits check a constant number of random linear functionals (moments) of the evaluation vector; but each party computes its own moment share locally from its own key, so the audit must first establish that the number a party feeds into the check really is the moment of the key it sealed. A digest of the evaluation vector does not give this: a collision-resistant hash binds an honestly supplied opening to its own digest, not a separately supplied linear functional to the hashed vector. The second is *algebraic compatibility*. Fixed-point payloads live in $\mathbb{Z}_{2^k}$ while the natural audit algebra is a large field; the bit-string map $\mathbb{Z}_{2^{64}} \to \mathrm{GF}(2^{128})$ is not even additive, since $\mathsf{lift}(1) + \mathsf{lift}(1) = 0 \neq \mathsf{lift}(2)$, so a scheme that adds ring values and authenticates their images in a binary field does not maintain its own tag invariant.

1.2 Our contributions

A dealer-free, actively secure two-party FSS scheme. We give Π_{VeriFSS}, a complete construction with a proof in the standard real/ideal framework against one static malicious corruption, with abort and with an explicitly modelled single-bit selective-failure predicate on the index—the same leakage profile as the best known actively secure distributed DPF [50], but now accompanied by full-domain function binding, report binding and a gate layer. The pivot is a *two-plane key*: the distributed generator produces a key for the paired function

$$F_{\theta,\Lambda} : x \longmapsto \big(f_\theta(x),\ \Lambda \cdot f_\theta(x)\big),$$

where Λ is a global authentication scalar, secret-shared and never opened. Two consequences follow, and together they close the chain. First, the pair $(y_b(x), t_b(x))$ produced by local evaluation is already a Λ-authenticated sharing of $f_\theta(x)$, so report binding costs one field element per released point and no interaction, and composes directly with an authenticated arithmetic layer. Second, the tag plane binds the audit: we prove (Lemma 3) that a corrupted party which misreports the value moments of its sealed

key must produce $\Lambda\delta$ for a nonzero δ, hence succeeds with probability at most $1/|E|$ per moment, where E is the challenge set. This discharges the moment-fidelity obligation without vector commitments, straight-line extraction, or succinct proofs of tree evaluation.

A tight shape-identification lemma, valid in characteristic two. Given fidelity, three moment equations—mass, position and support—identify the evaluation vector exactly. We prove (Lemma 5) that over any field, if $g \neq \beta e_\alpha$ then the three equations hold with probability at most $2n/|\mathbb{F}|$ over the succinct challenge $r_x(\rho) = \prod_{i:x_i=1} \rho_i$. The proof isolates the coefficient of the square monomial ρ^{2x} in the support polynomial and shows it equals $g_x(g_x - z_1)$, which is a characteristic-free statement. The corollary that all nonzero entries are equal is fatal in characteristic two, where any odd-size equal-weight support satisfies mass and support identically; the position equation then does the work, and we show it suffices. The zero-payload case $\beta = 0$, previously left open, is handled by the same coefficient argument. A discrete-derivative reduction (Lemma 7) transfers everything to comparison functions using an *invertible* difference operator, which removes the need for a separate boundary residual.

Ring payloads without an illegal lift. For fixed-point computation we instantiate the audit over the Galois ring $\mathrm{GR}(2^{k+s}, d)$ with its Teichmüller exceptional set. Zero divisors break the "all entries lie in $\{0, z_1\}$" step, and we replace it by a 2-adic valuation argument (Lemma 6): whenever $s \geq k$, every entry that survives the support equation is congruent to 0 or β modulo 2^k, which is exactly the precision at which the payload is defined. The scheme is therefore correct over $\mathbb{Z}_{2^k}$ with soundness $2n/2^d$, and no non-additive lift appears anywhere.

General-purpose gates. We lift the FSS gates of Boyle, Chandran, Gilboa, Gupta, Ishai, Kumar and Rathee [3]. Theorem 3 states the lift in the form the framework deserves: every gate that is a public affine post-processing of a constant number of DPF/DCF evaluations at publicly derived points—interval containment, multiple interval containment, splines, ReLU, arithmetic and logical shift, bit decomposition, zero test—inherits active security with *no* additional online communication, because public affine maps preserve the tag invariant. The offset randomness is drawn from the correlation pool in authenticated bit form and shifted by public constants with a ripple-carry adder, whose carry-out doubles as the wrap correction the gate needs. We also isolate a gate that is specific to our setting: one certified DPF plus a public table realises an *arbitrary* function

on n bits with zero online communication beyond opening the masked input.

Dynamic cross-domain aggregation. Residual ledgers of independent instances, over different domains, families and payload algebras, may be concatenated under one challenge and one CheckZero. Certification is then $O(n)$ elements per instance plus a fixed $O(1)$ overhead, and—this is the operationally important part—a constant number of rounds no matter how many instances are in flight. Measured certification traffic falls from 1 504 to 1 282 bytes per instance between one and 128 instances at $n = 12$, with the round count pinned at eleven.

A reference implementation that reproduces every claim. The artifact implements the complete workflow—correlation pool, distributed generation, certification, local and aggregated evaluation, release, and the gate layer—over two audit algebras of different characteristic and over a Galois ring. It executes as two parties over an instrumented channel that charges every byte and every round. All honest executions are accepted; a campaign of 1,400 injected generation deviations, plus shape attacks, moment forgeries and release forgeries, is rejected in six of seven deviation classes without exception. The seventh class is rejected in exactly the trials in which the attacked index bit equals one, which measures the selective-failure predicate of Theorem 2 rather than contradicting it; and a Λ-aware release forgery is accepted, witnessing that the reduction of Lemma 3 is tight. Section 7 reports measured communication, rounds, key sizes and pool consumption.

1.3 Technical overview

Generation. Both parties expand their entire tree level by level, as in the full-domain strategy of [20]. The invariant is that off-path nodes are *equal* between the parties, so the sum of a party's level—seeds, control values and, for DCF, value words—differs from its partner's sum by exactly the on-path node's contribution. Correction words are therefore affine in quantities each party holds privately, and the only secret-dependent step per level is a selection by the bit α_i, which we perform with one authenticated multiplication. Crucially, correction words are opened *without* authentication: a deviation there can only change the generated key, and every change to the key is caught downstream. Generation costs two rounds and five field elements per party per level. Exactly one deviation class escapes this argument: an additive error injected into the Beaver mask of the level

selection is multiplied by the secret bit α_i and is therefore invisible when $\alpha_i = 0$. We do not hide this. We isolate it, prove that it is the *only* such class (Lemma 8), model it in the functionality as a single-bit predicate, and show in Section 5.7 that dynamic cross-domain aggregation makes it self-limiting: harvesting k bits requires surviving k independent coin flips.

Certification. After the keys are sealed the parties draw ρ and each computes six full-domain moments, three on the value plane and three on the tag plane. The three residuals $u_\bullet - \Lambda z_\bullet$ enforce fidelity; the residuals $z_1 - \beta$, $z_\rho - \beta r_\alpha$ and $z_\rho^2 - z_1 z_{\rho^2}$ enforce shape. All six are checked by one $\mathsf{CheckZero}$. The audit is $\Theta(N)$ local work—the same order as generation—and $O(n)$ communication.

What the tag plane does not *do.* We are explicit about one point that a superficial reading would get wrong. The audit does not certify the tag plane: an adversary that corrupts h away from Λg can always patch its reported tag moments, because it knows the defect. What it cannot do is move the *value* moments, and that is what the shape lemma consumes. At release time a tag defect forces the adversary to patch again or abort; either way the value that is accepted is the correct one (Lemma 9). We prove this rather than assume it away.

1.4 Related work

FSS constructions. After [23,8,9], key sizes and evaluation costs were improved by Half-Tree [26], and the primitive was extended to programmable [12], multi-point [7], non-interactive [6], information-theoretic [11,37], CNF-based [13] and multi-party [24,35,19] settings. FSS gates and the offset-family paradigm [10,3] moved FSS into mixed-mode fixed-point computation and are the abstraction we lift.

Verifiability. Boneh et al. [2] introduced the sketching test for client-generated DPF shares in private heavy hitters; de Castro and Polychroniadou [16] gave maliciously secure verifiable FSS for general-output point and multi-point functions with constant verification communication across many clients. These works verify a *dealt* key against honest servers. Our setting is complementary: the key is jointly generated by the two evaluators, and the same parties must later be bound to their own reports. Further work in the verifiable line includes [34,36,51,39,30,31].

Why not a fully linear PCP? The moment ledger verifies a linear-algebraic statement about a secret-shared vector, which is what fully linear PCPs are for [1]. The reason we do not use one is the reason the fidelity problem exists. An FLPCP proves a statement about a vector held by a *prover*; here the evaluation vector is additively split between the two parties, neither holds it, and a proof produced jointly would itself need the binding that the proof was supposed to supply. The tag plane breaks the circularity because it is manufactured during generation, before any statement is made. An FLPCP remains the right tool when a single client generates the key, which is the client–server setting of [2,16].

Dealer removal and active security. Doerner and shelat [20] gave the semi-honest full-domain distributed generator we build on. Zhang et al. [50] obtained actively secure distributed DPF generation at semi-honest cost but with one bit of leakage; our audit-based route removes the leakage at the price of a $\Theta(N)$ certification pass that is of the same order as the generation itself. Non-interactive DPF [6] removes interaction from setup but is semi-honest. On the application side, SHARK [27] and LightShark [49] build actively secure inference from FSS with a dealer, and Cavern [38] attains malicious security for spline evaluation in the honest-majority $(2+1)$-party model via a verifiable incremental DPF. We compare against these in Section 7.11.

Authenticated arithmetic. The MAC machinery is SPDZ-style [18,32,22]; ring payloads use SPD$\mathbb{Z}_{2^k}$ [17] and cross-domain conversions use daBit/edaBit techniques [21]. Correlated randomness can be produced by silent OT/VOLE [5,4,47,40,41] and, if committed correlations are preferred, by [43].

1.5 Organisation

Section 2 fixes notation and the algebra. Section 3 defines auditable dealer-free FSS and the ideal functionality. Section 4 gives the construction and its cost. Section 5 contains the analysis. Section 6 lifts the FSS gates and describes cross-domain aggregation. Section 7 reports the evaluation. Deferred proofs, the ring instantiation and gate specifications are in the appendices.

2 Preliminaries

2.1 Notation

λ is the computational and σ the statistical security parameter. For $N = 2^n$ we write $[N] = \{0, \dots, N-1\}$ and identify $x \in [N]$ with its bit string $x_1 \cdots x_n$, most significant bit first. Parties are P_0, P_1, indexed by $b \in \{0,1\}$, and $\bar{b} = 1 - b$. We write $e_\alpha \in \mathbb{K}^N$ for the α-th unit vector, $\mathrm{supp}(g)$ for the support of a vector, $\mathsf{negl}(\lambda)$ for a negligible function and $X \approx_c Y$ for computational indistinguishability. Vectors are column vectors and $\langle a, b \rangle$ is the inner product.

Throughout, $\mathbb{K}$ denotes the *audit algebra*: either a finite field $\mathbb{F}$ or a Galois ring, as fixed in Section 2.4. Elements of $\mathbb{K}$ have a canonical $\lceil \log_2 |\mathbb{K}| \rceil$-bit encoding.

2.2 Function secret sharing

Definition 1 (FSS [8,9]). *Let $\mathcal{F} = \{f_\theta : [N] \to \mathbb{G}\}_{\theta \in \Theta}$ be a family of functions into an Abelian group $\mathbb{G}$. A two-party FSS scheme for $\mathcal{F}$ is a pair $(\mathsf{Gen}, \mathsf{Eval})$ with $(k_0, k_1) \leftarrow \mathsf{Gen}(1^\lambda, \theta)$ and $y_b \leftarrow \mathsf{Eval}(b, k_b, x)$ such that* (correctness) $\mathsf{Eval}(0, k_0, x) + \mathsf{Eval}(1, k_1, x) = f_\theta(x)$ *for all $x \in [N]$, and* (function privacy) *for each b there is a simulator $\mathcal{S}_b$ with $\mathcal{S}_b(1^\lambda, \mathsf{leak}(\theta)) \approx_c k_b$, where* leak *reveals only the public family description.*

A *distributed point function* (DPF) is an FSS scheme for $f_{\alpha,\beta}(x) = \beta$ if $x = \alpha$ and 0 otherwise [23]. A *distributed comparison function* (DCF) is an FSS scheme for $f^{<}_{\alpha,\beta}(x) = \beta \cdot \mathbb{1}\{x < \alpha\}$; the boundary convention $f^{<}_{\alpha,\beta}(\alpha) = 0$ is part of the specification. We write $\mathsf{EvalAll}(b, k_b) = (\mathsf{Eval}(b, k_b, x))_{x \in [N]}$.

We stress a distinction used throughout: an *FSS key* is an object on which the specified Eval algorithm realises the specified function. A byte array of the right length is not an FSS key, and neither is a well-formed key for the wrong parameter.

2.3 Authenticated secret sharing

Fix a global MAC key $\Delta \in \mathbb{K}$, additively shared as $\Delta = \Delta_0 + \Delta_1$. An *authenticated sharing* $[\![x]\!]$ of $x \in \mathbb{K}$ gives P_b a pair (x_b, γ_b) with

$$x_0 + x_1 = x, \qquad \gamma_0 + \gamma_1 = \Delta \cdot x. \tag{1}$$

Affine maps with public coefficients are local; multiplication consumes a Beaver triple. We use the standard commit-then-open MacCheck and

Functionality $\mathcal{F}_{\mathsf{prep}}$ (parameters: algebra $\mathbb{K}$)

- Init: sample $\Delta, \Lambda \leftarrow \mathbb{K}$, send Δ_b, Λ_b to P_b for a random additive splitting.
- Rand: sample $r \leftarrow \mathbb{K}$ and deliver $[\![r]\!]$; on request Rand(b) additionally send r to P_b.
- Bit: sample $u \leftarrow \{0,1\} \subseteq \mathbb{K}$ and deliver $[\![u]\!]$.
- Triple: sample $a, b \leftarrow \mathbb{K}$ and deliver $[\![a]\!], [\![b]\!], [\![ab]\!]$.

The adversary may specify additive errors on its own shares before delivery and may abort. All items are consumed once.

Fig. 1. The correlation pool. Instantiating it is orthogonal to this work; Section 7.4 converts the measured unit counts into communication using published per-unit costs.

CheckZero sub-protocols [18,32]; both are recalled in Appendix A. A second, independent key Λ plays a different role: it is never used to authenticate protocol wires, only to authenticate *FSS outputs*, and it is embedded in the key material itself. We write $\langle\!\langle x \rangle\!\rangle = (x_b, \tau_b)$ with $\tau_0 + \tau_1 = \Lambda x$ for a Λ-authenticated sharing.

Remark 1 (Why two keys). Δ authenticates the offline arithmetic that produces and checks the key; Λ authenticates the values that the key later outputs. Identifying the two is possible and costs nothing in the proof—the analysis only uses that each key is uniform and unopened—but keeping them apart makes the accounting of Section 5 modular and lets a deployment refresh the online key without regenerating the offline pool.

The correlation pool. $\mathcal{F}_{\mathsf{prep}}$ (Figure 1) supplies $[\![\Delta]\!]$-authenticated random values, authenticated random bits, and multiplication triples, all in $\mathbb{K}$. It is realised with malicious security from vectorial oblivious linear evaluation or correlated OT [5,47,40,32]; we work in the $\mathcal{F}_{\mathsf{prep}}$-hybrid model, as is standard, and account for its instantiation cost separately in Section 7.4.

2.4 The audit algebra

The analysis needs a challenge domain in which a nonzero low-degree polynomial is unlikely to vanish. We use one of the following.

Binary field.
$\mathbb{K} = \mathrm{GF}(2^\kappa)$, $\kappa \geq 128$. Addition is XOR, so the FSS tree is bit-for-bit the classical GGM/BGI tree. This is the natural choice for Boolean payloads: private information retrieval, keyword search, heavy hitters, table lookup.

Prime extension field.
$\mathbb{K} = \mathbb{F}_{p^d}$ with p prime. The artifact uses $p = 2^{61} - 1$, $d = 2$, so $|\mathbb{K}| = (2^{61} - 1)^2 > 2^{122}$ and every arithmetic operation fits in 64-bit lanes. Fixed-point payloads are encoded in the prime subfield.

Galois ring.
$\mathbb{K} = \mathrm{GR}(2^m, d) = \mathbb{Z}_{2^m}[X]/(h(X))$ with h a Hensel lift of a degree-d irreducible polynomial over $\mathbb{F}_2$, and $m = k + s$. Payloads live in $\mathbb{Z}_{2^k}$, matching fixed-point secure computation directly. Challenges are drawn from the Teichmüller exceptional set $E = \{\sum_i e_i X^i : e_i \in \{0, 1\}\}$ of size 2^d; differences of distinct elements of E are units.

Lemma 1 (Schwartz–Zippel over exceptional sets [17]). *Let R be a commutative ring with exceptional set E and let $Q \in R[\rho_1, \ldots, \rho_n]$ be nonzero of total degree D. Then $\Pr_{\rho \leftarrow E^n}[Q(\rho) = 0] \le D/|E|$. For a field, $E = \mathbb{K}$.*

Remark 2 (The lift that does not exist). It is tempting to keep $\mathbb{Z}_{2^{64}}$ payloads and authenticate their images under $x \mapsto (x, 0) \in \mathrm{GF}(2^{128})$. That map is not additive: $\mathsf{lift}(1) + \mathsf{lift}(1) = 0$ while $\mathsf{lift}(2) \neq 0$. A scheme that adds ring values and authenticates binary-field images therefore violates Eq. (1) on the very first addition. The Galois-ring instantiation exists precisely to avoid this, and Lemma 6 shows that the audit survives the zero divisors it introduces.

2.5 Succinct challenge weights

All audits use the multilinear weight family

$$r_x(\rho) \;=\; \prod_{i\,:\,x_i=1} \rho_i, \qquad \rho = (\rho_1, \ldots, \rho_n) \in E^n. \tag{2}$$

The full vector $(r_x)_{x \in [N]}$ is computed in $\Theta(N)$ operations from n challenge elements by one pass of the recursion $r_{2j}^{(i+1)} = r_j^{(i)}$, $r_{2j+1}^{(i+1)} = r_j^{(i)} \rho_{i+1}$. Distinct x give distinct multilinear monomials; this is the only property the analysis uses. For $\alpha \in [N]$ with authenticated bits $[\![\alpha_i]\!]$,

$$r_\alpha(\rho) = \prod_{i=1}^{n} \big(1 + \alpha_i(\rho_i - 1)\big) \tag{3}$$

is computed from the shares by $n-1$ authenticated multiplications arranged in a balanced tree of depth $\lceil \log_2 n \rceil$.

2.6 The tree expander

The construction expands each node with the fixed-key two-call map

$$G(s) = \big(\pi(s \oplus c_0) \oplus s,\ \pi(s \oplus c_1) \oplus s\big), \tag{4}$$

where π is a public permutation instantiated by AES-128 under a fixed key and $c_0 \neq c_1$ are public tweaks. The security notion this use requires is not plain pseudorandomness of G: the same π is queried on inputs that are related across nodes and across the two parties. We assume that π is *circular correlation robust* in the sense of Guo, Katz, Wang and Yu [25], which is exactly the notion under which the map of Eq. (4) yields outputs indistinguishable from uniform to a party that does not hold s, in the ideal-cipher model. We write $\varepsilon_{\mathrm{prg}}$ for the resulting advantage; every appeal to "pseudorandomness of G" in Section 5 means this assumption. Nothing else in the analysis depends on the choice of expander, so a Half-Tree-style single-call construction [26] may be substituted with its own correlation-robustness assumption.

2.7 Threat model

Two parties, at most one statically corrupted by a malicious, rushing probabilistic polynomial-time adversary. Security is with abort; fairness, guaranteed output delivery, adaptive corruption and side channels are out of scope. Evaluation points are public. Sessions are separated by a session identifier that is prefixed to every hash and every $\mathcal{F}_{\mathsf{prep}}$ request, and the generation verdict for a key is fixed before that key answers any evaluation request.

3 Auditable Dealer-Free FSS

This section fixes the syntax, the ideal functionality, and the three binding properties that a construction has to deliver. The definitions are stated so that each one is falsifiable on its own; Section 5 discharges all of them for the construction of Section 4.

3.1 Syntax

Definition 2 (Auditable dealer-free FSS). *For a family $\mathcal{F} = \{f_\theta\}_{\theta \in \Theta}$, an auditable dealer-free FSS scheme is a tuple of protocols and algorithms*

$$(\mathsf{PreCommit},\ \mathsf{GenTrace},\ \mathsf{GenCheck},\ \mathsf{Eval},\ \mathsf{Release})$$

with the following interfaces.

$\mathsf{PreCommit}(1^\lambda, \mathsf{sid}, \mathsf{inv})$.
An interactive protocol producing $[\![\Delta]\!]$, $[\![\Lambda]\!]$ *and an inventory* inv *of authenticated correlations. Its inputs are public.*

$\mathsf{GenTrace}([\![\theta]\!])$.
An interactive protocol taking authenticated parameter handles and outputting to P_b *a key* k_b *and a local state, together with a public transcript* tr.

$\mathsf{GenCheck}(\mathsf{tr}, k_b)$.
An interactive protocol outputting a single bit $v \in \{\mathsf{accept}, \mathsf{reject}\}$, *run after* tr *and both keys are sealed and before any evaluation request.*

$\mathsf{Eval}(b, k_b, x)$.
A deterministic local algorithm outputting a pair $(y_b, \tau_b) \in \mathbb{K}^2$.

$\mathsf{Release}(\{(x_j, y_b^{(j)}, \tau_b^{(j)})\}_j)$.
An interactive protocol that reconstructs the requested outputs and outputs a bit.

The syntax makes the audit an explicit phase with its own verdict, and makes Eval produce a *pair*. Both choices are load-bearing: the verdict separates key certification from key use, and the second component is the Λ-tag that binds a released value to the certified key.

3.2 The ideal functionality

$\mathcal{F}_{\mathsf{aFSS}}$ (Figure 2) is deliberately close to the ordinary dealer-based FSS ideal: it samples an *honest* key pair for the reconstructed parameter and hands one key to each party. An adversary therefore cannot obtain a key for a function of its choosing, cannot make a released value differ from $f_\theta(x)$, and cannot learn θ beyond one bit of the selective-failure predicate. This is the strongest of the formulations in the literature that is achievable here, and it is strictly stronger than a definition that merely asks the reconstructed output to be correct.

Remark 3 (On the predicate class). Π_{sf} is a conjunction of conditions on individual index bits, which is exactly what an adversary can test in a level-structured tree generator. Zhang et al. [50] model the same phenomenon as "one-bit leakage". Note that $\Pr[\pi_S(\theta) = 1] = 2^{-|S|}$ for a uniform index, so an adversary that wants k bits must survive k independent $\frac{1}{2}$-probability detections; Section 5.7 makes this quantitative and shows that aggregation over m instances multiplies the penalty.

Functionality $\mathcal{F}_{\mathsf{aFSS}}$

Parameters: family $\mathcal{F}$ over $[N]$, algebra $\mathbb{K}$, corrupted party P_{b^*}.

Setup.

Sample $\Lambda \leftarrow \mathbb{K}$ and send Λ_{b^*} to $\mathcal{A}$ for a random additive splitting.

Generate$(\mathsf{kid}, \theta_0, \theta_1)$**.**

On matching requests, set $\theta = \theta_0 + \theta_1$.

1. Receive from $\mathcal{A}$ a *leakage predicate* $\pi \in \Pi_{\mathrm{sf}}$, where $\Pi_{\mathrm{sf}} = \{\pi_S : \theta \mapsto \bigwedge_{i \in S}(\theta_i = 0) \ : \ S \subseteq [n]\}$ is the class of index-bit conjunctions.
2. Send $\pi(\theta)$ to $\mathcal{A}$. If $\pi(\theta) = 0$, send abort to both parties and halt this key.
3. Store $(\mathsf{kid}, \theta, \mathsf{accept})$ and send accept to both parties. $\mathcal{A}$ may instead abort; the record is then discarded.

Once accept is stored the record is immutable. Keys are internal to the functionality; parties observe only verdicts and evaluation results.

Evaluate$(\mathsf{kid}, x_1, \ldots, x_m)$**.**

If the record is accept, compute $v_j = f_\theta(x_j)$. Send $\{v_j\}$ to $\mathcal{A}$; on $\mathcal{A}$'s $\mathsf{deliver}$ send $\{v_j\}$ to the honest party, on $\mathcal{A}$'s abort send abort.

Fig. 2. The ideal auditable dealer-free FSS functionality. The only non-standard element is the single-bit selective-failure predicate of step 1–2; setting $\Pi_{\mathrm{sf}} = \{\mathbf{1}\}$ recovers the leakage-free notion, which Section 5.7 shows is achievable at the cost of a generic secure evaluation of the key-generation circuit.

3.3 Binding properties

The following three definitions are the working form of $\mathcal{F}_{\mathsf{aFSS}}$; the proof of Theorem 2 establishes them and then assembles the simulator.

Definition 3 (Generation soundness). *Let θ^* be the value determined by the authenticated parameter handles, let (k_0, k_1) be the key pair fixed at the end of* GenTrace*, and let g be the function their value planes realise. The scheme has generation soundness $\varepsilon_{\mathrm{gen}}$ if for every efficient adversary*

$$\Pr\big[\mathsf{GenCheck} = \mathsf{accept} \ \wedge \ \exists x \in [N] : \ g(x) \neq f_{\theta^*}(x)\big] \ \leq \ \varepsilon_{\mathrm{gen}}.$$

Definition 3 is a full-domain statement. It rules out the attack that verifiable-FSS definitions in the client–server model do not have to consider, namely a perfectly well-formed key for the wrong index.

Definition 4 (Report binding). *Write $g(x) = \mathsf{Eval}^{\mathrm{val}}(0, k_0, x) + \mathsf{Eval}^{\mathrm{val}}(1, k_1, x)$. The scheme has report binding $\varepsilon_{\mathrm{rep}}$ if for every efficient adversary and every public query set $X = (x_1, \ldots, x_m)$ answered after an accepting* GenCheck*,*

$$\Pr\big[\mathsf{Release} = \mathsf{accept} \ \wedge \ \exists j : \ \widetilde{v}_j \neq g(x_j)\big] \ \leq \ \varepsilon_{\mathrm{rep}},$$

where $\widetilde{v}_j$ is the value the honest party outputs.

Definition 5 (Execution privacy). *The scheme has execution privacy if for each b there is a simulator that, given 1^λ, the public family description, the corrupted party's parameter share and $\mathcal{F}_{\mathsf{prep}}$ outputs, the value $\pi(\theta^*)$ for the predicate the adversary submits, the public query set, and the released values $f_{\theta^*}(x_j)$, produces a view computationally indistinguishable from the real view of P_b in the whole execution* PreCommit‖GenTrace‖GenCheck‖Eval‖Release.

The definition deliberately covers all five phases. Restricting it to generation would leave the release phase unconstrained, and release is where the adversary receives the only values in the protocol that depend on the honest party's key. The simulator of Theorem 2 establishes the definition in this full form.

3.4 Two major bottlenecks hinder

Two obligations resist the obvious approaches, and naming them precisely is what makes the construction possible.

Moment fidelity. A full-domain audit checks a constant number of linear functionals of the evaluation vector g. Each party computes its own share of such a functional locally, so the audit must first establish that the number a party submits is the functional of the key it sealed. This does *not* follow from committing to a digest of the evaluation vector: a collision-resistant hash binds an honestly supplied opening to its own digest, whereas here the audit consumes a separately supplied linear functional, and nothing links the two. Vector commitments with linear openings would suffice but cost $\Omega(N)$ work *and* a proof system. Section 5.2 obtains fidelity from the tag plane instead, for free.

Shape identification in characteristic two. Given fidelity, the audit still has to conclude "g is a point function at the right index with the right payload" from a constant number of moment equations. The natural test $z_\rho^2 = z_1 z_{\rho^2}$ is not by itself a unit-support test: in characteristic two *any* odd-size support with equal nonzero entries satisfies it identically, and also satisfies the mass equation. Section 5.3 gives the coefficient identity that makes the analysis characteristic-free and shows which equation rescues which case, including $\beta = 0$.

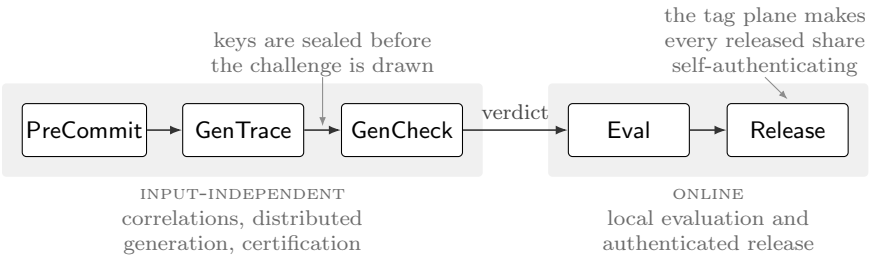


Fig. 3. The five phases. The verdict of GenCheck is fixed before the key answers any query, and the same Λ that certifies the key authenticates its outputs.

4 The VeriFSS Construction

4.1 Overview

The scheme has one structural idea and three consequences. The idea is that the distributed generator does not produce a key for f_θ; it produces a key for the *paired* function

$$F_{\theta,\Lambda} : [N] \to \mathbb{K} \times \mathbb{K}, \qquad x \mapsto \big(f_\theta(x),\ \Lambda \cdot f_\theta(x)\big), \tag{5}$$

where Λ is the secret output authentication key from $\mathcal{F}_{\mathsf{prep}}$. Writing $g, h : [N] \to \mathbb{K}$ for the two functions that the sealed key pair actually realises,

$$g(x) = y_0(x) + y_1(x), \qquad h(x) = \tau_0(x) + \tau_1(x),$$

an honest execution has $h = \Lambda g$ pointwise. The three consequences are:

1. **Free online authentication.** $(y_b(x), \tau_b(x))$ is a Λ-authenticated sharing of $g(x)$. Releasing an evaluation costs one element per party and the verification of arbitrarily many released points costs one aggregated check (Section 4.6).
2. **Moment fidelity.** Because the tag plane is Λ times the value plane, a party that misreports a value moment must supply Λ times its lie. This is the reduction that the previous literature obtained, if at all, from vector commitments (Section 5.2).
3. **Composability.** Gate outputs are already authenticated in the algebra in which the online phase computes, so the gate layer of Section 6 needs no conversion.

Figure 4 shows the same five phases as a system flow, with the data that crosses each boundary made explicit. Three features of that picture

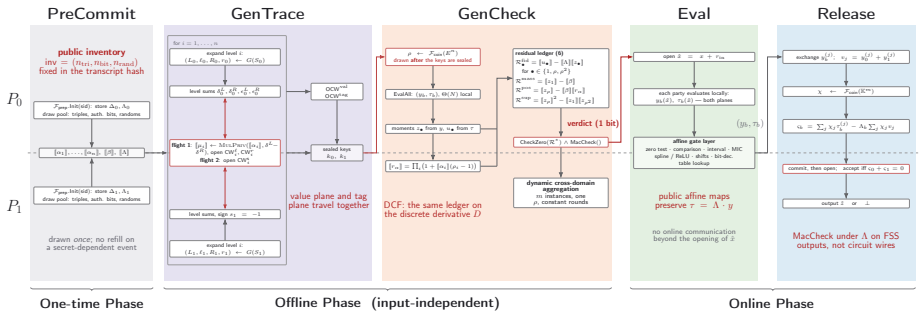


Fig. 4. System flow of Π_{VeriFSS}. Reading left to right: the correlation pool supplies the two authentication keys and the authenticated parameter handles; the distributed generator walks the tree level by level, opening one seed and two control correction words per level while the secret index bit is consumed inside a single authenticated multiplication; the sealed key pair is certified by a challenge drawn after sealing and a six-residual ledger closed by one aggregated zero check; and the certified key is then evaluated locally, with the tag plane travelling alongside the value plane so that every released share arrives already authenticated. The dashed return path is the only feedback edge in the system: a verdict, and nothing else, crosses from certification back to the key store.

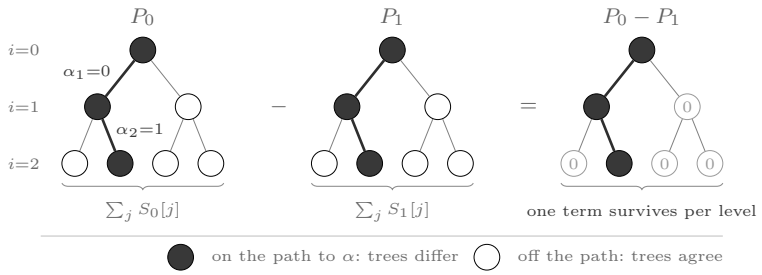
carry the design. The pool sits upstream of everything and is drawn from once, on a public inventory, so no later step can request material whose quantity depends on a secret. The two planes of the key travel together from generation through certification to release—they are never separated, which is why an evaluation share needs no accompanying proof. And certification returns a single bit: the key store learns whether to mark a key usable, and the evaluator learns nothing else about the audit. Everything below is an elaboration of that diagram.

The tree itself is written *additively* over $\mathbb{K}$. In characteristic two this is the classical XOR-based BGI tree; in odd characteristic the same code runs with $+$ in place of $\oplus$. Concretely, let $G : \mathbb{K} \to \mathbb{K}^2 \times \{0,1\}^2$ be a length-doubling pseudorandom generator with control bits, and let $\mathsf{Cv}_{\mathsf{val}}, \mathsf{Cv}_{\mathsf{tag}} : \mathbb{K} \to \mathbb{K}$ be two domain-separated output maps. Party P_b maintains, at level i, arrays $S_b \in \mathbb{K}^{2^i}$ and $T_b \in \mathbb{K}^{2^i}$ with the invariant

$$S_0[j] = S_1[j] \text{ and } T_0[j] = T_1[j] \;\text{ for } j \neq \alpha_{<i}, \qquad T_0[\alpha_{<i}] - T_1[\alpha_{<i}] = 1, \tag{6}$$

where $\alpha_{<i}$ is the length-i prefix of α. The invariant is the whole point: the sum of a party's level differs from its partner's sum by exactly the on-path contribution, so

$$\nabla L^{(i)} := \textstyle\sum_j L_0[j] - \sum_j L_1[j] = L_0[\alpha_{<i}] - L_1[\alpha_{<i}],$$



$$\nabla L^{(i)} = \sum_j L_0[j] - \sum_j L_1[j] = L_0[\alpha_{<i}] - L_1[\alpha_{<i}] \qquad \text{and likewise } \nabla R^{(i)}$$

$$\mathsf{CW}^{\mathsf{s}} = -\big(\nabla R + \alpha_i(\nabla L - \nabla R)\big), \qquad \mathsf{CW}^{\mathsf{t}}_L = 1 - \alpha_i - \nabla t_L, \qquad \mathsf{CW}^{\mathsf{t}}_R = \alpha_i - \nabla t_R$$

Fig. 5. The invariant behind GenTrace, and the reason the correction words are computable at all. Off-path nodes are expanded from the *same* seed in both trees, so they cancel node by node in the difference and exactly one term survives at each level. Each party therefore obtains ∇L and ∇R from a local sum over its own level, and every correction word is affine in those two quantities and in the secret index bit α_i—which is the only value that has to be consumed inside an authenticated multiplication.

Algorithm 1 $\Pi_{\mathsf{PreCommit}}$

Require: session identifier sid, public inventory bound $\mathsf{inv} = (n_{\mathrm{tri}}, n_{\mathrm{bit}}, n_{\mathrm{rand}})$
1: Call $\mathcal{F}_{\mathsf{prep}}.\mathsf{Init}(\mathsf{sid})$; P_b stores Δ_b, Λ_b.
2: Draw n_{tri} triples, n_{bit} authenticated bits and n_{rand} authenticated random values.
3: Both parties fix inv in the transcript hash before any key request is admitted.
Ensure: No pool unit is reused; no refill is triggered by a secret-dependent event.

and correction words become affine in privately held quantities (Figure 5).

4.2 Phase I: PreCommit

The inventory is fixed in advance and is a function of the public job description only. This is not a formality: a refill triggered by a secret-dependent event would create a leakage channel outside Π_{sf}, and the simulator of Theorem 2 would not go through.

4.3 Phase II: GenTrace for DPF

Algorithm 2 is complete: every message is listed, and the two **flight** markers are the only synchronisation points, so the protocol runs in $2n + 2$ rounds. Per level, each party sends exactly five elements of $\mathbb{K}$—the two Beaver masks d, e, the two control correction words, and the seed correction word—plus four elements outside the loop, for the multiplication $\Lambda\beta$ and the two output correction words.

Algorithm 2 $\Pi^{\mathsf{DPF}}_{\mathsf{GenTrace}}$ — run by P_b, $b \in \{0,1\}$

Require: $[\![\alpha_1]\!], \ldots, [\![\alpha_n]\!]$ (MSB first), $[\![\beta]\!]$, $[\![\Lambda]\!]$; sign $s_b = (-1)^b$
$[\![w]\!] \leftarrow [\![\Lambda]\!] \cdot [\![\beta]\!]$ ▷ one triple; $w = \Lambda\beta$
$S_b \leftarrow (\mathsf{s}_b)$ for $\mathsf{s}_b \leftarrow \mathbb{K}$; $T_b \leftarrow (1-b)$ ▷ root: $T_0 - T_1 = 1$
for $i = 1$ **to** n **do**
  $(L_b, \ell_b, R_b, r_b) \leftarrow G(S_b[j])_{j \in [2^{i-1}]}$ ▷ $\Theta(2^i)$ local PRG work
  $\delta_b^L \leftarrow s_b \sum_j L_b[j]$, $\delta_b^R \leftarrow s_b \sum_j R_b[j]$, $\epsilon_b^L \leftarrow s_b \sum_j \ell_b[j]$, $\epsilon_b^R \leftarrow s_b \sum_j r_b[j]$
  [0] **flight 1**
  $[\![\mu_i]\!] \leftarrow \textsc{MulPriv}\big([\![\alpha_i]\!],\, \delta_b^L - \delta_b^R\big)$ ▷ $\mu_i = \alpha_i(\nabla L - \nabla R)$; Alg. 3
  $\mathsf{CW}_i^\ell \leftarrow \mathsf{Open}\big((1-b) - \alpha_{i,b} - \epsilon_b^L\big)$, $\mathsf{CW}_i^r \leftarrow \mathsf{Open}\big(\alpha_{i,b} - \epsilon_b^R\big)$
  [0] **flight 2**
  $\mathsf{CW}_i^{\mathsf{s}} \leftarrow \mathsf{Open}\big(-\delta_b^R - \mu_{i,b}\big)$ ▷ $\mathsf{CW}_i^{\mathsf{s}} = -\big(\nabla R + \alpha_i(\nabla L - \nabla R)\big)$
  [0] **local update**
  for $j \in [2^{i-1}]$ **do**
    $S_b[2j] \leftarrow L_b[j] + T_b[j]\mathsf{CW}_i^{\mathsf{s}}$, $S_b[2j{+}1] \leftarrow R_b[j] + T_b[j]\mathsf{CW}_i^{\mathsf{s}}$
    $T_b[2j] \leftarrow \ell_b[j] + T_b[j]\mathsf{CW}_i^{\ell}$, $T_b[2j{+}1] \leftarrow r_b[j] + T_b[j]\mathsf{CW}_i^{r}$
  end for
end for
$c_b^{\mathrm{val}} \leftarrow s_b \sum_j \mathsf{Cv}_{\mathsf{val}}(S_b[j])$, $c_b^{\mathrm{tag}} \leftarrow s_b \sum_j \mathsf{Cv}_{\mathsf{tag}}(S_b[j])$
$\mathsf{OCW}^{\mathrm{val}} \leftarrow \mathsf{Open}(\beta_b - c_b^{\mathrm{val}})$, $\mathsf{OCW}^{\mathrm{tag}} \leftarrow \mathsf{Open}(w_b - c_b^{\mathrm{tag}})$
Ensure: $k_b = \big(b, \mathsf{s}_b, (\mathsf{CW}_i^{\mathsf{s}}, \mathsf{CW}_i^{\ell}, \mathsf{CW}_i^{r})_{i \in [n]}, \mathsf{OCW}^{\mathrm{val}}, \mathsf{OCW}^{\mathrm{tag}}\big)$

Algorithm 3 $\textsc{MulPriv}([\![a]\!], v_b)$: authenticated product of a shared value with a privately held one

Require: $[\![a]\!]$; P_b holds v_b, and $v = v_0 + v_1$
Take a triple $([\![u]\!], [\![z]\!], [\![uz]\!])$ from the pool.
$d \leftarrow \mathsf{Open}([\![a]\!] - [\![u]\!])$ ▷ authenticated: covered by $\mathsf{MacCheck}$
$e \leftarrow \mathsf{Open}_{\mathrm{raw}}(v_b - z_b)$ ▷ unauthenticated: see Lemma 8
return $[\![uz]\!] + d[\![z]\!] + e[\![u]\!] + de$

Proposition 1 (Correctness). *If both parties follow Algorithm 2, then Eq. (6) holds at every level, and the key pair satisfies $g = \beta e_\alpha$ and $h = \Lambda\beta e_\alpha = \Lambda g$.*

Proof. Induction on i. At the root the invariant is vacuous off-path and holds on-path by construction. Assume it at level $i-1$. For an off-path parent j, both parties hold equal seeds and equal control values, hence equal PRG outputs and equal corrections, so both children stay equal. For the on-path parent, write $\nabla L, \nabla R$ for its children's seed differences and $\nabla\ell, \nabla R$ for its control differences; the level sums equal these because all other terms cancel. If $\alpha_i = 0$ then $\mathsf{CW}_i^{\mathsf{s}} = -\nabla R$, so the right child (off path) has difference $\nabla R + \mathsf{CW}_i^{\mathsf{s}} = 0$ and the left child has difference $\nabla L - \nabla R$, which is pseudorandom; the control differences become $\nabla\ell + \mathsf{CW}_i^{\ell} = 1 - \alpha_i = 1$

Algorithm 4 $\Pi^{\mathsf{DCF}}_{\mathsf{GenTrace}}$ — differences from Algorithm 2

Precompute $[\![\alpha_i\beta]\!]$ and $[\![\alpha_i w]\!]$ for all i ▷ $2n$ triples, one flight
$A_b \leftarrow 0$, $A_b^{\mathrm{tag}} \leftarrow 0$ ▷ running accumulator shares
for $i = 1$ **to** n **do**
  $[\![\nu_i]\!] \leftarrow \textsc{MulPriv}([\![\alpha_i]\!], \nabla\omega_b^L - \nabla\omega_b^R)$; $\lambda_b \leftarrow \nabla\omega_b^R + \nu_{i,b}$ ▷ lose-branch share
  $\kappa_b \leftarrow \nabla\omega_b^L + \nabla\omega_b^R - \lambda_b$ ▷ keep-branch share
  $\mathsf{VCW}_i \leftarrow \mathsf{Open}_{\mathrm{raw}}\big((\alpha_i\beta)_b - A_b - \lambda_b\big)$
  $A_b \leftarrow A_b + \kappa_b + \big((\alpha_i\beta)_b - A_b - \lambda_b\big)$
  (identically for the tag plane, with w in place of β)
end for
$\mathsf{OCW}^{\mathrm{val}} \leftarrow \mathsf{Open}_{\mathrm{raw}}\big(-c_b^{\mathrm{val}} - A_b\big)$, $\mathsf{OCW}^{\mathrm{tag}} \leftarrow \mathsf{Open}_{\mathrm{raw}}\big(-c_b^{\mathrm{tag}} - A_b^{\mathrm{tag}}\big)$

on the left and $\nabla R + \mathsf{CW}_i^r = \alpha_i = 0$ on the right. The case $\alpha_i = 1$ is symmetric. At the leaves, $\mathsf{Cv}_{\mathsf{val}}$ of equal seeds cancels off path, so $g(x) = 0$ for $x \neq \alpha$ and $g(\alpha) = c^{\mathrm{val}} + \mathsf{OCW}^{\mathrm{val}} = \beta$; the tag plane is identical with $w = \Lambda\beta$ in place of β. □

4.4 Phase II′: GenTrace for DCF

The comparison function needs a value word at every level. Let the expander also output (ω_b^L, ω_b^R) per node, and let $\nabla\omega^L, \nabla\omega^R$ be their on-path differences. Define the *keep* and *lose* branches at level i by α_i, so that $\nabla\omega^{\mathrm{lose}} = \nabla\omega^R + \alpha_i(\nabla\omega^L - \nabla\omega^R)$ and $\nabla\omega^{\mathrm{keep}} = \nabla\omega^L + \nabla\omega^R - \nabla\omega^{\mathrm{lose}}$.

Proposition 2 (DCF correctness). *Let* Eval *accumulate* $\omega^{x_i} + T \cdot \mathsf{VCW}_i$ *along the path of* x *and add* $\mathsf{Cv}_{\mathsf{val}}(S) + T \cdot \mathsf{OCW}^{\mathrm{val}}$ *at the leaf. Then the key pair produced by Algorithm 4 satisfies* $g = f^{<}_{\alpha,\beta}$ *and* $h = \Lambda g$.

Proof. Let d be the first index at which x and α differ, if any. By Eq. (6) the contributions of levels $i > d$ vanish, the levels $i < d$ contribute $\nabla\omega^{\mathrm{keep},(i)} + \mathsf{VCW}_i$ each, and level d contributes $\nabla\omega^{\mathrm{lose},(d)} + \mathsf{VCW}_d$. Writing $A_m = \sum_{i \le m}(\nabla\omega^{\mathrm{keep},(i)} + \mathsf{VCW}_i)$ and substituting $\mathsf{VCW}_d = \alpha_d\beta - A_{d-1} - \nabla\omega^{\mathrm{lose},(d)}$ gives $g(x) = \alpha_d\beta$, which is β exactly when $x_d = 0, \alpha_d = 1$, i.e. when $x < \alpha$, and 0 when $x_d = 1, \alpha_d = 0$, i.e. when $x > \alpha$. If $x = \alpha$ the leaf term applies and $g(\alpha) = A_n + c^{\mathrm{val}} + \mathsf{OCW}^{\mathrm{val}} = 0$, matching the boundary convention. The tag plane is the same computation with $w = \Lambda\beta$. □

4.5 Phase III: GenCheck

After both keys are sealed, the parties draw $\rho \leftarrow E^n$ by a coin toss (or by hashing the transcript; see Remark 4) and each computes six full-domain moments.

Algorithm 5 $\Pi^{\mathsf{DPF}}_{\mathsf{GenCheck}}$

Require: sealed k_b; $[\![\alpha_i]\!], [\![\beta]\!], [\![\Lambda]\!]$

1: $\rho \leftarrow \mathcal{F}_{\mathsf{coin}}(E^n)$; $(r_x)_{x\in[N]} \leftarrow$ Eq. (2); $r_x^{(2)} \leftarrow r_x^2$
2: $(y_b, \tau_b) \leftarrow \mathsf{EvalAll}(b, k_b)$ ▷ $\Theta(N)$ local work
3: $z_\bullet^{(b)} \leftarrow \langle \mathbf{1}, y_b\rangle,\ \langle r, y_b\rangle,\ \langle r^{(2)}, y_b\rangle$; $u_\bullet^{(b)}$ likewise from τ_b
4: $[\![z_\bullet]\!] \leftarrow \textsc{Input}_0(z_\bullet^{(0)}) + \textsc{Input}_1(z_\bullet^{(1)})$; $[\![u_\bullet]\!]$ likewise ▷ one flight
5: $[\![r_\alpha]\!] \leftarrow \prod_i \big(1 + [\![\alpha_i]\!](\rho_i - 1)\big)$ ▷ $n-1$ triples, depth $\lceil \log_2 n \rceil$
6: **residual ledger**
   [1] $\mathcal{R}_\bullet^{\mathrm{fid}} \leftarrow [\![u_\bullet]\!] - [\![\Lambda]\!][\![z_\bullet]\!]$, $\bullet \in \{1, \rho, \rho^2\}$
   [1] $\mathcal{R}^{\mathrm{mass}} \leftarrow [\![z_1]\!] - [\![\beta]\!]$
   [1] $\mathcal{R}^{\mathrm{pos}} \leftarrow [\![z_\rho]\!] - [\![\beta]\!][\![r_\alpha]\!]$
   [1] $\mathcal{R}^{\mathrm{sup}} \leftarrow [\![z_\rho]\!]^2 - [\![z_1]\!][\![z_{\rho^2}]\!]$
7: **return** $\mathsf{CheckZero}(\mathcal{R}^*) \wedge \mathsf{MacCheck}()$

Algorithm 6 $\Pi^{\mathsf{DCF}}_{\mathsf{GenCheck}}$ — differences from Algorithm 5

1: Replace y_b, τ_b by $Dy_b, D\tau_b$ from Eq. (7).
2: $[\![e_0]\!] \leftarrow \prod_i (1 - [\![\alpha_i]\!])$; $[\![\hat\beta]\!] \leftarrow [\![\beta]\!](1 - [\![e_0]\!])$ ▷ $e_0 = \mathbb{1}\{\alpha = 0\}$
3: Derive $[\![\alpha_i']\!]$ for $\alpha' = \alpha - 1 \bmod N$ by a borrow chain: for i from the least significant bit, $[\![\alpha_i']\!] = [\![\alpha_i]\!] + [\![c]\!] - 2[\![\alpha_i]\!][\![c]\!]$, $[\![c]\!] \leftarrow [\![c]\!] - [\![\alpha_i]\!][\![c]\!]$, with c initialised to 1.
4: $\mathcal{R}^{\mathrm{mass}} \leftarrow [\![z_1]\!] - [\![\hat\beta]\!]$, $\mathcal{R}^{\mathrm{pos}} \leftarrow [\![z_\rho]\!] - [\![\hat\beta]\!][\![r_{\alpha'}]\!]$; $\mathcal{R}^{\mathrm{sup}}$ and $\mathcal{R}_\bullet^{\mathrm{fid}}$ unchanged.

For the comparison function the same ledger is applied to the discrete derivative. Let $D : \mathbb{K}^N \to \mathbb{K}^N$ be

$$(Dg)(x) = g(x) - g(x+1) \ \ (x < N-1), \qquad (Dg)(N-1) = g(N-1). \tag{7}$$

D is invertible, with $g(x) = \sum_{j \ge x} (Dg)(j)$; this is why no separate boundary residual is needed.

Remark 4 (Fiat–Shamir). ρ may be derived by hashing the sealed transcript instead of tossing coins, which removes one round. The final $\mathsf{CheckZero}$ still needs its commit-then-open structure: a rushing party must not see the honest check share before fixing its own. Grinding costs the usual factor q_{RO} in the soundness bound (Proposition 5).

4.6 Phase IV–V: evaluation and release

$\mathsf{Eval}(b, k_b, x)$ walks the path of x and returns the pair $(y_b(x), \tau_b(x))$; it is entirely local and is the ordinary FSS evaluation on two output planes. Release is one flight plus one aggregated check.

Algorithm 7 is the SPDZ $\mathsf{MacCheck}$ under the key Λ, applied to values that were never wires of an arithmetic protocol but outputs of an FSS key. That is exactly what the second plane buys.

Algorithm 7 Π_{Release} on a set of points from possibly different keys

Require: pairs $(y_b^{(j)}, \tau_b^{(j)})_{j\in[m]}$ from certified keys
1: Exchange $(y_b^{(j)})_j$; set $v_j \leftarrow y_0^{(j)} + y_1^{(j)}$ ▷ one flight
2: $\chi \leftarrow \mathcal{F}_{\mathsf{coin}}(\mathbb{K}^m)$; $\varsigma_b \leftarrow \sum_j \chi_j \tau_b^{(j)} - \Lambda_b \sum_j \chi_j v_j$
3: Commit to ς_b, then open; accept iff $\varsigma_0 + \varsigma_1 = 0$.

Table 1. Cost of one certified instance on a domain of size $N = 2^n$, in elements of $\mathbb{K}$ per party. "Pool" counts authenticated multiplication triples. The audit costs are independent of N in communication and linear in N in local work, matching generation.

Phase	Communication	Rounds	Local work	Triples
$\Pi^{\mathsf{DPF}}_{\mathsf{GenTrace}}$	$5n+4$	$2n+2$	$\Theta(N)$ PRG	$n+1$
$\Pi^{\mathsf{DCF}}_{\mathsf{GenTrace}}$	$15n+4$	$2n+3$	$\Theta(N)$ PRG	$5n+1$
$\Pi^{\mathsf{DPF}}_{\mathsf{GenCheck}}$	$2n+23$	$\lceil\log_2 n\rceil + 7$	$\Theta(N)$ mult	$n+5$
$\Pi^{\mathsf{DCF}}_{\mathsf{GenCheck}}$	$6n+23$	$n + 2\lceil\log_2 n\rceil + 6$	$\Theta(N)$ mult	$3n+5$
Eval per point	0	0	$O(n)$ PRG	0
Π_{Release} of m points	$m+2$	3	$O(m)$	0

4.7 Cost

Table 1 collects the counts. They are exact, not asymptotic: Section 7 reproduces every entry to the byte and to the round against an instrumented channel. Three points deserve emphasis.

First, certification communication does not depend on N. The audit reads the whole domain locally—which it must, since it certifies a full-domain statement—but says only $O(n)$ elements out loud. Its $\Theta(N)$ local pass is of the same order as the generation it certifies, because the distributed generator is itself full-domain. A reader with a logarithmic-work generator in mind should note the consequence: paired with such a generator the audit would dominate, and the construction would be attractive only where a full-domain pass is already being paid for—which is the regime of every dealer-free distributed generator we know of, and of the offset keys that the gate layer consumes.

Second, the DCF audit's extra n rounds come from the borrow chain that derives $\alpha - 1$. When many DCF instances are certified together the chains run in parallel, so the aggregate round count is again $O(n + \log n)$ regardless of how many keys are in flight.

Third, the key itself grows by only two field elements over the semi-honest dealer-free baseline for a DPF, since the tag plane adds one output correction word and the control words remain two-bit quantities. A DCF

pays more, because every level carries a value correction word in each plane; the measured figures are in Table 5.

5 Security Analysis

Throughout this section P_1 is the corrupted party (the other case is symmetric), $\mathbb{K}$ is the audit algebra of Section 2.4 with exceptional set E (so $E = \mathbb{K}$ in the field case), and (g, h) denote the two functions that the key pair sealed at the end of $\mathsf{GenTrace}$ actually realises. We write $\varepsilon_{\mathrm{mac}}$ for the soundness error of $\mathsf{MacCheck}$ and $\mathsf{CheckZero}$ over $\mathbb{K}$; the standard analysis gives $\varepsilon_{\mathrm{mac}} \le 2/|E|$.

The argument has four steps: the adversary's view is independent of Λ (§5.1); therefore it cannot move the audited value moments (§5.2); therefore the three shape equations pin the evaluation vector down (§5.3); and every deviation that can change the key is covered by one of these residuals, except for a single class whose effect is the modelled selective-failure predicate (§5.4). Section 5.6 assembles the simulator.

5.1 The generation view is independent of Λ

Lemma 2 (Λ-independence). *Let* view_1 *be the view of* P_1 *from the start of* $\mathsf{PreCommit}$ *up to and including the challenge* ρ*. Then there is a simulator that produces a distribution computationally indistinguishable from* view_1 *without using* Λ*; consequently, conditioned on* view_1*,* Λ *is computationally indistinguishable from uniform on* $\mathbb{K}$*.*

Proof. Λ enters view_1 in exactly three places. *(i)* The pool share Λ_1, which is uniform and independent of Λ by construction of $\mathcal{F}_{\mathsf{prep}}$. *(ii)* The authenticated wire $[\![w]\!]$, $w = \Lambda\beta$, which is never opened; the Beaver openings that produce it reveal $\Lambda - u$ and $\beta - z$ for fresh pool masks u, z, both uniform. *(iii)* The opened tag correction words. For the DPF these are the single value $\mathsf{OCW}^{\mathrm{tag}} = w - \sum_j \big(\mathsf{Cv}_{\mathsf{tag}}(S_0[j]) - \mathsf{Cv}_{\mathsf{tag}}(S_1[j])\big) = w - \mathsf{Cv}_{\mathsf{tag}}(S_0[\alpha]) + \mathsf{Cv}_{\mathsf{tag}}(S_1[\alpha])$ by Eq. (6); for the DCF they additionally include the n words $\mathsf{VCW}_i^{\mathrm{tag}}$, each of which contains the honest party's on-path value word $\omega_0^{\mathrm{lose},(i)}$. In every case the opened value is masked by a $\mathsf{Cv}_{\mathsf{tag}}$- or G-image of a seed that P_1 does not hold: $S_0[\alpha]$ is the honest party's on-path seed, and by Eq. (6) P_1's array agrees with P_0's everywhere except at that one index, whose position P_1 does not know. Replacing each such image by a uniform element of $\mathbb{K}$ is one application of the circular correlation robustness of π (Section 2.6), at cost $\varepsilon_{\mathrm{prg}}$; after the replacement,

each opened tag word is uniform and independent of Λ. Finally ρ is either public coins or a hash of the transcript, and in both cases is a function of values already argued to be Λ-free. □

5.2 Moment fidelity

This is the step that the previous literature could only obtain from a vector commitment with linear openings. Here it falls out of the tag plane.

Lemma 3 (Moment fidelity). *Fix* $\bullet \in \{1, \rho, \rho^2\}$ *and let* $Z_\bullet = \sum_x r_x^{(\bullet)} g(x)$ *be the true value moment of the sealed key pair. Let* $\hat{z}_\bullet$ *be the value that* GenCheck *audits. Then*

$$\Pr\big[\mathsf{GenCheck} = \mathsf{accept} \ \wedge \ \exists \bullet : \ \hat{z}_\bullet \neq Z_\bullet\big] \ \leq \ \frac{3}{|E|} + \varepsilon_{\mathrm{mac}} + \mathsf{negl}(\lambda).$$

Proof. Write $\hat{z}_\bullet = Z_\bullet + \delta_\bullet$ and $\hat{u}_\bullet = U_\bullet + \varepsilon_\bullet$, where $U_\bullet = \sum_x r_x^{(\bullet)} h(x)$ and $(\delta_\bullet, \varepsilon_\bullet)$ are the deviations P_1 introduces in its input gates. Acceptance of CheckZero implies, except with probability $\varepsilon_{\mathrm{mac}}$, that every residual is zero, in particular $\hat{u}_\bullet - \Lambda \hat{z}_\bullet = 0$. Let $\Psi = h - \Lambda g$ be the tag defect and $\Phi_\bullet = \sum_x r_x^{(\bullet)} \Psi(x)$. Then $U_\bullet - \Lambda Z_\bullet = \Phi_\bullet$ and the residual reads

$$\Phi_\bullet + \varepsilon_\bullet \ = \ \Lambda\, \delta_\bullet. \tag{8}$$

The left-hand side is a function of view_1 only: Ψ is fixed by the sealed keys, ρ is in the view, and $\varepsilon_\bullet$ is chosen by P_1. By Lemma 2, Λ is (computationally) uniform given that view. Suppose $\delta_\bullet \neq 0$ and write $\delta_\bullet = 2^t \delta'$ with δ' a unit and $t < m$ in the ring case, $t = 0$ in the field case. Then $\Lambda \delta_\bullet$ is uniform on the ideal $2^t \mathbb{K}$, which has at least $|E|$ elements, so Eq. (8) holds with probability at most $1/|E|$. A union bound over the three moments gives the claim. □

Remark 5 (What is not *claimed).* Lemma 3 says nothing about Ψ. Indeed P_1 can always satisfy Eq. (8) with $\delta_\bullet = 0$ and $\varepsilon_\bullet = -\Phi_\bullet$, since it knows Ψ and ρ; a defective tag plane therefore survives GenCheck. This is harmless and we prove it so: Lemma 9 shows that at release time a tag defect forces P_1 either to patch its own tag—in which case the released *value* is still correct—or to abort. The artifact exhibits both behaviours (Section 7.10).

5.3 Shape identification

Fidelity reduces the audit to a statement about a vector fixed before the challenge. The following coefficient identity is the engine; it is characteristic-free and, unlike the usual Cauchy–Schwarz-style manipulation, needs no division by 2.

Lemma 4 (Support coefficient). *Let $\mathbb{K}$ be a commutative ring, $g \in \mathbb{K}^N$, and let $z_1 = \sum_x g_x$, $z_\rho = \sum_x r_x(\rho) g_x$, $z_{\rho^2} = \sum_x r_x(\rho)^2 g_x$ with r_x as in Eq. (2). Regard*

$$T(\rho) \;:=\; z_\rho^2 - z_1 z_{\rho^2} \;\in\; \mathbb{K}[\rho_1, \ldots, \rho_n]$$

as a polynomial. Then for every $x_0 \in [N]$ the coefficient of the monomial $\prod_{i \in x_0} \rho_i^2$ in T equals $g_{x_0}(g_{x_0} - z_1)$.

Proof. Expanding, $T = \sum_{x,y} g_x g_y r_x r_y - \left(\sum_x g_x\right) \sum_y g_y r_y^2$. In the first sum $r_x r_y = \prod_i \rho_i^{\mathbb{1}[i \in x] + \mathbb{1}[i \in y]}$, whose exponents lie in $\{0, 1, 2\}$; it equals $\prod_{i \in x_0} \rho_i^2$ only if every $i \in x_0$ lies in $x \cap y$ and no $i \notin x_0$ lies in $x \cup y$, that is only for $x = y = x_0$, contributing $g_{x_0}^2$. In the second sum $r_y^2 = \prod_{i \in y} \rho_i^2$ matches only for $y = x_0$, contributing $-z_1 g_{x_0}$. □

Lemma 5 (Shape identification over a field). *Let $\mathbb{F}$ be a field, let $g \in \mathbb{F}^N$ and $\alpha \in [N]$, $\beta \in \mathbb{F}$ be fixed before $\rho \leftarrow \mathbb{F}^n$ is drawn, and let* ACC *be the event*

$$z_1 = \beta \;\wedge\; z_\rho = \beta\, r_\alpha(\rho) \;\wedge\; z_\rho^2 = z_1 z_{\rho^2}.$$

If $g \neq \beta e_\alpha$ then $\Pr[\mathsf{ACC}] \leq 2n/|\mathbb{F}|$.

Proof. Assume the mass equation, so $z_1 = \beta$; otherwise $\Pr[\mathsf{ACC}] = 0$.

Case A: $g_{x_0}(g_{x_0} - \beta) \neq 0$ for some x_0. By Lemma 4, T is a nonzero polynomial of total degree at most $2n$. The support equation asserts $T(\rho) = 0$, so by Lemma 1 it holds with probability at most $2n/|\mathbb{F}|$.

Case B: $g_x(g_x - \beta) = 0$ for all x. Over a field this means $g_x \in \{0, \beta\}$ for every x. If $\beta = 0$ then $g = 0 = \beta e_\alpha$, contradicting the hypothesis, so $\beta \neq 0$ and $g = \beta \cdot \mathbf{1}_S$ with $S = \mathrm{supp}(g)$. Here $S \neq \emptyset$, since $S = \emptyset$ would give $z_1 = 0 \neq \beta$ and the mass equation already fails; and $S \neq \{\alpha\}$ by hypothesis. The position equation asserts $\beta \sum_{x \in S} r_x(\rho) = \beta\, r_\alpha(\rho)$, i.e. $P(\rho) = 0$ for $P = \sum_{x \in S} r_x - r_\alpha$. The r_x are pairwise distinct multilinear monomials, so the coefficients of P lie in $\{0, \pm 1\}$ and $P \neq 0$: if $\alpha \notin S$ the monomial r_α survives with coefficient -1, and if $\alpha \in S$ then $|S| \geq 2$ because $S \neq \{\alpha\}$, so some r_x, $x \in S \setminus \{\alpha\}$, survives with coefficient 1. Since $\deg P \leq n$, Lemma 1 bounds $\Pr[P(\rho) = 0]$ by $n/|\mathbb{F}|$. □

Remark 6 (Why all three equations are needed). Case B is not vacuous. In characteristic two, $g = \beta(e_{x_1} + e_{x_2} + e_{x_3})$ has $z_1 = 3\beta = \beta$ and satisfies the support equation identically—every nonzero entry equals z_1, so all coefficients in Lemma 4 vanish—yet g is not a point function. Only the position equation rejects it. Conversely, g with two unequal nonzero entries

summing to β passes mass and can pass position only by accident; it is the support equation that rejects it. And $\beta = 0$ with $g \neq 0$ falls in Case A precisely because $g_x^2 \neq 0$ for a field. The artifact instantiates all four situations and observes the predicted rejections (Section 7.10).

Lemma 6 (Shape identification over a Galois ring). *Let* $\mathbb{K} = \mathrm{GR}(2^m, d)$ *with* $m = k + s$ *and* $s \geq k$*, let payloads lie in* $\mathbb{Z}_{2^m} \subseteq \mathbb{K}$ *and let* E *be the Teichmüller exceptional set. If* $g \not\equiv \beta e_\alpha \pmod{2^k}$ *then* $\Pr_{\rho \leftarrow E^n}[\mathsf{ACC}] \leq 2n/2^d$.

Proof. As before, assume $z_1 = \beta$. Case A is unchanged: if some coefficient $g_{x_0}(g_{x_0} - \beta)$ is nonzero then $T \neq 0$ and Lemma 1 applies over E, giving $2n/2^d$.

Case B now reads $g_x(g_x - \beta) = 0$ in $\mathbb{Z}_{2^m}$ for all x, which no longer forces $g_x \in \{0, \beta\}$. Write ν_2 for the 2-adic valuation with $\nu_2(0) = m$, and put $v = \nu_2(\beta)$, $a = \nu_2(g_x)$. If $a < v$ then $\nu_2(g_x - \beta) = a$ and $2a \geq m$, so $a \geq \lceil m/2 \rceil \geq k$, contradicting $a < v \leq k - 1$ unless $v \geq k$. If $a > v$ then $\nu_2(g_x - \beta) = v$ and $a + v \geq m$, so $a \geq m - v \geq s + 1 > k$. If $a = v$ then $\nu_2(g_x - \beta) \geq m - v \geq s + 1 > k$. Hence in all cases $g_x \equiv 0$ or $g_x \equiv \beta \pmod{2^k}$; and if $v \geq k$, i.e. $\beta \equiv 0 \pmod{2^k}$, then every branch forces $a \geq k$, so $g \equiv 0 \equiv \beta e_\alpha \pmod{2^k}$, contradicting the hypothesis. Therefore $\beta \not\equiv 0$, $g \equiv \beta \mathbf{1}_S \pmod{2^k}$ for some $S \neq \{\alpha\}$, and the position equation gives $\beta\, P(\rho) \equiv 0 \pmod{2^k}$ with P as in Lemma 5. Dividing by 2^v leaves $P(\rho) \equiv 0 \pmod{2^{k-v}}$ with $k - v \geq 1$; P has unit coefficients, hence is nonzero modulo 2, and Lemma 1 bounds the probability by $n/2^d$. □

Lemma 7 (Derivative reduction for DCF). *Let* D *be the operator of Eq. (7) and let* $e_0 = \mathbb{1}\{\alpha = 0\}$, $\alpha' = \alpha - 1 \bmod N$. *Then* D *is a linear bijection of* $\mathbb{K}^N$ *and* $D f^{<}_{\alpha,\beta} = \beta(1 - e_0)\, e_{\alpha'}$. *Consequently* $g = f^{<}_{\alpha,\beta}$ *if and only if* $Dg = \hat{\beta}\, e_{\alpha'}$ *with* $\hat{\beta} = \beta(1 - e_0)$, *and Lemmas 5 and 6 apply verbatim to* Dg.

Proof. D is upper triangular with unit diagonal in the basis (e_x), hence bijective, with inverse $g(x) = \sum_{j \geq x} (Dg)(j)$. For $x < N - 1$, $(Df^{<})(x) = \beta(\mathbb{1}\{x < \alpha\} - \mathbb{1}\{x + 1 < \alpha\}) = \beta\, \mathbb{1}\{x = \alpha - 1\}$, and $(Df^{<})(N - 1) = f^{<}(N - 1) = 0$ because $\alpha \leq N - 1$. If $\alpha = 0$ then $f^{<} \equiv 0$ and both sides vanish, which is what the factor $1 - e_0$ records; the index $\alpha' = N - 1$ is then irrelevant since its coefficient is 0. □

Lemma 7 replaces the ad-hoc combination of a derivative test with a separate boundary residual: because D is invertible on the whole of $\mathbb{K}^N$, no additive constant can escape, and the DCF audit is literally the DPF audit composed with a local linear map.

5.4 Coverage of adversarial deviations

Lemma 8 (Coverage). *Consider an execution of* Π_{GenTrace} *in which* P_1 *deviates. Every deviation belongs to exactly one of the following classes.*

(a) A deviation in an authenticated opening (the mask d *of Algorithm 3, or any opening inside* GenCheck*). It is detected by* MacCheck *except with probability* $\varepsilon_{\mathrm{mac}}$*, independently of* θ*.*

(b) A deviation in a raw opening of $\mathsf{CW}_i^{\mathsf{s}}$*,* CW_i^{ℓ}*,* CW_i^{r}*,* VCW_i*,* $\mathsf{VCW}_i^{\mathrm{tag}}$*,* $\mathsf{OCW}^{\mathrm{val}}$ *or* $\mathsf{OCW}^{\mathrm{tag}}$*. Its effect on* (g, h) *is an additive change that does not depend on* θ*, and the resulting* g *differs from* f_θ*.*

(c) A deviation of magnitude δ *in the raw opening* e *of Algorithm 3 at level* i*. Its effect on the level's correction word is* $-\alpha_i\delta$*; the key is unchanged if* $\alpha_i = 0$ *and altered as in class (b) if* $\alpha_i = 1$*.*

(d) An additive error on P_1*'s own* $\mathcal{F}_{\mathsf{prep}}$ *shares, which* $\mathcal{F}_{\mathsf{prep}}$ *already permits and which is equivalent to a class (a) or (c) deviation.*

No other deviation affects the sealed keys.

Proof. P_1's messages during GenTrace are exactly: its share of each authenticated opening, its share of each raw opening, and its $\mathcal{F}_{\mathsf{prep}}$ interaction; the tree update is local and deterministic given the opened correction words, and the two parties compute the same correction words because each reconstructs own + received. Class (a) is the standard MacCheck guarantee. For class (b), inspect Algorithms 2 and 4: each listed word enters the key and the tree update additively with public coefficients, so an error δ propagates to a θ-independent perturbation. Concretely, an error in $\mathsf{OCW}^{\mathrm{val}}$ shifts $g(\alpha)$ by δ and hence violates the mass equation; an error in $\mathsf{CW}_i^{\mathsf{s}}$ leaves the lose branch uncancelled, so $\mathrm{supp}(g)$ acquires an entire subtree; an error in CW_i^{ℓ} or CW_i^{r} breaks Eq. (6) and makes $\mathrm{supp}(g)$ empty or of size at least two. Class (c) is the only place where a message is consumed multiplicatively: substituting $e + \delta$ in Algorithm 3 yields the product $\alpha_i(v + \delta)$, since $c + dz + (e + \delta)u + d(e + \delta) = \alpha_i v + \alpha_i\delta$. □

Corollary 1 (Structure of the abort). *Let* $S \subseteq [n]$ *be the set of levels at which* P_1 *mounts a class (c) deviation with* $\delta \neq 0$*, and suppose* P_1 *mounts no class (a) or (b) deviation. Then the sealed key pair is honest if* $\pi_S(\theta) = \bigwedge_{i \in S}(\alpha_i = 0)$ *holds, and otherwise differs from an honest pair as in class (b). If* P_1 *mounts any class (a) or (b) deviation, then the sealed pair differs from an honest pair, so by Definition 3 the probability that* GenCheck *nevertheless accepts is at most* $\varepsilon_{\mathrm{gen}}$ *of Theorem 1, whatever* θ *is.*

5.5 Generation soundness and report binding

Theorem 1 (Generation soundness). Π_{VeriFSS} *satisfies Definition 3 with*

$$\varepsilon_{\mathrm{gen}} \;\le\; \frac{2n+4}{|E|} + 2\varepsilon_{\mathrm{mac}} + \mathsf{negl}(\lambda),$$

for both the DPF and the DCF instantiation, and over both the field and the Galois-ring algebras.

Proof. Suppose $\mathsf{GenCheck}$ accepts and $g \neq f_{\theta^*}$. By Lemma 3, except with probability $3/|E| + \varepsilon_{\mathrm{mac}} + \mathsf{negl}$, the audited moments are the true moments of g. $\mathsf{CheckZero}$ accepts a nonzero residual vector only if the random combination vanishes, which costs a further $1/|E|$, or if a MAC is forged, which costs $\varepsilon_{\mathrm{mac}}$. Hence the three shape equations hold for the true moments, and Lemma 5 (respectively Lemma 6) bounds their joint probability by $2n/|E|$ when $g \neq \beta e_\alpha$. For the DCF, Lemma 7 first replaces g by Dg and β by $\hat{\beta}$; the authenticated derivations of $\hat{\beta}$ and of the bits of $\alpha - 1$ are ordinary arithmetic in the authenticated layer and are covered by class (a). Summing, $\varepsilon_{\mathrm{gen}} \le 3/|E| + 1/|E| + 2n/|E| + 2\varepsilon_{\mathrm{mac}} + \mathsf{negl}$. □

Lemma 9 (Report binding). *Fix a session with a single output key Λ, as Algorithm 7 prescribes. Then Π_{Release} satisfies Definition 4 with $\varepsilon_{\mathrm{rep}} \le 2/|E| + \mathsf{negl}(\lambda)$ for any number m of released points drawn from any number of certified keys, and regardless of whether the tag plane of those keys is correct.*

Proof. Let P_1 release $y_1(x_j) + d_j$ and $\tau_1(x_j) + e_j$. Acceptance means

$$\sum_j \chi_j\big(h(x_j) + e_j\big) = \Lambda \sum_j \chi_j\big(g(x_j) + d_j\big),$$

i.e. $\sum_j \chi_j \Psi(x_j) + \sum_j \chi_j e_j = \Lambda \sum_j \chi_j d_j$ with $\Psi = h - \Lambda g$ the tag defect. Suppose some $d_j \neq 0$. The coefficients χ are drawn after the values are exchanged, so $\Pr_\chi[\sum_j \chi_j d_j = 0] \le 1/|E|$; conditioned on $\sum_j \chi_j d_j \neq 0$, Lemma 2 and the argument of Lemma 3 give probability at most $1/|E|$. Hence all $d_j = 0$ except with probability $2/|E| + \mathsf{negl}$, and the honest party outputs $g(x_j)$, which equals $f_{\theta^*}(x_j)$ by Theorem 1. Note that the argument never used $\Psi = 0$: a defective tag plane only forces P_1 to choose $e_j = -\Psi(x_j)$, which it can do, or to abort. □

5.6 Simulation

Theorem 2 (Main). *Assume G is a correlation-robust length-doubling pseudorandom generator and* Com *is a computationally hiding and binding commitment. Then* Π_{VeriFSS} *securely realises* $\mathcal{F}_{\mathsf{aFSS}}$ *of Figure 2 in the* $(\mathcal{F}_{\mathsf{prep}}, \mathcal{F}_{\mathsf{coin}})$*-hybrid model against a static malicious corruption of one party, with distinguishing advantage at most*

$$\varepsilon_{\mathrm{gen}} + \varepsilon_{\mathrm{rep}} + \varepsilon_{\mathrm{prg}} + \varepsilon_{\mathrm{com}}.$$

Proof (Proof sketch; the full hybrid argument is Appendix B). The simulator $\mathcal{S}$ runs $\mathcal{A}$ internally and plays $\mathcal{F}_{\mathsf{prep}}$, so it knows every pool value and every additive error $\mathcal{A}$ requests.

Generation. $\mathcal{S}$ simulates the honest party's messages. Each raw opening carries P_0's private level sum, which by Lemma 2's hybrid is pseudorandom given $\mathcal{A}$'s view; $\mathcal{S}$ therefore samples every correction word uniformly, and each authenticated opening is a pool mask that $\mathcal{S}$ chose. $\mathcal{S}$ records the deviation vector and classifies it by Lemma 8. Let S be the set of levels carrying a class (c) deviation. $\mathcal{S}$ submits π_S to $\mathcal{F}_{\mathsf{aFSS}}$ and receives $\pi_S(\theta)$. If $\pi_S(\theta) = 0$, or if any class (a) or (b) deviation was recorded, $\mathcal{S}$ makes the simulated GenCheck reject and instructs $\mathcal{F}_{\mathsf{aFSS}}$ to abort; by Corollary 1 and Theorem 1 this matches the real execution except with probability $\varepsilon_{\mathrm{gen}}$. Otherwise the real key pair is honest and $\mathcal{S}$ makes GenCheck accept.

Certification. $\mathcal{S}$ knows $\mathcal{A}$'s simulated key, hence its moments; it computes P_0's moment shares so that the six residuals vanish, and simulates the input gates with the pool masks it chose. This is a perfect simulation because the input gates open only masked values.

Release. On an evaluation request $\mathcal{F}_{\mathsf{aFSS}}$ returns $\{v_j\}$ to $\mathcal{S}$. $\mathcal{S}$ sets P_0's released shares to $v_j - y_1(x_j)$ and its tag shares to $\Lambda v_j - \tau_1(x_j)$, using the simulated key of $\mathcal{A}$ and the Λ it sampled. These are distributed exactly as in the real execution. $\mathcal{S}$ then runs the aggregated check; by Lemma 9 it accepts precisely when $\mathcal{A}$ released the correct values, so $\mathcal{S}$ forwards deliver to $\mathcal{F}_{\mathsf{aFSS}}$ in that case and abort otherwise.

The only steps at which the real and ideal executions diverge are the PRG replacement of Lemma 2, the commitment binding in CheckZero and Π_{Release}, and the two soundness events of Theorem 1 and Lemma 9. □

5.7 The residual leakage, quantified

Class (c) of Lemma 8 is the only source of leakage, and Corollary 1 says exactly what it gives: the value of a conjunction $\pi_S(\theta) = \bigwedge_{i \in S}(\alpha_i = 0)$, at

the price of an abort whenever the conjunction fails. Three observations bound its practical significance.

Proposition 3 (The predicate class is tight). *For every $S \subseteq [n]$ there is an efficient adversary that causes* GenCheck *to accept exactly when $\pi_S(\theta) = 1$ and to reject otherwise, and that learns nothing else. Hence no subclass of Π_{sf} suffices in Figure 2.*

Proof. Let $\mathcal{A}$ inject a class (c) deviation with an arbitrary nonzero δ_i at each level $i \in S$ and behave honestly elsewhere. By Lemma 8, the level-i correction word is perturbed by $-\alpha_i \delta_i$, so the sealed key equals the honest key if and only if $\alpha_i = 0$ for every $i \in S$. In that case GenCheck accepts with probability 1; otherwise the key differs from an honest key as in class (b) and GenCheck rejects except with probability $\varepsilon_{\mathrm{gen}}$. Since the deviations are chosen before any challenge and their only effect is through α, the verdict is a deterministic function of $\pi_S(\theta)$. □

Second, the two deployment settings differ and must be kept apart. In the gate layer of Section 6 the index α is the *offset randomness* r_{in} of a single gate instance, drawn uniformly from $\mathcal{F}_{\mathsf{prep}}$ and used once. There Proposition 4 applies verbatim, the leaked bit is a bit of a one-time mask, and a detected attempt destroys the whole preprocessing session. In the bare FSS setting, where α is a long-lived user secret with an arbitrary distribution, the $2^{-|S|}$ bound is replaced by $\Pr[\pi_S(\alpha) = 1]$, which the adversary may be able to make large by choosing S well; the leakage is then a genuine single-bit oracle on the secret and should be treated as such. We recommend the construction for the first setting and flag the second.

Proposition 4 (Harvesting cost, uniform index). *Suppose the parties certify m instances under one aggregated* CheckZero *(Section 6.5) and the indices are uniform. An adversary that attempts to learn π_S on one instance succeeds with probability $2^{-|S|}$ and with the remaining probability destroys all m instances of the session. The expected number of certified keys obtained per leaked bit is therefore $m/2$ at $|S| = 1$ and decays geometrically in $|S|$.*

Proof. Immediate from Proposition 3 and the fact that CheckZero is taken over the concatenated ledger, so one rejected residual rejects the whole session. □

Third, the leakage can be removed entirely, at a price we state rather than hide. Class (c) exists because the party's level sums are private values

that nothing binds to its tree. Binding them requires proving relations about the images of π, i.e. evaluating the key-generation circuit inside a maliciously secure two-party computation. That is a polynomial-size circuit—about n invocations of the block cipher plus $O(n\lambda)$ Boolean gates—so a leakage-free variant is *implied* by generic maliciously secure two-party computation of that circuit, replacing a $\Theta(N)$ local pass and $O(n)$ elements of communication by $O(n)$ garbled block-cipher evaluations. We do not construct it here, and we do not claim a leakage-free scheme. We regard the trade as unattractive in practice and record it for completeness.

5.8 Fiat–Shamir and aggregation

Proposition 5 (Fiat–Shamir loss). *Suppose ρ is derived by a random oracle from the sealed transcript. Fix an adversary that produces at most q* distinct sealed transcripts*, i.e. at most q candidate key pairs, and makes at most q_{RO} oracle queries. Then the bound of Theorem 1 degrades to $q \cdot \varepsilon_{\mathrm{gen}} + q_{\mathrm{RO}}^2/2^{L_H}$ for an L_H-bit digest.*

Proof. The challenge is a function of the sealed transcript, so re-querying the oracle without changing the transcript returns the same ρ; only a fresh transcript yields a fresh challenge. Each of the q transcripts is an independent instance of Theorem 1, and a union bound gives the first term; the second bounds a digest collision. □

We state the loss in terms of q rather than q_{RO} because the two are far apart here: producing a fresh sealed transcript means re-running a $\Theta(N)$ generation, so an adversary with work budget T obtains $q \approx T/\Theta(N)$ attempts, not q_{RO}. In the interactive profile, where ρ comes from a coin toss, the loss disappears. Either way the commit-then-open structure of $\mathsf{CheckZero}$ remains necessary: Fiat–Shamir removes the coin toss, not the need to fix a rushing party's final check share before the honest one is revealed.

Proposition 6 (Aggregation). *Let $\mathcal{L}_1, \ldots, \mathcal{L}_m$ be residual ledgers of independent instances, over possibly different domains, families and payload algebras sharing one Δ and one Λ. If every ledger is sealed before the common challenge and the concatenation order is fixed in the transcript hash, then $\mathsf{CheckZero}(\mathcal{L}_1 \| \cdots \| \mathcal{L}_m)$ accepts a nonzero concatenation with probability at most $1/|E| + \varepsilon_{\mathrm{mac}}$, and the resulting generation-soundness bound is $\max_j \varepsilon_{\mathrm{gen}}^{(j)} + 1/|E| + \varepsilon_{\mathrm{mac}}$.*

Proof. The random combination of a nonzero vector over $\mathbb{K}$ vanishes with probability at most $1/|E|$; the per-instance analyses are unchanged because each instance sealed its own keys before the common ρ was drawn. Aggregation does not add error terms per instance, so the union is over the instance-specific identification events only, of which at most one is triggered by a given deviation. □

Remark 7 (Sharing the challenge across domains). Instances of different domain sizes reuse a prefix of one challenge vector: instance j takes $\rho_{1..n_j}$ from a uniform $\rho \in E^{\max_j n_j}$. Soundness is unaffected, because Lemma 5 is applied independently to each instance's own n_j variables and a uniform vector restricts to a uniform prefix.

6 From Certified DPF/DCF to General FSS Gates

Boyle, Chandran, Gilboa, Gupta, Ishai, Kumar and Rathee [3] turned FSS into a general-purpose secure-computation layer by evaluating a gate G through an FSS scheme for the *offset family* $G^{[r_{\text{in}},r_{\text{out}}]}(\hat{x}) = G(\hat{x}-r_{\text{in}})+r_{\text{out}}$, where $\hat{x} = x + r_{\text{in}}$ is public. This section shows that the whole catalogue lifts to the dealer-free actively secure setting for free, and identifies one gate that is specific to it.

6.1 Authenticated offset gates

Definition 6 (Authenticated offset gate). *An authenticated offset gate for $G : [N] \to \mathbb{K}^\ell$ consists of $\mathcal{F}_{\mathsf{prep}}$-supplied authenticated offsets $[\![r_{\text{in}}]\!]$ (in bit form) and $[\![r_{\text{out}}]\!]$, a tuple of certified DPF/DCF instances, and a deterministic local algorithm that maps the public $\hat{x}$ and the local key material to a pair $(o_b, \varpi_b) \in \mathbb{K}^\ell \times \mathbb{K}^\ell$ with*

$$o_0 + o_1 = G(x) + r_{\text{out}}, \qquad \varpi_0 + \varpi_1 = \Lambda\big(G(x) + r_{\text{out}}\big).$$

Definition 7 (Affine FSS gate). *A gate for $G : [N] \to \mathbb{K}^\ell$ is* affine *with arity C if there exist*

- index maps $\phi_1, \ldots, \phi_C : [N] \to [N]$ *computable from the public masked input alone;*
- coefficient maps $A_1, \ldots, A_C : [N] \to \mathbb{K}^{\ell\times 1}$ *and* $B : [N] \to \mathbb{K}^{\ell\times q}$*, again functions of the public masked input alone; and*
- *a vector $\zeta \in \mathbb{K}^q$ of* key constants*, each of which is a function of the offsets $r_{\text{in}}, r_{\text{out}}$ and public data only, and is supplied in the key as a Λ-authenticated sharing,*

such that for every x and every honest key tuple $(k^{(1)}, \ldots, k^{(C)})$ for the families $\mathcal{F}_1, \ldots, \mathcal{F}_C$,

$$\sum_{c=1}^{C} A_c(\hat{x})\, f_{\theta_c}\big(\phi_c(\hat{x})\big) \;+\; B(\hat{x})\,\zeta \;=\; G(x) + r_{\mathrm{out}}.$$

No coefficient may depend on a secret; secret dependence enters only through ζ and through the indices θ_c of the instances.

Definition 7 is exactly wide enough. It admits the spline, whose piece coefficients depend on r_{in}, because those coefficients are key constants rather than coefficient maps; and it excludes a gate that multiplies two FSS outputs, because such a gate is quadratic in the f_{θ_c}.

Theorem 3 (Gate lifting). *Let $\mathcal{G}$ be an affine FSS gate of arity C in the sense of Definition 7. Suppose every instance $k^{(c)}$ is certified by Π_{GenCheck} and every key constant is Λ-authenticated. Then party P_b's local outputs*

$$o_b = \textstyle\sum_c A_c(\hat{x})\, \mathsf{Eval}^{\mathrm{val}}\big(b, k_b^{(c)}, \phi_c(\hat{x})\big) + B(\hat{x})\,\zeta_b,$$
$$\varpi_b = \textstyle\sum_c A_c(\hat{x})\, \mathsf{Eval}^{\mathrm{tag}}\big(b, k_b^{(c)}, \phi_c(\hat{x})\big) + B(\hat{x})\,\varsigma_b,$$

where (ζ_b, ς_b) is the Λ-sharing of ζ, form an authenticated offset gate in the sense of Definition 6. The lift adds no online communication and no online round; its offline cost is the certification of the C instances plus one authenticated multiplication per key constant.

Proof. Certification gives $g^{(c)} = f_{\theta_c}$ on the whole domain (Theorem 1), and Lemma 9 binds every released value to it. Summing the two plane-wise expressions, $o_0 + o_1 = \sum_c A_c\, g^{(c)}(\phi_c(\hat{x})) + B\zeta = G(x) + r_{\mathrm{out}}$ by Definition 7, and $\varpi_0 + \varpi_1 = \sum_c A_c\, h^{(c)}(\phi_c(\hat{x})) + B\Lambda\zeta = \Lambda(o_0 + o_1)$ because $h^{(c)} = \Lambda g^{(c)}$ pointwise and the coefficient maps are public. Public affine maps therefore commute with the tag plane, which is the whole content of the statement. □

Theorem 3 covers, verbatim, the gates of [3]: zero test, integer comparison, interval containment, multiple interval containment, splines and ReLU, logical and arithmetic right shift, bit decomposition, and the public-interval variants; it also covers the table-lookup gates of Pika [44] and Grotto [42], and the mixed-mode primitives of LLAMA [29] and SIGMA [28]. What it does not cover is a gate whose evaluation multiplies two FSS outputs together; those need an authenticated Beaver triple, which the pool supplies and which preserves the invariant by the usual argument.

Algorithm 8 ShiftIndex($[\![r_1]\!], \ldots, [\![r_n]\!]; c$)

Require: MSB-first authenticated bits of r; public constant c
$[\![\mathsf{ca}]\!] \leftarrow 0$
for $i = n$ **down to** 1 **do**
  $[\![p]\!] \leftarrow [\![r_i]\!][\![\mathsf{ca}]\!]$ ▷ one triple
  if $c_i = 0$ **then**
    $[\![s_i]\!] \leftarrow [\![r_i]\!] + [\![\mathsf{ca}]\!] - 2[\![p]\!]$; $[\![\mathsf{ca}]\!] \leftarrow [\![p]\!]$
  else
    $[\![s_i]\!] \leftarrow 1 - ([\![r_i]\!] + [\![\mathsf{ca}]\!] - 2[\![p]\!])$; $[\![\mathsf{ca}]\!] \leftarrow [\![r_i]\!] + [\![\mathsf{ca}]\!] - [\![p]\!]$
  end if
end for
Ensure: bits of $(r + c)$ mod N, and the carry-out $[\![\mathsf{ca}]\!] = [\![\mathbb{1}\{r + c \geq N\}]\!]$

6.2 Deriving the offsets

The offset r_{in} is the special index of the underlying instances, so it must reach $\mathsf{GenTrace}$ in authenticated bit form. $\mathcal{F}_{\mathsf{prep}}$ supplies n authenticated bits; gates whose thresholds are public constants need the shifted indices $(r_{\text{in}} + c)$ mod N, which we derive with a ripple-carry adder inside the authenticated layer.

The constant 2 in Algorithm 8 is the field element $1+1$, which vanishes in characteristic two; writing the literal integer would denote the polynomial X in $\mathrm{GF}(2^\kappa)$ and silently break the gate. We flag this because it is exactly the kind of mismatch that the artifact caught during development. The carry-out is not a by-product to be discarded: it is the *wrap correction* that the comparison identity below needs.

6.3 Interval containment and its descendants

Write $A_c(\hat{x}) = \mathbb{1}\{\hat{x} < (r_{\text{in}} + c) \bmod N\}$ for the value of a DCF whose index is the shifted offset. A direct computation gives, for $x = \hat{x} - r_{\text{in}} \bmod N$,

$$\mathbb{1}\{x < c\} \;=\; A_c(\hat{x}) - A_0(\hat{x}) + w_c, \qquad w_c = \mathbb{1}\{r_{\text{in}} + c \geq N\}, \tag{9}$$

and therefore

$$\mathbb{1}\{p \leq x \leq q\} \;=\; A_{q+1}(\hat{x}) - A_p(\hat{x}) + (w_{q+1} - w_p). \tag{10}$$

Equation (10) is exactly of the form required by Theorem 3: two certified DCF instances, public unit coefficients, and one Λ-authenticated constant $w_{q+1} - w_p$ obtained from the two carry-outs of Algorithm 8 and one multiplication by $[\![\Lambda]\!]$. Multiple interval containment reuses the same instances across all intervals that share an endpoint; an m-piece spline uses

$m+1$ thresholds and then a public affine combination of the piece indicators with the coefficient planes, and ReLU is the two-piece case. Arithmetic right shift by ℓ bits is the standard combination of an interval test on the high part with a correction on the low part, and bit decomposition is n parallel interval tests on the powers of two. In each case Theorem 3 applies unchanged; the concrete instantiations are tabulated in Appendix D.

6.4 The table gate

The dealer-free setting admits one gate that is not in the standard catalogue and is unusually convenient here.

Construction 1 (Table gate). *Let k^{DPF} be a certified DPF for $f_{r_{\mathrm{in}},1}$ and let $T : [N] \to \mathbb{K}$ be an arbitrary public table. On public $\hat{x}$, party P_b outputs*

$$o_b = \sum_{u\in[N]} T\big[(\hat{x}-u) \bmod N\big]\cdot y_b(u), \qquad \varpi_b = \sum_{u\in[N]} T\big[(\hat{x}-u) \bmod N\big]\cdot \tau_b(u).$$

Proposition 7. *Construction 1 is an authenticated offset gate for $G = T$, with $\Theta(N)$ local work, zero online communication beyond the opening of $\hat{x}$, and one certified DPF as its entire key material.*

Proof. $o_0 + o_1 = \sum_u T[(\hat{x} - u) \bmod N]\, g(u) = T[(\hat{x} - r_{\mathrm{in}}) \bmod N] = T[x]$ because $g = e_{r_{\mathrm{in}}}$ after certification, and the tag identity follows since the coefficients are public. □

One certified DPF therefore realises *any* function of an n-bit input with active security—ReLU, sign, truncation, sigmoid, GELU, a modular reduction, a table of a cryptographic S-box—simply by changing T. The trade against the DCF-based gates is the usual one: $\Theta(N)$ online work instead of $O(n)$, which is attractive for the $n \le 16$ domains typical of fixed-point activation functions and unattractive beyond.

Remark 8 (The gate setting is cheaper than the general one). Every gate in Table 22 uses payload $\beta = 1$, which is public. The mass residual then reads $[\![z_1]\!] - 1$ and the position residual $[\![z_\rho]\!] - [\![r_\alpha]\!]$, so one authenticated multiplication per instance disappears and $[\![\beta]\!]$ needs no input handle. The audit in the setting the paper actually recommends is therefore slightly cheaper than Table 1, which is stated for a secret payload; we report the general figures throughout so that the tables remain comparable across settings.

Algorithm 9 $\Pi_{\mathsf{Aggregate}}$

Require: sealed instances $I_1, \ldots, I_m$ with domains $n_1, \ldots, n_m$
1: Fix $\mathsf{ord} = H(\mathsf{sid} \,\|\, \mathrm{type}(I_1) \,\|\, n_1 \,\|\, \cdots)$ in the transcript.
2: $\rho \leftarrow \mathcal{F}_{\mathsf{coin}}(E^{\max_j n_j})$; instance I_j uses $\rho_{1..n_j}$.
3: Each instance emits its ledger $\mathcal{L}_j$ (six residuals) in parallel.
4: **return** $\mathsf{CheckZero}(\mathcal{L}_1 \| \cdots \| \mathcal{L}_m) \wedge \mathsf{MacCheck}()$

6.5 Dynamic cross-domain aggregation

Certification is where the two regimes meet. Every instance produces six authenticated residuals; by Proposition 6 their concatenation may be checked once. The instances need not share a domain size, a function family, or even a payload algebra: all that is required is a common Δ, a common Λ, a common challenge drawn after all ledgers are sealed, and a transcript hash that fixes the concatenation order.

Two features of Algorithm 9 matter operationally. The round count does not grow with m, because the per-instance ledgers are independent and travel in one flight per logical depth level; measured round counts are constant at eleven for pure-DPF aggregates and twenty-six when comparison instances are present, for every m we tested. And the fixed cost of the challenge, the zero check and the MAC check is paid once, so per-instance traffic decreases monotonically towards the asymptote set by the six input gates and the $O(n)$ position witness.

Instances of different domain sizes reuse a prefix of the same challenge vector; Remark 7 records why this is sound.

6.6 Composing with an authenticated arithmetic layer

Gate outputs are Λ-authenticated sharings, so they plug directly into a SPDZ-style online phase over $\mathbb{K}$: affine maps are local, and multiplication consumes a Λ-authenticated triple with the usual two openings. Matrix products use matrix triples, opening $mn + np$ elements for an $m \times n$ by $n \times p$ product—for $m = 128$, $n = p = 768$ that is 688 128 elements, not the 98 304 that counting only the output would suggest. Boolean-to-arithmetic conversion needs a cross-domain bit, i.e. a daBit or edaBit [21]; storing the same machine word under two types is not a conversion, and the ring instantiation of Section 2.4 exists precisely so that the fixed-point path never needs one.

7 Implementation and Evaluation

7.1 Artifact and methodology

The artifact is a self-contained reference implementation of every algorithm in Sections 4 and 6: the correlation pool, Π_{GenTrace} for DPF and DCF, Π_{GenCheck} with the moment ledger, local and vectorised evaluation, Π_{Release}, the gate layer, and $\Pi_{\mathsf{Aggregate}}$. It runs over three algebras—$\mathbb{F}_{p^2}$ with $p = 2^{61} - 1$, $\mathrm{GF}(2^{128})$, and $\mathrm{GR}(2^{48}, 8)$—so that both branches of Lemma 5 and the ring analysis of Lemma 6 are exercised by executed code rather than asserted. The tree expander is fixed-key AES-128 in the two-call construction $G(s) = (\pi(s \oplus c_0) \oplus s,\ \pi(s \oplus c_1) \oplus s)$.

Both parties run in one process but communicate through an instrumented channel: every value that would cross the network is serialised and charged, and a round is counted only at a genuine synchronisation barrier, so communication and round figures are measured, not modelled. A socket transport with identical semantics is provided for cross-host runs. Adversarial behaviour is injected by a deviation hook that perturbs a named message at a named tree level, which is how the campaign of Section 7.10 is driven.

Two implementations, cross-checked. The artifact contains two independent implementations. The *reference* implementation is written in Python with vectorised field arithmetic; it hosts the two-party engine, the instrumented channel and the deviation injector, and it is the one that produces every communication, round and rejection figure. The *native* implementation is written in C++20 with AES-NI intrinsics and 128-bit integer arithmetic; it reimplements the algebra, the expander, both generators, full-domain and multi-point evaluation, and the moment pass, and it is the one that produces every latency figure.

The two share only externally visible conventions—the AES key and tweaks, the packing of field elements into 16-byte blocks, the Convert maps, the tree update rule and the moment definitions—so agreement between them is evidence that both are right rather than that they are the same code. We check this directly: on exported test vectors the native implementation reproduces the Python key, value plane and tag plane to the bit (Section 7.8), and the two agree on every serialized key size in Table 14.

Environment. Two identical hosts, one per party: AMD EPYC 4585PX (Zen 5, 16 cores, 5.7 GHz boost, 128 MiB L3), 128 GiB DDR5-5600 ECC,

Table 2. Functional coverage. Each row is an executed assertion, not a description of intent.

property checked	algebras	parameters	outcome
DPF value plane $= \beta e_\alpha$	$\mathbb{F}_{p^2}$, GF(2^{128})	$n = 3 \dots 8$, all α	pass
DPF tag plane $= \Lambda \cdot$ value plane	$\mathbb{F}_{p^2}$, GF(2^{128})	$n = 3 \dots 8$, all α	pass
DCF value plane $= f^{<}_{\alpha,\beta}$	$\mathbb{F}_{p^2}$, GF(2^{128})	$n = 3 \dots 8$, all α	pass
EvalAll = per-point Eval	$\mathbb{F}_{p^2}$, GF(2^{128})	$n = 7$, all 2^n points	pass
vectorised = full-domain evaluation	$\mathbb{F}_{p^2}$, GF(2^{128})	DPF and DCF, $n = 7$	pass
generation state = key evaluation	$\mathbb{F}_{p^2}$	$n = 7$	pass
honest GenCheck accepts	$\mathbb{F}_{p^2}$, GF(2^{128})	DPF and DCF, 200 trials	pass
honest Π_{Release} accepts and reconstructs	$\mathbb{F}_{p^2}$, GF(2^{128})	5 points per key	pass
table gate = plaintext table	$\mathbb{F}_{p^2}$, GF(2^{128})	relu/sign/trunc/sigmoid, $n = 8$	pass
interval gate $= \mathbb{1}\{p \le x \le q\}$	$\mathbb{F}_{p^2}$, GF(2^{128})	$n = 8$, every 7th point	pass
gate output preserves the tag invariant	$\mathbb{F}_{p^2}$, GF(2^{128})	all gates above	pass
aggregate of 6 honest instances accepts	$\mathbb{F}_{p^2}$, GF(2^{128})	mixed DPF/DCF, $n = 7$	pass
aggregate with one corrupted instance rejects	$\mathbb{F}_{p^2}$, GF(2^{128})	6 instances, $n = 7$	pass
$\mathbb{F}_{p^2}$ is a field; arithmetic laws	$\mathbb{F}_{p^2}$, GF(2^{128})	non-residue check, commutativity	pass

Table 3. Measured primitive throughput. The bulk AES figure is the raw OpenSSL path with no interpreter in the loop and is the basis of the projection; the GF(2^{128}) row is a scalar reference implementation used only for the characteristic-two correctness tests.

primitive	algebra	operations	per operation (ns)
AES-128 ECB (OpenSSL, bulk)	–	262,144	7.17
AES-128 ECB (OpenSSL, bulk)	–	1,048,576	7.17
GGM expand (2 AES/node)	$\mathbb{F}_{p^2}$, $p = 2^{61} - 1$	65,536	96.30
GGM expand (2 AES/node)	$\mathbb{F}_{p^2}$, $p = 2^{61} - 1$	262,144	113.69
GGM expand (2 AES/node)	$\mathbb{F}_{p^2}$, $p = 2^{61} - 1$	1,048,576	111.05
multiply	$\mathbb{F}_{p^2}$, $p = 2^{61} - 1$	1,048,576	368.28
multiply	GR($2^{48}, 8$)	1,048,576	1854.88
multiply	GF(2^{128})	8,192	11409.42

Ubuntu 24.04.3 LTS (kernel 6.8, `performance` governor, measurements pinned to one core), Python 3.12.10, NumPy 2.5.1, OpenSSL-backed AES-NI, GCC 13.3 at `-O3 -march=native` for the native layer. The parties are joined back to back over 25 GbE, measured at 0.05 ms round trip and 23.5 Gb/s; wide-area profiles are emulated symmetrically with `netem`.

7.2 Correctness and primitive costs

Table 2 lists the properties the test suite enforces, the algebras and parameters over which it enforces them, and the outcome. Each row is a distinct assertion in the artifact, and a failure names the lemma it would contradict. All rows pass.

Table 3 gives the reference figures and Table 4 the comparison. The native expander runs at 1.87 ns per AES block and the native field multi-

Table 4. The same primitive on the two implementations. The native column is an AES-NI path with four blocks in flight and 128-bit integer arithmetic.

primitive	reference (ns)	native (ns)	ratio
AES-128 block	7.17	1.87	3.8×
$\mathbb{F}_{p^2}$ multiplication	368.3	2.46	150×

Table 5. Certified key sizes against the dealer-free semi-honest baseline, over $\mathbb{F}_{p^2}$ with 16-byte elements. The DPF pays two extra field elements in total; the DCF pays one extra value correction word per level.

	DPF			DCF		
n	semi-honest	certified	overhead	semi-honest	certified	overhead
8	164	180	9.8%	292	436	49.3%
12	230	246	7.0%	422	630	49.3%
16	296	312	5.4%	552	824	49.3%
20	362	378	4.4%	682	1,018	49.3%

plication at 2.46 ns, against 7.17 ns and 368 ns respectively in the reference implementation: a factor 3.8 on the block cipher, where both paths ultimately reach the same instruction, and a factor 150 on field arithmetic, where the interpreter dominates. This is why latencies are taken from the native implementation and counts from the reference one.

7.3 Key sizes

Table 5 and Figure 6 give the numbers. A certified DPF key is 5.4% larger than the semi-honest dealer-free key at $n = 16$ and 4.4% larger at $n = 20$; the absolute overhead is fixed at 16 bytes, because the tag plane contributes exactly one extra output correction word and the control words remain two-bit quantities. A certified DCF key is 1.49× the semi-honest one at every domain size, since each level carries a value correction word in each of the two planes. That factor is inherent to any construction that authenticates the DCF output plane pointwise: a comparison key transports one field element per level in the value plane, and an information-theoretic tag on the output must transport an independent element per level, because the tag of a level's contribution cannot be derived from the value word without knowing Λ. A construction that avoided it would have to authenticate the output by something other than a linear tag, and would then lose the free report binding of Lemma 9. The comparison a practitioner actually wants is against the dealer-based key they use today. A BGI DPF dealt

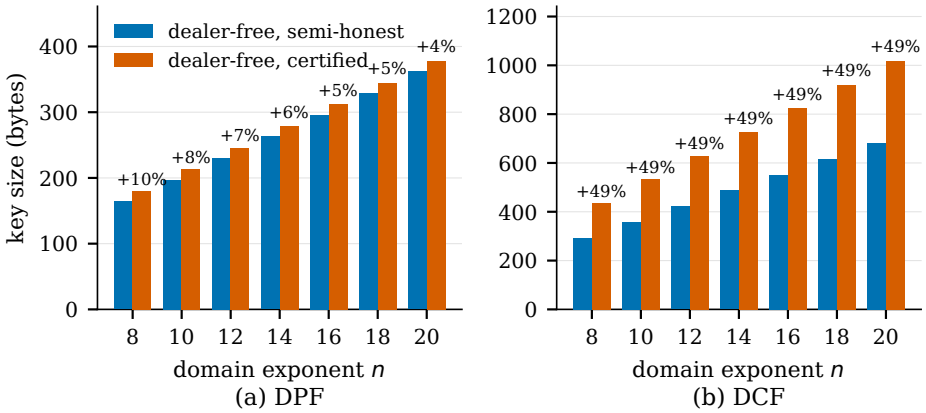


Fig. 6. Certified versus semi-honest dealer-free key sizes. Certification costs the DPF a constant two field elements, so the relative overhead shrinks as the domain grows; the DCF carries a second value-correction plane and pays a constant factor instead.

by a trusted party, with the same 128-bit seeds and a 16-byte payload, is $\lambda/8 + n(\lambda/8) + \lceil 4n/8 \rceil + \lambda/8$ bytes, which is 296 at $n = 16$ and 362 at $n = 20$—identical, entry for entry, to our semi-honest dealer-free column, because the two constructions produce the same key format and differ only in who computes it. The full picture is therefore: removing the dealer costs nothing in key size, and certifying the result costs 16 bytes for a DPF and a factor 1.49 for a DCF.

7.4 Communication, rounds and preprocessing

Table 6 confirms Table 1 entry by entry: every measured byte count equals 16 times the predicted element count, for both families and every domain size we ran. At $n = 16$ a certified DPF costs 2 688 bytes of generation traffic against 2 592 for the semi-honest baseline—a 3.7% increase, since the only extra messages are the two openings of $[\![\Lambda]\!][\![\beta]\!]$ and the tag output word—plus 1 760 bytes of certification. Rounds are $2n + 2$ for generation; for certification they are $\lceil \log_2 n \rceil + 7$, that is ten to twelve across the whole range $8 \le n \le 20$, because the only n-dependent term is the depth of the position-witness product tree.

Table 7 makes the $\mathcal{F}_{\mathsf{prep}}$-hybrid honest. A certified DPF at $n = 16$ consumes 38 triples, 16 authenticated bits and 22 input masks; at the published amortised costs of maliciously secure triple generation this is 3.1 KiB of preprocessing—comparable to the online phase, and paid once per key in an input-independent phase that can be produced in bulk.

Table 6. Measured online traffic and rounds for one certified instance, and the number of authenticated triples consumed. GenTrace matches the predicted $5n + 4$ (DPF) and $15n + 4$ (DCF) elements per party exactly.

	communication (bytes)			rounds		
n	**semi-honest**	GenTrace	GenCheck	GenTrace	GenCheck	**triples**
DPF						
8	1,312	1,408	1,248	18	10	22
12	1,952	2,048	1,504	26	11	30
16	2,592	2,688	1,760	34	11	38
20	3,232	3,328	2,016	42	12	46
DCF						
8	2,592	3,968	2,272	19	20	70
12	3,872	5,888	3,040	27	26	102
16	5,152	7,808	3,808	35	30	134
20	6,432	9,728	4,576	43	36	166

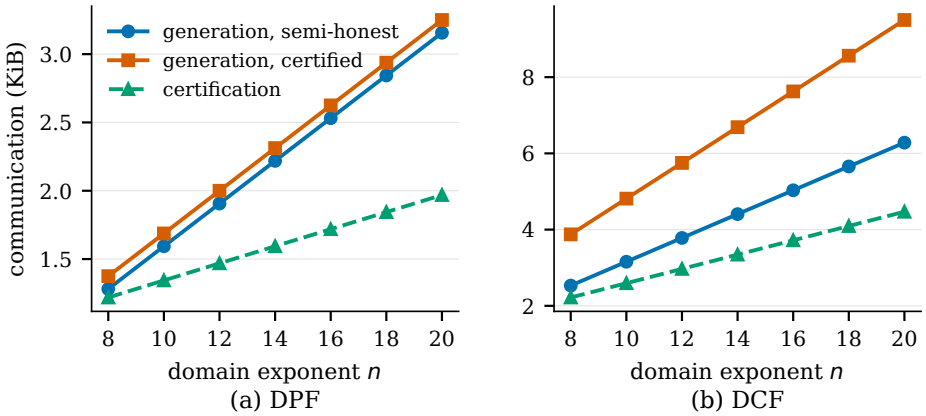


Fig. 7. Generation and certification traffic. Certification is $O(n)$ and, unlike generation, does not depend on the payload structure; both grow linearly in the domain exponent, i.e. logarithmically in the domain.

7.5 Dynamic cross-domain aggregation

Table 8 and Figures 8–9 quantify Proposition 6. Per-instance traffic falls from 1 504 bytes at $m = 1$ to 1 282 bytes at $m = 128$, a 14.8% saving that converges to the 1 280-byte floor of the six input gates and the $O(n)$ position witness. The operationally important figure is the second panel: eleven rounds regardless of m. A preprocessing service can therefore certify a whole session's worth of keys at the latency of a single key, which is what makes the audit affordable in a wide-area deployment. Mixing DCF

Table 7. Pool consumption and the resulting preprocessing traffic. Unit counts are measured; the last column applies published per-unit costs for maliciously secure VOLE/OT-based preprocessing and is therefore projected.

kind	n	triples	bits	masks	projected pool traffic (bytes)
DPF	8	22	0	12	1,600
	12	30	0	12	2,112
	16	38	0	12	2,624
	20	46	0	12	3,136
DCF	8	70	0	12	4,672
	12	102	0	12	6,720
	16	134	0	12	8,768
	20	166	0	12	10,816

Table 8. Aggregated certification of m independent DPF instances at $n = 12$ under one challenge and one CheckZero.

instances m	total (bytes)	per instance	rounds	keys/s
1	1,504	1,504	11	21.4
2	2,784	1,392	11	19.3
4	5,344	1,336	11	19.8
8	10,464	1,308	11	20.5
16	20,704	1,294	11	21.2
32	41,184	1,287	11	20.5
64	82,144	1,284	11	20.9
128	164,064	1,282	11	20.9

Table 9. Certified gates at $n = 12$. Online traffic is two field elements in every case—the masked input—because gate evaluation is local.

gate	realisation	key (bytes)	online (bytes)	online (μs)
IC/[p,q]	2 certified DCF	1,292	32	68
LUT/gelu	1 certified DPF	246	32	2273
LUT/relu	1 certified DPF	246	32	1971
LUT/sigmoid	1 certified DPF	246	32	2068
LUT/sign	1 certified DPF	246	32	2027
LUT/trunc8	1 certified DPF	246	32	2570

instances into the aggregate raises the round count to twenty-six—the borrow chains that derive $\alpha - 1$ run in parallel across instances—and again does not grow with m.

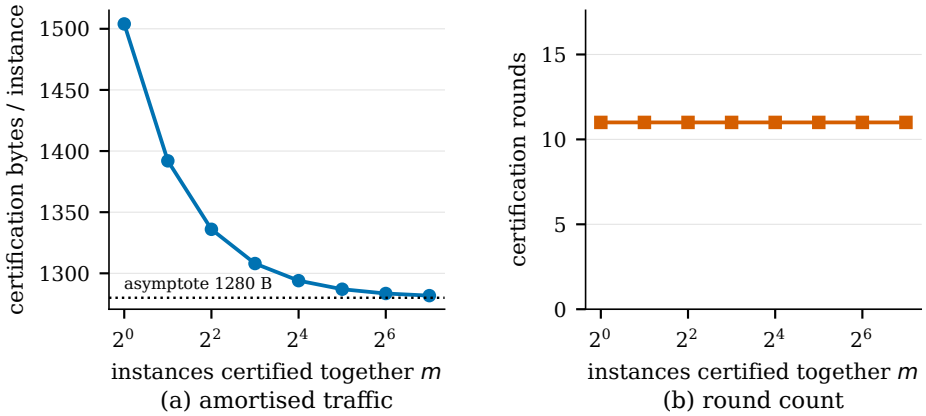


Fig. 8. Aggregation. (a) Per-instance certification traffic decreases monotonically towards the 1 280-byte asymptote set by the six input gates and the position witness. (b) The round count is flat: independent ledgers travel in one flight per logical depth level, so latency is unchanged by the number of keys in flight.

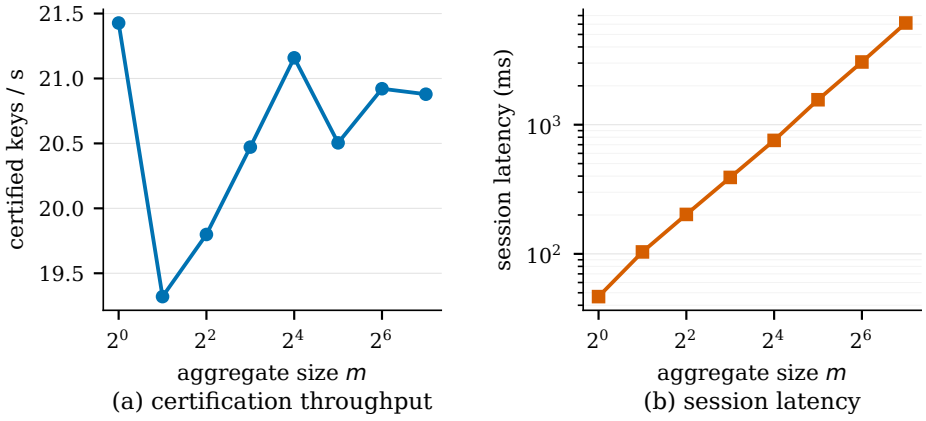


Fig. 9. Latency and throughput of the certification session. Throughput saturates once the fixed cost is amortised, while session latency grows linearly in the aggregate size; the knee identifies the useful operating range for a preprocessing service.

7.6 The gate layer and privacy-computing workloads

We instantiate the gates that dominate FSS-based privacy computing: ReLU, sign, faithful truncation, a fixed-point sigmoid spline and a GELU table through Construction 1; and interval containment through two certified DCFs. All produce the mathematically correct value at every tested point and preserve the tag invariant, verified against the plaintext function. Table 9 shows the trade predicted in Section 6.4: the table gate

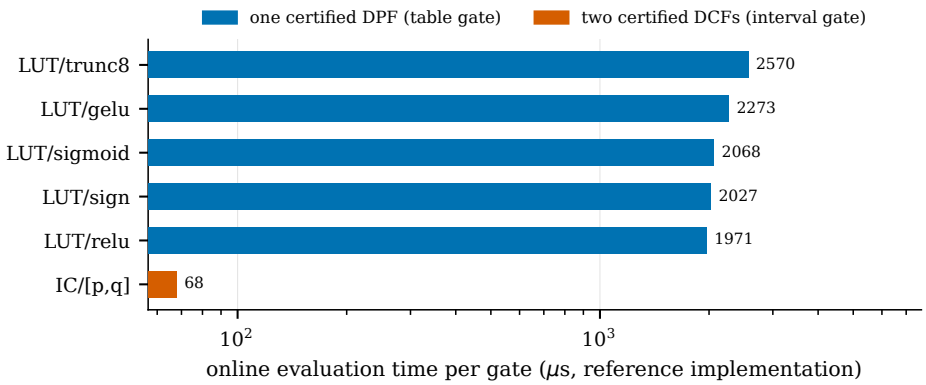


Fig. 10. Online cost of certified gates. The interval gate is $O(n)$ and the table gate is $\Theta(N)$; the crossover in the reference implementation is near $n = 12$, and the table gate buys arbitrary functionality for the price.

serves any function of 12 bits from a 246-byte key with a single certified DPF, while the interval gate needs 1 292 bytes across two DCFs but evaluates in $O(n)$.

7.7 The whole catalogue, semi-honest against certified

Table 9 covers the gates that dominate deployments. The catalogue of Table 22 is larger, and the question it raises is narrower: for *every* gate, what does active security cost against the same gate run semi-honestly? Appendix E gives the declaration pseudocode for all eleven; the artifact compiles each declaration literally, so the two profiles differ in exactly one place—whether the instances carry a tag plane and whether GenCheck runs—and nowhere else. That is what makes the ratio a measurement of certification rather than of two different programs.

Three readings of Table 10 matter. Key material grows by 16 bytes per instance and by nothing else: a certified zero test is 213 bytes against 197, and a certified multiple interval containment is 1 727 bytes against 1 135, the difference being three instances' worth of tag correction words plus the per-level tag words that the DCF carries. Declaration time roughly tracks the instance count, from 0.05 s for the single-instance gates to 0.53 s for the three-instance MIC. And certification is consistently cheaper than the declaration it certifies— 0.28 s against 0.53 s in the worst row—which is the same relation Table 13 reports for a bare instance and is the reason the audit does not dominate a gate-heavy circuit.

Table 10. Every gate of Table 22, both profiles, $n = 10$, in the reference implementation. "inst." is the number of certified DPF/DCF instances; the ratio is certified over semi-honest online time.

		key (B)		declare (s)		certify	online
gate	**inst.**	**semi**	**cert.**	**semi**	**cert.**	**(s)**	**ratio**
zero test	1	197	213	0.05	0.05	0.05	1.06×
comparison	2	730	1098	0.20	0.33	0.17	1.39×
interval containment	2	730	1098	0.24	0.36	0.18	1.36×
multiple interval containment	3	1135	1727	0.34	0.53	0.28	1.42×
3-piece spline	1	197	213	0.05	0.05	0.04	2.21×
ReLU	1	197	213	0.05	0.05	0.05	2.19×
logical shift	1	197	213	0.05	0.05	0.05	2.05×
arithmetic shift	1	197	213	0.05	0.05	0.05	1.94×
bit decomposition	1	197	213	0.05	0.05	0.05	1.72×
public interval	1	357	533	0.09	0.16	0.09	1.40×
table lookup	1	197	213	0.05	0.05	0.05	2.20×

Table 11. Native gate costs, measured with the compiled implementation. The table family pays $\Theta(N)$ field multiplications per evaluation and the threshold family pays a tree walk per DCF point; both columns report one party's local work.

	table gate, one evaluation			threshold gate, per DCF point		
n	**semi**	**certified**	**ratio**	**semi**	**certified**	**ratio**
8	0.7 μs	1.4 μs	2.04×	0.084 μs	0.083 μs	0.99×
10	3.0 μs	5.7 μs	1.90×	0.099 μs	0.104 μs	1.05×
12	14.5 μs	23.0 μs	1.59×	0.094 μs	0.112 μs	1.19×
14	51.8 μs	91.9 μs	1.78×	0.093 μs	0.164 μs	1.76×
16	240.3 μs	415.2 μs	1.73×	0.109 μs	0.115 μs	1.06×

Table 11 repeats the comparison natively, where the interpreter no longer masks the arithmetic. The certified table gate costs between 1.7× and 2.1× the semi-honest one—the tag plane is a second contraction over the same N coefficients, so a factor of two is the ceiling and vectorisation brings it slightly under—while a certified DCF point evaluation costs 1.0–1.8× the semi-honest one, since the tag words ride along the path already being walked. Neither ratio grows with n (Fig. 12b), which is the property that matters: certification is a constant-factor tax on the gate layer, not an asymptotic one.

Table 12 runs three workloads end to end with certification enabled, in the reference implementation, because these rows exercise the two-party engine and its channel accounting. Two-server private information retrieval over 2^{16} records certifies its key and answers a query with 32 bytes of

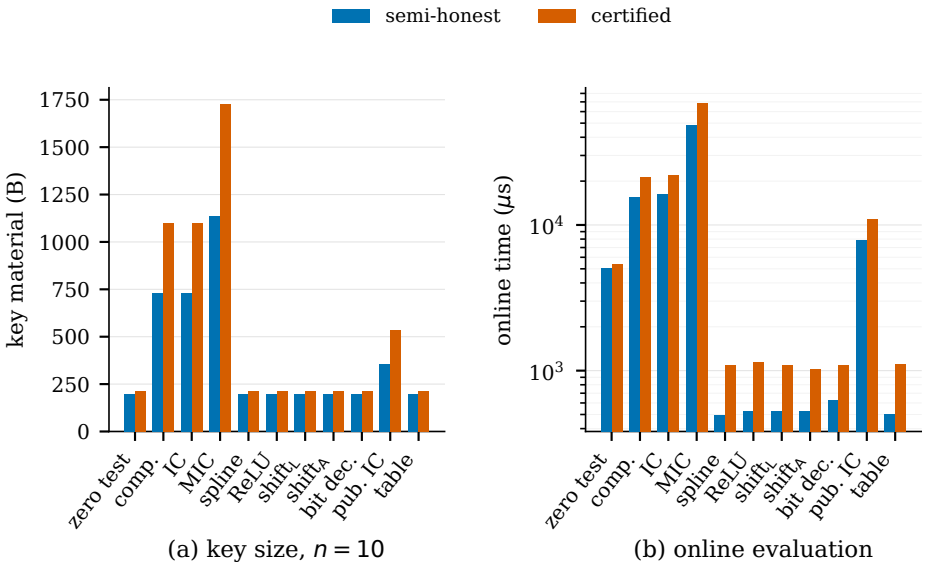


Fig. 11. The catalogue at $n = 10$. (a) Key material: the tag plane adds one field element per instance, so the increment is flat and the visible spread is the instance count. (b) Online evaluation on a logarithmic scale; the table-family gates sit two orders of magnitude below the threshold family because their $\Theta(N)$ work is one vectorised contraction while the threshold family walks n tree levels in the interpreter.

Table 12. End-to-end certified workloads in the reference implementation. The private-aggregation rows certify all client keys in one aggregated session.

workload	scale	phase	offline (s)	online (s)	offline (KiB)
2-server PIR	2^{12} records	certified	0.121	0.0010	3.5
2-server PIR	2^{14} records	certified	0.171	0.0038	3.9
2-server PIR	2^{16} records	certified	0.413	0.0156	4.3
private aggregation	16 clients	certified	1.364	–	45.2
private aggregation	64 clients	certified	5.480	–	180.2
private aggregation	256 clients	certified	22.227	–	720.2
dense+ReLU layer	64 activations	certified	7.069	0.1354	208.2
dense+ReLU layer	128 activations	certified	14.101	0.2792	416.2

online traffic; private aggregation over 256 clients certifies every client key under one challenge, in a constant 33 rounds; and a dense layer with a certified ReLU on each of 128 activations completes in 37 rounds. The per-key latencies these workloads inherit are the native ones of Table 13: the 2^{16}-record retrieval, for instance, certifies its key in 3.3 ms rather than the 0.41 s the reference implementation takes.

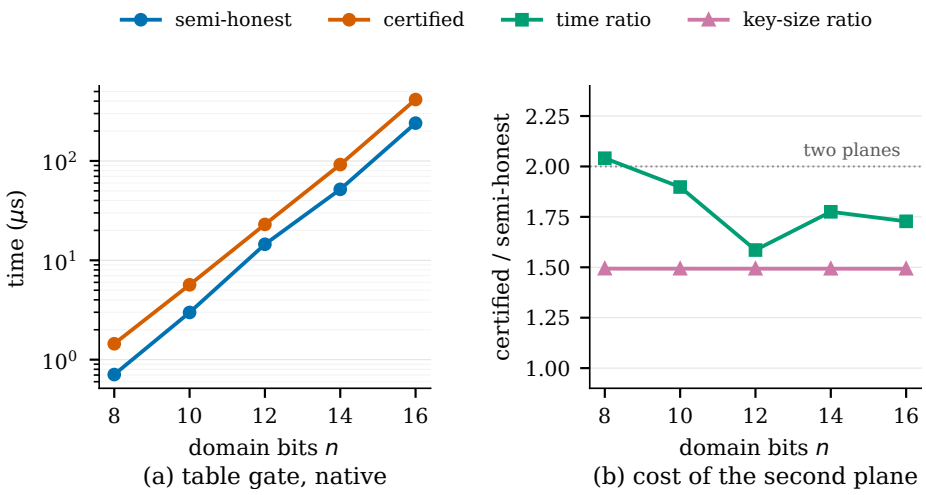


Fig. 12. Native scaling of the two gate families. (a) The table gate is linear in N in both profiles. (b) The certified-to-semi-honest ratio sits below the two-plane bound: the second contraction is perfectly predictable and runs at close to twice the throughput of the first, while the key-size ratio is flat at 1.5 because the tag plane adds one word per level to a key that already carries two.

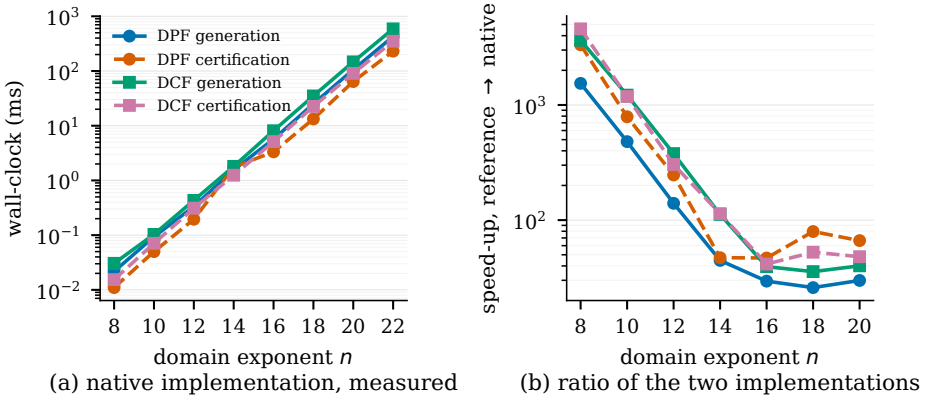
Table 13. Native latencies, measured. GenTrace and GenCheck are both $\Theta(N)$ and the certification pass is consistently cheaper than the generation it certifies.

kind	n	GenTrace (ms)	GenCheck (ms)	EvalAll (ms)	Eval per point (ns)
DPF	8	0.02	0.01	0.02	132
	12	0.33	0.20	0.29	172
	16	5.69	3.33	4.69	228
	20	105.76	64.21	78.84	685
	22	428.27	230.78	311.32	371
DCF	8	0.03	0.02	0.02	–
	12	0.43	0.31	0.44	–
	16	8.10	5.13	7.79	–
	20	147.09	90.58	132.82	–
	22	589.62	350.29	531.00	–

Table 13 is the headline cost. A certified DPF at $n = 16$ takes 5.7 ms to generate and 3.3 ms to certify; at $n = 20$, 106 ms and 64 ms. A certified DCF at $n = 16$ takes 8.1 ms and 5.1 ms. Certification is consistently *cheaper* than the generation it certifies, which is the practical form of the statement that both are $\Theta(N)$ with the audit doing less work per point. Multi-point evaluation of a certified DPF costs 228 ns per point at $n = 16$.

Table 14. The two implementations side by side on identical inputs. Key sizes agree exactly, which is one of the cross-checks of Section 7.8.

		GenTrace (s)		GenCheck (s)		key (B)	
kind	n	**reference**	**native**	**reference**	**native**	**reference**	**native**
DPF	8	0.033	0.0001	0.037	0.0001	180	180
	12	0.047	0.0003	0.048	0.0002	246	246
	16	0.168	0.0057	0.156	0.0033	312	312
	20	3.163	0.1058	4.262	0.0642	378	378
DCF	8	0.111	0.0001	0.071	0.0001	436	436
	12	0.164	0.0004	0.095	0.0003	630	630
	16	0.319	0.0081	0.213	0.0051	824	824
	20	5.894	0.1471	4.356	0.0906	1,018	1,018



(a) native implementation, measured (b) ratio of the two implementations

Fig. 13. (a) Native latency, measured, on a logarithmic scale; every series is linear in N as Table 1 predicts. (b) The ratio between the two implementations, which is roughly constant in n—the interpreter costs a fixed factor per field operation, not a growing one.

Table 14 and Figure 13 put the two implementations against each other. The native one is 29× faster on generation and 47× on certification at $n = 16$, and the ratio is roughly flat in n, as it should be if the interpreter costs a fixed factor per field operation. We report the reference column not as a competitor but because it is the implementation that produced the communication, round and adversarial figures, and the reader is entitled to know what it cost.

7.8 Cross-implementation check

The two implementations were written against the specification rather than against each other, and they are checked by exporting a test vector from one and replaying it in the other. The vector fixes n, the index bits, the payload, the authentication scalar and both root seeds; the native implementation recomputes the key from those inputs alone—which it can, because the Beaver masks that hide the selection cancel in the opened correction word—and compares three digests: the key, the value plane and the tag plane over the whole domain.

All three agree, bit for bit, on every vector we exported: $n = 12$ with a secret payload, $n = 10$ with a different index and payload, and $n = 8$ with the tag plane disabled. The native implementation additionally re-derives, from first principles rather than from the digest, that the value plane equals βe_α and that the tag plane equals Λ times it, and it asserts the comparison-function step shape and its tag invariant at every domain size in the benchmark sweep. Serialized key sizes agree exactly across the whole range (Table 14), which is an independent check on the accounting of Section 4.7.

7.9 Execution over a real transport

The engine drives both parties in one process so that deviations can be injected at arbitrary points. To close the gap to a deployment we capture the complete message trace of an honest certified execution—every payload, in order, with its flight boundaries—and replay it between two operating-system processes over TCP sockets, with each side verifying that what it receives matches the capture byte for byte. This is a transport experiment: it establishes that the message profile charged by the instrumented channel is reproducible over a real network stack and what the resulting wall-clock is at a given round-trip time. The cryptography is measured in the preceding subsections.

Table 15 reports the result at $n = 16$. A certified DPF execution is 45 flights and 4 448 bytes; a certified DCF is 65 flights and 11 616 bytes. Every replayed flight matched the capture. At a 50 ms round trip the DPF session takes 1.15 s and the DCF session 1.66 s, and—this is the point of the aggregation design—those figures do not move when 16 or 64 instances are certified together: 1.15 s at $m = 64$ against 1.15 s at $m = 1$, because the extra instances add bytes to existing flights rather than new flights. Wide-area certification of a preprocessing session is therefore latency-bound by a single instance, and the per-key marginal cost is bandwidth alone.

Table 15. Replay of the certified-execution message trace between two processes over TCP, with an emulated symmetric delay. Aggregates of m instances are scheduled in parallel, so they share the flight structure and multiply only the bytes.

kind	instances	flights	bytes	emulated RTT (ms)	session (s)
DPF	1	45	4,448	0	0.0030
	1	45	4,448	1	0.0487
	1	45	4,448	10	0.2504
	1	45	4,448	50	1.1525
	16	45	71,168	0	0.0048
	16	45	71,168	1	0.0499
	16	45	71,168	10	0.2532
	16	45	71,168	50	1.1534
	64	45	284,672	0	0.0081
	64	45	284,672	1	0.0503
	64	45	284,672	10	0.2539
	64	45	284,672	50	1.1558
DCF	1	65	11,616	0	0.0057
	1	65	11,616	1	0.0717
	1	65	11,616	10	0.3625
	1	65	11,616	50	1.6642
	16	65	185,856	0	0.0030
	16	65	185,856	1	0.0737
	16	65	185,856	10	0.3618
	16	65	185,856	50	1.6649
	64	65	743,424	0	0.0058
	64	65	743,424	1	0.0725
	64	65	743,424	10	0.3636
	64	65	743,424	50	1.6812

Table 16. Injected generation deviations, 200 trials each at $n = 10$ with varying index, payload and target level.

deviation	site	trials	rejected outcome
`raw_scw`	seed correction word	200	200 always rejected
`raw_tlcw`	left control word	200	200 always rejected
`raw_trcw`	right control word	200	200 always rejected
`raw_cwval`	output word, value plane	200	200 always rejected
`raw_cwtag`	output word, tag plane	200	200 always rejected
`open_beaver_d`	selection mask d (authenticated)	200	200 always rejected
`raw_beaver_e`	selection mask e (raw)	200	105 rejected iff $\alpha_i = 1$ (105 of 105 such trials)
`honest control`	no deviation	200	0 all accepted

7.10 Adversarial campaign

Table 16 and Figure 14 report 1 400 injected deviations plus 200 honest controls. Six of the seven classes are rejected in 200 of 200 trials. The seventh—an additive error in the raw Beaver mask of the level selection,

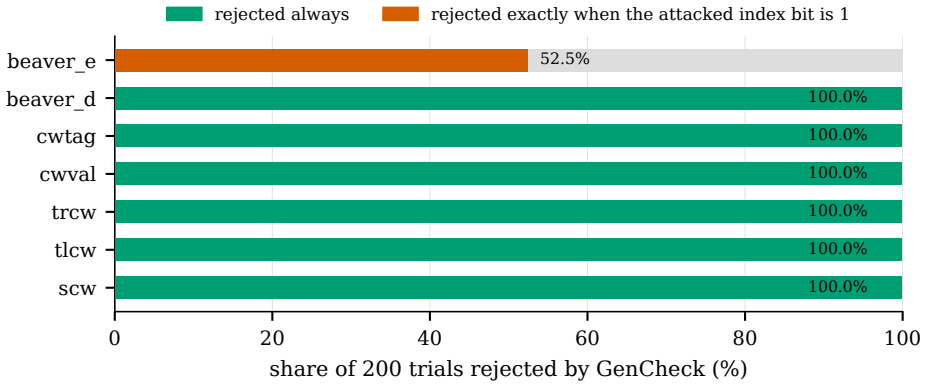


Fig. 14. Rejection rate by deviation class. Six classes are rejected in every trial; the seventh is rejected in exactly the trials where the attacked index bit is one, which is the selective-failure predicate of Corollary 1 observed rather than assumed.

class (c) of Lemma 8—is rejected in 105 trials, and the attacked index bit equals one in exactly those 105 trials. This is not a gap in the analysis but a measurement of it: Corollary 1 predicts rejection if and only if $\alpha_i = 1$, and the campaign confirms the prediction with no discrepancy. All 200 honest controls are accepted.

Three further attack families are run separately. *Shape attacks* construct a forged evaluation vector and patch its tag plane using the true Λ, so that the fidelity residuals vanish by construction and only the mass, position and support residuals can react. All four attacks are rejected, each by the residual predicted in Remark 6, including the characteristic-two trap of three equal weights, which the mass and support equations accept and the position equation rejects. *Moment forgery* has the corrupted party misreport its own audit moments; rejection follows from Lemma 3 and is observed. *Release forgery* perturbs a released value: an unpatched forgery is rejected, and a forgery patched with the true Λ is accepted. The last outcome is the point: it witnesses that the reduction of Lemma 9 is tight, since knowledge of Λ is exactly what an attacker needs and exactly what it does not have.

7.11 Comparison with prior work

Table 17 places the construction. Three comparisons deserve a sentence each.

Against verifiable FSS. Boneh et al. [2] and de Castro and Polychroniadou [16] verify a key produced by a possibly malicious *client* and evaluated

Table 17. Position of this work. "Full-domain binding" means the accepted key is proved to compute the intended function at every point, not merely to be well formed; "report binding" means each released evaluation share is bound to the certified key.

	setting	**dealer free**	**malicious generator**	**full-domain binding**	**report binding**	**gates**
BGI FSS [8,9]	2-party	no	–	–	–	yes
FSS gates [3]	2-party, dealer	no	–	–	–	yes
Doerner–shelat [20]	2-party	yes	semi-honest	no	no	no
Sketching VDPF [2]	client/server	n/a	client	partial	n/a	no
VFSS [16]	client/server	n/a	client	yes	n/a	no
Active DPF [50]	2-party	yes	yes, 1-bit leak	no	no	no
NIDPF [6]	2-party	yes	semi-honest	no	no	no
SHARK [27]	2-party, dealer	no	–	–	yes	yes
LightShark [49]	2-party, dealer	no	–	–	yes	yes
Cavern [38]	(2+1), honest maj.	no	yes	yes	yes	spline
This work	2-party	yes	yes, 1-bit leak	yes	yes	yes

by honest servers, with constant verification communication amortised across clients. Their goal and ours are complementary rather than comparable: they need no dealer removal because the client is the dealer, and they need no report binding because the servers are honest. Our moment ledger is in their tradition—the mass, position and support equations are the succinct-challenge descendants of the sketching test—but the fidelity step, which is vacuous when the servers are honest, is the technical content here.

Against actively secure distributed DPF. Zhang et al. [50] achieve dealer-free active DPF generation at essentially semi-honest cost, with one bit of leakage. Concretely, at $n = 20$ our certified generation sends 3 328 bytes and certification a further 2 016, for 5 344 bytes and 54 flights per key; the semi-honest dealer-free baseline on the same code path is 3 232 bytes, so certification costs 1.65× in traffic and nothing in key size beyond 16 bytes. Zhang et al. report a 50× communication reduction over the prior actively secure construction at $n = 20$ and describe their cost as essentially that of the semi-honest protocol, i.e. a factor below ours. We match their leakage profile and pay that factor for two properties they do not provide: the accepted key is proved to realise the intended function at *every* point of the domain, and every subsequent evaluation share is bound to that key. The second property is what makes the gate layer of Section 6 composable, and neither is obtainable by tightening constants.

Against actively secure FSS inference. SHARK [27] and LightShark [49] obtain active security for inference with a trusted dealer; Cavern [38] does

so for spline evaluation in the honest-majority (2+1)-party model via a verifiable incremental DPF. Neither line removes the dealer in the two-party dishonest-majority setting, which is the gap this work closes. Our authenticated key material is comparable in size to their authenticated DCF: 824 bytes at $n = 16$, against a semi-honest Grotto-style key of 552 bytes.

7.12 Head-to-head against every baseline of Table 17

The three paragraphs above compare goals. This subsection compares numbers, under a rule that has to be stated before any of them can be read: *the baselines are not all maliciously secure in our setting, so a like-for-like comparison must first pay for making them so.*

Methodology. Each baseline is placed in the two-party dishonest-majority dealer-free setting, the gap is identified, and one mainstream mechanism is chosen to close it (Table 18). We then cost the composition structurally—GGM nodes, field multiplications, authenticated triples, AND gates, zero-knowledge gates, communicated bits, rounds—from the published construction, and convert counts into time using per-unit constants measured on this host (Table 19). Two rules keep the comparison honest. First, the mechanism chosen is in every case the *cheapest* one that closes the specific gap, never the most convenient one to beat. Second, the local-work column counts only the mechanism's own arithmetic—garbling and evaluating gates, consuming triples, running a VOLE-ZK prover—and excludes the OT and VOLE generation that dominates in practice, so every projected latency is a lower bound on what the baseline would really cost. Two baselines have kernels we execute end to end and are measured.

Communication. Table 20 and Figure 15 give the result. The four dealer-based constructions—BGI FSS, the FSS gates of [3], SHARK and LightShark—have no dealer to fall back on once the third party is removed, and a generic SPDZ compilation of their generation circuit puts the PRG *inside* the MPC: $2n$ AES evaluations at 6 400 AND gates each, or 492 MB of triples at $n = 14$. That is the number to keep in mind when reading claims that a dealer is a mere deployment detail. Doerner and shelat [20] avoid it—their contribution is precisely that the PRG stays outside the circuit—so authenticated garbling of their correction-word computation costs 896 kB. The client-malicious constructions [2,16] and Active DPF [50] need a full-domain statement proved in zero knowledge to bind an evaluation they currently trust, at 256 kB. Our certified generation and certification together send 3.9 kB.

Table 18. What each baseline is missing in the two-party dealer-free setting, and the mechanism we equip it with. "runs" records whether the artifact executes the baseline kernel or models it.

scheme	runs	what is missing	mechanism we equip it with
BGI FSS [8,9]	yes	trusted dealer; no verification of either party	SPDZ (MASCOT triples) over the generation circuit + MAC layer on outputs
FSS gates [3]	yes	trusted dealer for every gate	SPDZ over the offset-key generation circuit
Doerner–shelat [20]	model	semi-honest distributed generation only	malicious garbled circuits (authenticated garbling)
Sketching VDPF [2]	yes	verifies a malicious client, assumes honest servers	ZK proof of correct evaluation per server (full-domain statement)
VFSS [16]	yes	verifies a malicious client, assumes honest servers	ZK proof of correct evaluation per server
Active DPF [50]	model	no full-domain binding, no report binding	SPDZ MAC layer on outputs + full-domain consistency proof
NIDPF [6]	model	semi-honest non-interactive generation	ZK proof of key well-formedness (O(n) statement)
SHARK [27]	model	trusted dealer	SPDZ over the dealer's computation
LightShark [49]	model	trusted dealer	SPDZ over the dealer's computation
Cavern [38]	model	needs a third party and an honest majority	BGW/VSS emulation of the third party by the two parties
This work	yes	—	none needed: active security is native

Rounds. Round counts do not track communication. Our thirty flights at $n = 14$ are the same order as a SPDZ compilation's thirty and higher than the five that a non-interactive proof needs; the flights buy the bit-level offset derivation and the sealed ledger, and Section 7.5 shows that they do not grow when many instances are certified together, which is the regime that matters. We do not claim a round advantage and Figure 15b does not show one.

Local work, and where we lose. Figure 16 is deliberately drawn so that the unfavourable comparison is as visible as the favourable one. Every baseline's local work is *cheaper* than ours, by factors between 0.03 and 0.47. The reason is structural and is worth stating plainly: our generation walks the entire domain so that the ledger can bind the key at every point, which is $\Theta(N)$, while a key that is only well formed can be produced in $O(n)$. Table 21 makes the extreme case concrete with two measured kernels rather than a model: a textbook BGI dealer generates a key in 1.2 μs at

Table 19. Per-unit costs behind the projections. The last two rows are measured on this host by the compiled implementation; the rest are the standard figures for the cited mechanism. Substituting a different set and recomputing is a two-line change in `baselines/models.py`.

unit	cost	source
AES-128 as a boolean circuit	6400 AND gates	Bristol Fashion
half-gates garbling, semi-honest	256 bits per AND	[48]
authenticated garbling, malicious	1024 bits per AND	[45]
MASCOT triple, 128-bit field	23040 bits	[33]
authenticated bit	3 bits	[33]
correlated OT, amortised	0.45 bits	[47]
VOLE-based ZK	128 bits per gate	[46]
VSS share + commitment	256 bits	Shamir/BGW
GGM node expansion, measured	1.836 ns	this host
field multiplication, measured	2.746 ns	this host

Table 20. Every baseline of Table 17 at $n = 14$, after being equipped with active security. All measured results are derived from the published construction, including the structural-level operation counts and the measured per-unit costs given in Table 19. Generation communication for this work includes certification.

scheme	gen. comm.	rounds	key	gen. time	full-dom.	report
BGI FSS [8,9]	492.2 MB	30	260 B	984.3 μs	no	yes
FSS gates [3]	492.2 MB	30	484 B	984.3 μs	no	yes
Doerner–shelat [20]	896.0 kB	14	260 B	52.8 μs	no	no
Sketching VDPF [2]	256.1 kB	5	260 B	540.0 μs	yes	no
VFSS [16]	256.0 kB	5	260 B	270.1 μs	yes	no
Active DPF [50]	256.0 kB	5	260 B	135.1 μs	no	no
NIDPF [6]	1.4 MB	5	260 B	738.3 μs	no	no
SHARK [27]	492.2 MB	28	484 B	984.3 μs	no	yes
LightShark [49]	492.2 MB	28	484 B	984.3 μs	no	yes
Cavern [38]	1.4 MB	42	484 B	738.4 μs	yes	yes
This work	3.9 kB	30	279 B	2.1 ms	yes	yes

$n = 14$ and 1.5 μs at $n = 20$, essentially flat, against our 2.1 ms and 108 ms. A trusted dealer is very cheap. It is also the assumption this paper exists to remove, and the full-domain pass is what Theorem 1 is proved about.

What the comparison establishes. Only two baselines bind both the key and the report after uplift—Cavern, which needs a third party and an honest majority that we do not assume, and this work. Among constructions that stay in the two-party dishonest-majority model, the certified scheme is between 66× and 1.3×10^5 cheaper in communication than the nearest equipped competitor, at a local-work cost of between 2 and 30× theirs and

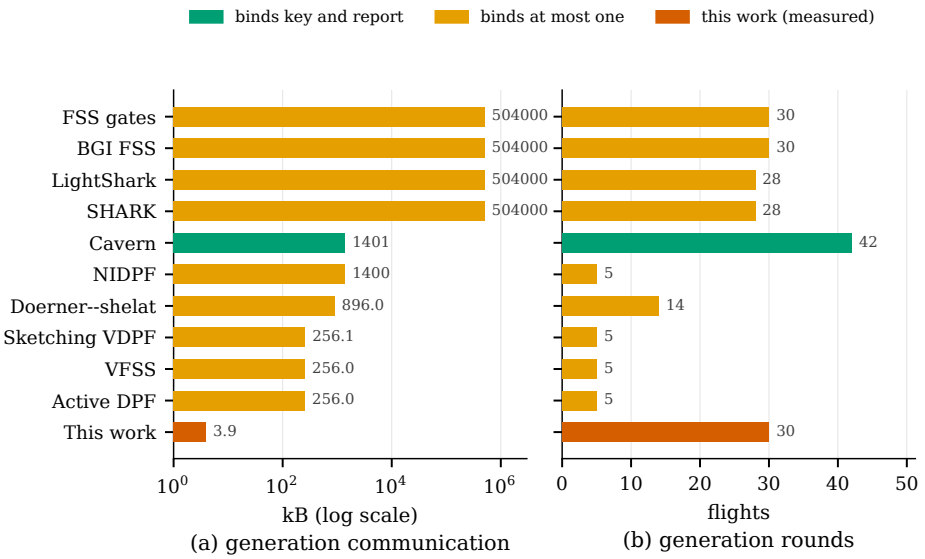


Fig. 15. Every baseline at $n = 14$ after uplift. Colour encodes what the scheme binds, not how well it scores. Communication spans six decades; rounds span one, and we are not at the good end of that panel.

Table 21. Two measured kernels, compiled, on the same host and the same expander: the textbook BGI dealer, our dealer-free certified generation, and the full-domain sketch of [2] at one and three repetitions. No projection appears in this table.

n	BGI, dealer	this work, dealer-free	full-domain eval.	sketch $\times 1$	sketch $\times 3$
10	1.1 μs	81.2 μs	62.5 μs	6.8 μs	20.4 μs
12	1.1 μs	336.4 μs	258.2 μs	27.3 μs	81.7 μs
14	1.2 μs	2.1 ms	1.1 ms	110.3 μs	328.4 μs
16	1.3 μs	5.8 ms	4.9 ms	456.6 μs	1.6 ms
18	1.4 μs	26.3 ms	20.5 ms	1.8 ms	5.3 ms
20	1.5 μs	108.0 ms	77.8 ms	7.9 ms	22.4 ms

no round-count advantage. That is the trade, and we would not present it as anything else.

7.13 Deployment note: PPML and secure model inference

The operators of Section 6 map onto the standard FSS-based inference stack without modification. Linear layers consume Λ-authenticated matrix triples; the non-linearities that dominate the round budget—ReLU, sign extraction, faithful truncation after fixed-point multiplication, and the spline approximations of sigmoid, GELU and softmax's exponential—are

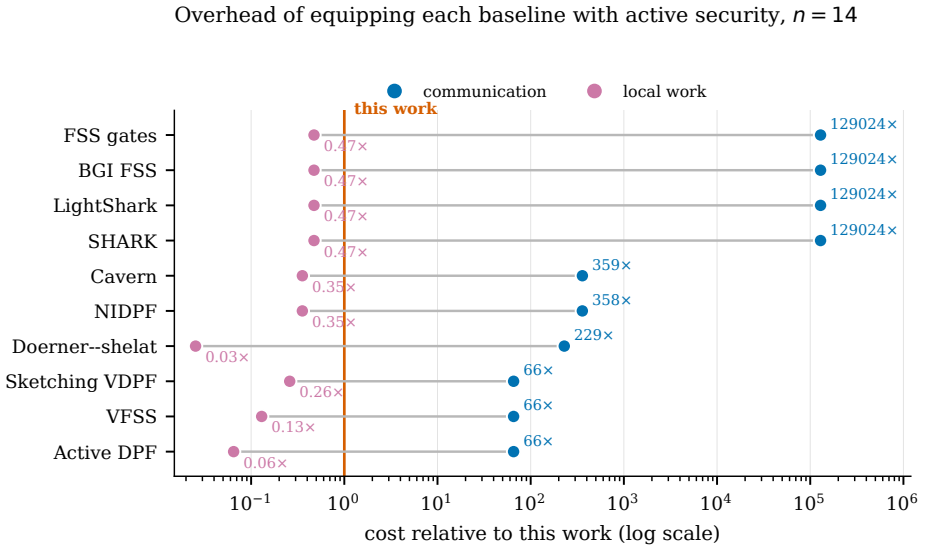


Fig. 16. The same data relative to this work. Communication ratios lie between 66× and 1.3×10^5; local-work ratios lie *below* one for every baseline, because our generation performs a full-domain pass that they do not. Both facts are properties of the same design decision.

all authenticated offset gates in the sense of Definition 6, and they evaluate locally, so a certified network has the same online round complexity as its semi-honest counterpart. The certification cost is confined to the input-independent phase, where dynamic cross-domain aggregation certifies an entire model's worth of gate keys under one challenge and a constant number of rounds; for transformer inference, where every block contributes thousands of independent activation gates, this is the regime in which the amortisation of Figure 8 is most favourable. The remaining obstacle to large language model inference is not the audit but the offset inventory: one certified instance per non-linear activation, which for a multi-billion-parameter model is a preprocessing question of storage and bandwidth rather than of security. We leave a full inference deployment to future work and make no performance claim about it here.

7.14 Threats to validity

The measurements come from one host and one toolchain. The native implementation is single-threaded and makes no use of the 24 cores available; a production system would parallelise across instances, which the aggregation schedule already makes embarrassingly parallel, so our latencies

should be read as per-core. The cross-implementation check establishes that the two agree on what they compute, not that either matches a third implementation of the specification. The $\mathcal{F}_{\mathsf{prep}}$-hybrid is standard but real: Table 7 converts measured unit counts into traffic using published per-unit costs, and a deployment that instantiates the pool differently will see different numbers. Finally, the adversarial campaign is a falsification instrument, not a proof; it agrees with Section 5 at every point we tested, which is the most an experiment can do.

8 Conclusion

Dealer-free function secret sharing with a malicious generator has been held back by a binding problem rather than an efficiency problem: nothing linked the key that a distributed generator sealed to the numbers its owner later fed into an audit, and nothing linked either of those to the shares it released at evaluation time. Generating the key for the paired function $(f_\theta, \Lambda f_\theta)$ resolves all three links at once. The tag plane is what makes an evaluation share self-authenticating, and it is also what makes a party unable to misreport the moments of its own key—the step that previously called for a vector commitment with linear openings and a proof system to go with it.

With fidelity in hand, three moment equations suffice, and we gave them a characteristic-free proof by reading off the coefficient of a square monomial. That proof made two facts visible that a characteristic-zero intuition hides: in characteristic two the support equation is blind to odd equal-weight supports and the position equation must carry the argument, and over a Galois ring the same coefficient condition survives the zero divisors provided the statistical margin is at least the payload precision. The construction is therefore correct over the algebras that fixed-point secure computation actually uses, with no non-additive lift anywhere.

The result is not leakage-free, and we have been explicit about why. One deviation class—an additive error consumed multiplicatively by a secret index bit—is invisible when that bit is zero, and we model it as a single-bit predicate rather than argue it away. The artifact measures it exactly: the audit rejects in precisely the trials where the attacked bit is one. Removing the last bit means binding a party's level sums to its own tree, which means proving relations about pseudorandom-generator images, which means a generic secure evaluation of the key-generation circuit. That trade is available and, we think, unattractive; making it precise is the natural next question.

Two further directions follow. The lifting theorem is stated for gates that post-process a constant number of DPF/DCF evaluations affinely, which covers the published catalogue but not gates that multiply two FSS outputs; a direct treatment of those would remove the last authenticated triples from the online phase. And the certification pass is $\Theta(N)$ local work by design, since it certifies a full-domain statement from $O(n)$ communication; whether a sublinear certifier exists for point and comparison functions, at comparable communication and without a proof system, is open.

References

1. Boneh, D., Boyle, E., Corrigan-Gibbs, H., Gilboa, N., Ishai, Y.: Zero-knowledge proofs on secret-shared data via fully linear PCPs. In: Advances in Cryptology – CRYPTO 2019. Lecture Notes in Computer Science, vol. 11694, pp. 67–97. Springer (2019). https://doi.org/10.1007/978-3-030-26954-8_3
2. Boneh, D., Boyle, E., Corrigan-Gibbs, H., Gilboa, N., Ishai, Y.: Lightweight techniques for private heavy hitters. In: IEEE Symposium on Security and Privacy (S&P). pp. 762–776 (2021). https://doi.org/10.1109/SP40001.2021.00048
3. Boyle, E., Chandran, N., Gilboa, N., Gupta, D., Ishai, Y., Kumar, N., Rathee, M.: Function secret sharing for mixed-mode and fixed-point secure computation. In: Advances in Cryptology – EUROCRYPT 2021. Lecture Notes in Computer Science, vol. 12697, pp. 871–900. Springer (2021). https://doi.org/10.1007/978-3-030-77886-6_30, Full version: IACR ePrint 2020/1392
4. Boyle, E., Couteau, G., Gilboa, N., Ishai, Y., Kohl, L., Rindal, P., Scholl, P.: Efficient two-round OT extension and silent non-interactive secure computation. In: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security (CCS). pp. 291–308. ACM (2019). https://doi.org/10.1145/3319535.3354255
5. Boyle, E., Couteau, G., Gilboa, N., Ishai, Y., Kohl, L., Scholl, P.: Efficient pseudorandom correlation generators: Silent OT extension and more. In: Advances in Cryptology – CRYPTO 2019. Lecture Notes in Computer Science, vol. 11694, pp. 489–518. Springer (2019). https://doi.org/10.1007/978-3-030-26954-8_16
6. Boyle, E., Devadas, L., Servan-Schreiber, S.: Non-interactive distributed point functions. In: Public-Key Cryptography – PKC 2025. Lecture Notes in Computer Science, vol. 15674, pp. 3–35. Springer (2025). https://doi.org/10.1007/978-3-031-91820-9_1
7. Boyle, E., Gilboa, N., Hamilis, M., Ishai, Y., Tu, Y.: Improved constructions for distributed multi-point functions. In: IEEE Symposium on Security and Privacy (S&P). pp. 2414–2432 (2025). https://doi.org/10.1109/SP61157.2025.00044
8. Boyle, E., Gilboa, N., Ishai, Y.: Function secret sharing. In: Advances in Cryptology – EUROCRYPT 2015. Lecture Notes in Computer Science, vol. 9057, pp. 337–367. Springer (2015). https://doi.org/10.1007/978-3-662-46803-6_12
9. Boyle, E., Gilboa, N., Ishai, Y.: Function secret sharing: Improvements and extensions. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS). pp. 1292–1303. ACM (2016). https://doi.org/10.1145/2976749.2978429

10. Boyle, E., Gilboa, N., Ishai, Y.: Secure computation with preprocessing via function secret sharing. In: Theory of Cryptography Conference (TCC). Lecture Notes in Computer Science, vol. 11891, pp. 341–371. Springer (2019). https://doi.org/10.1007/978-3-030-36030-6_14
11. Boyle, E., Gilboa, N., Ishai, Y., Kolobov, V.I.: Information-theoretic distributed point functions. In: 3rd Conference on Information-Theoretic Cryptography (ITC). LIPIcs, vol. 230, pp. 17:1–17:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik (2022). https://doi.org/10.4230/LIPIcs.ITC.2022.17
12. Boyle, E., Gilboa, N., Ishai, Y., Kolobov, V.I.: Programmable distributed point functions. In: Advances in Cryptology – CRYPTO 2022. Lecture Notes in Computer Science, vol. 13510, pp. 121–151. Springer (2022). https://doi.org/10.1007/978-3-031-15985-5_5
13. Bunn, P., Kushilevitz, E., Ostrovsky, R.: CNF-FSS and its applications. In: Public-Key Cryptography – PKC 2022. Lecture Notes in Computer Science, vol. 13177, pp. 283–314. Springer (2022). https://doi.org/10.1007/978-3-030-97121-2_11
14. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: 42nd Annual Symposium on Foundations of Computer Science (FOCS). pp. 136–145. IEEE Computer Society (2001). https://doi.org/10.1109/SFCS.2001.959888
15. Canetti, R., Rabin, T.: Universal composition with joint state. In: Advances in Cryptology – CRYPTO 2003. Lecture Notes in Computer Science, vol. 2729, pp. 265–281. Springer (2003). https://doi.org/10.1007/978-3-540-45146-4_16
16. de Castro, L., Polychroniadou, A.: Lightweight, maliciously secure verifiable function secret sharing. In: Advances in Cryptology – EUROCRYPT 2022. Lecture Notes in Computer Science, vol. 13275, pp. 150–179. Springer (2022). https://doi.org/10.1007/978-3-031-06944-4_6
17. Cramer, R., Damgård, I., Escudero, D., Scholl, P., Xing, C.: SPD$\mathbb{Z}_{2^k}$: Efficient MPC mod 2^k for dishonest majority. In: Advances in Cryptology – CRYPTO 2018. Lecture Notes in Computer Science, vol. 10992, pp. 769–798. Springer (2018). https://doi.org/10.1007/978-3-319-96881-0_26
18. Damgård, I., Pastro, V., Smart, N.P., Zakarias, S.: Multiparty computation from somewhat homomorphic encryption. In: Advances in Cryptology – CRYPTO 2012. Lecture Notes in Computer Science, vol. 7417, pp. 643–662. Springer (2012). https://doi.org/10.1007/978-3-642-32009-5_38
19. Damie, M., Hahn, F., Peter, A., Ramon, J.: DDH-based schemes for multi-party function secret sharing. In: Nordic Conference on Secure IT Systems (NordSec). Lecture Notes in Computer Science, vol. 16325, pp. 3–22. Springer (2026). https://doi.org/10.1007/978-3-032-14782-0_1
20. Doerner, J., Shelat, A.: Scaling ORAM for secure computation. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS). pp. 523–535. ACM (2017). https://doi.org/10.1145/3133956.3133967
21. Escudero, D., Ghosh, S., Keller, M., Rachuri, R., Scholl, P.: Improved primitives for MPC over mixed arithmetic-binary circuits. In: Advances in Cryptology – CRYPTO 2020. Lecture Notes in Computer Science, vol. 12171, pp. 823–852. Springer (2020). https://doi.org/10.1007/978-3-030-56880-1_29
22. Frederiksen, T.K., Keller, M., Orsini, E., Scholl, P.: A unified approach to MPC with preprocessing using OT. In: Advances in Cryptology – ASIACRYPT 2015. Lecture Notes in Computer Science, vol. 9452, pp. 711–735. Springer (2015). https://doi.org/10.1007/978-3-662-48797-6_29

23. Gilboa, N., Ishai, Y.: Distributed point functions and their applications. In: Advances in Cryptology – EUROCRYPT 2014. Lecture Notes in Computer Science, vol. 8441, pp. 640–658. Springer (2014). https://doi.org/10.1007/978-3-642-55220-5_35
24. Goel, A., Wang, M., Wang, Z.: Multiparty distributed point functions. In: Advances in Cryptology – CRYPTO 2025. pp. 140–173. Lecture Notes in Computer Science, Springer (2025). https://doi.org/10.1007/978-3-032-01884-7_5
25. Guo, C., Katz, J., Wang, X., Yu, Y.: Efficient and secure multiparty computation from fixed-key block ciphers. In: IEEE Symposium on Security and Privacy (S&P). pp. 825–841 (2020). https://doi.org/10.1109/SP40000.2020.00016
26. Guo, X., Yang, K., Wang, X., Zhang, W., Xie, X., Zhang, J., Liu, Z.: Half-tree: Halving the cost of tree expansion in COT and DPF. In: Advances in Cryptology – EUROCRYPT 2023. Lecture Notes in Computer Science, vol. 14004, pp. 330–362. Springer (2023). https://doi.org/10.1007/978-3-031-30545-0_12
27. Gupta, K., Chandran, N., Gupta, D., Katz, J., Sharma, R.: SHARK: Actively secure inference using function secret sharing. In: IEEE Symposium on Security and Privacy (S&P). pp. 2472–2490 (2025). https://doi.org/10.1109/SP61157.2025.00175, Full version: IACR ePrint 2025/716.
28. Gupta, K., Jawalkar, N., Mukherjee, A., Chandran, N., Gupta, D., Panwar, A., Sharma, R.: Sigma: Secure GPT inference with function secret sharing. Proceedings on Privacy Enhancing Technologies (PoPETs) **2024**(4), 61–79 (2024). https://doi.org/10.56553/popets-2024-0107
29. Gupta, K., Kumaraswamy, D., Chandran, N., Gupta, D.: LLAMA: A low latency math library for secure inference. Proceedings on Privacy Enhancing Technologies (PoPETs) **2022**(4), 274–294 (2022). https://doi.org/10.56553/popets-2022-0109
30. Ji, K., Zhang, B., Ren, K.: Fine-grained policy constraints for distributed point function. Cryptology ePrint Archive, Paper 2023/1672 (2023), https://eprint.iacr.org/2023/1672
31. Ji, K., Zhang, B., Zhou, H.S., Ren, K.: Distributed point function with constraints, revisited. Cryptology ePrint Archive, Paper 2024/937 (2024), https://eprint.iacr.org/2024/937
32. Keller, M., Orsini, E., Scholl, P.: Actively secure OT extension with optimal overhead. In: Advances in Cryptology – CRYPTO 2015. Lecture Notes in Computer Science, vol. 9215, pp. 724–741. Springer (2015). https://doi.org/10.1007/978-3-662-47989-6_35
33. Keller, M., Orsini, E., Scholl, P.: MASCOT: Faster malicious arithmetic secure computation with oblivious transfer. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS). pp. 830–842. ACM (2016). https://doi.org/10.1145/2976749.2978357
34. Koshiba, T.: Fourier-based verifiable function secret sharing. In: International Symposium on Information Theory and Its Applications (ISITA). pp. 442–446 (2020). https://doi.org/10.23919/ISITA50066.2020.9263257
35. Krips, T., Pullonen-Raudvere, P.: Multi-party distributed point functions with polylogarithmic key size from invariants of matrices. Cryptology ePrint Archive, Paper 2025/978 (2025), https://eprint.iacr.org/2025/978
36. Kruglik, S., Dau, S.H., Kiah, H.M., Wang, H., Zhang, L.F.: Verifiable information-theoretic function secret sharing. Cryptology ePrint Archive, Paper 2024/453 (2024), https://eprint.iacr.org/2024/453
37. Li, W.M., Zhang, L.F.: Towards efficient information-theoretic function secret sharing. IEEE Access **8**, 28512–28523 (2020). https://doi.org/10.1109/ACCESS.2020.2971722

38. Liu, Y., Zhang, L.F.: Cavern: Efficient honest-majority maliciously secure (2+1)-PC for $\mathbb{Z}_{2^n}$ via DPF. In: IEEE Symposium on Security and Privacy (S&P). pp. 1921–1936. IEEE (2026), https://eprint.iacr.org/2026/214, Only preprint available online; proceedings not yet published.
39. Mouris, D., Patton, C., Davis, H., Sarkar, P., Tsoutsos, N.G.: Mastic: Private weighted heavy-hitters and attribute-based metrics. Proceedings on Privacy Enhancing Technologies (PoPETs) **2025**(1), 290–319 (2025). https://doi.org/10.56553/popets-2025-0017
40. Roy, L.: SoftSpokenOT: Quieter OT extension from small-field silent VOLE in the minicrypt model. In: Advances in Cryptology – CRYPTO 2022. Lecture Notes in Computer Science, vol. 13507, pp. 657–687. Springer (2022). https://doi.org/10.1007/978-3-031-15802-5_23
41. Schoppmann, P., Gascón, A., Reichert, L., Raykova, M.: Distributed vector-OLE: Improved constructions and implementation. In: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security (CCS). pp. 1055–1072. ACM (2019). https://doi.org/10.1145/3319535.3363228
42. Storrier, K., Vadapalli, A., Lyons, A., Henry, R.: Grotto: Screaming fast $(2+1)$-PC for $\mathbb{Z}_{2^n}$ via $(2,2)$-DPFs. In: Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security (CCS). pp. 2143–2157. ACM (2023). https://doi.org/10.1145/3576915.3623147
43. Sun, Y., Liu, H., Yang, K., Yu, Y., Wang, X., Weng, C.: Committed vector oblivious linear evaluation and its applications. In: Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security (CCS). pp. 3635–3648. ACM (2025). https://doi.org/10.1145/3719027.3744887
44. Wagh, S.: Pika: Secure computation using function secret sharing over rings. Proceedings on Privacy Enhancing Technologies (PoPETs) **2022**(4), 351–377 (2022). https://doi.org/10.56553/popets-2022-0113
45. Wang, X., Ranellucci, S., Katz, J.: Authenticated garbling and efficient maliciously secure two-party computation. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS). pp. 21–37. ACM (2017). https://doi.org/10.1145/3133956.3134053
46. Yang, K., Sarkar, P., Weng, C., Wang, X.: QuickSilver: Efficient and affordable zero-knowledge proofs for circuits and polynomials over any field. In: Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security (CCS). pp. 2986–3001. ACM (2021). https://doi.org/10.1145/3460120.3484556
47. Yang, K., Weng, C., Lan, X., Zhang, J., Wang, X.: Ferret: Fast extension for correlated OT with small communication. In: Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security (CCS). pp. 1607–1626. ACM (2020). https://doi.org/10.1145/3372297.3417276
48. Zahur, S., Rosulek, M., Evans, D.: Two halves make a whole: Reducing data transfer in garbled circuits using half gates. In: Advances in Cryptology – EUROCRYPT 2015. LNCS, vol. 9057, pp. 220–250. Springer (2015). https://doi.org/10.1007/978-3-662-46803-6_8
49. Zeng, C., Feng, Q., He, D., Luo, M.: LightShark: Actively secure machine-learning inference based on lightweight authenticated distributed comparison function. Cryptology ePrint Archive, Paper 2026/1487 (2026), https://eprint.iacr.org/2026/1487
50. Zhang, W., Guo, X., Yang, K., Zhu, R., Yu, Y., Wang, X.: Efficient actively secure DPF and RAM-based 2PC with one-bit leakage. In: IEEE Symposium on Security and Privacy (S&P). pp. 561–577 (2024). https://doi.org/10.1109/SP54263.2024.00205, Full version: IACR ePrint 2024/426.

51. Zhu, L., Kong, L., Ning, X., Qu, X., Wang, J.: Publicly verifiable private information retrieval protocols based on function secret sharing. In: Information Security and Cryptology – Inscrypt 2025. Lecture Notes in Computer Science, vol. 16410, pp. 62–82. Springer, Singapore (2025). https://doi.org/10.1007/978-981-95-6209-1_4

A Authenticated Sub-Protocols

For completeness we record the two standard sub-protocols in the exact form the construction uses; both are the SPDZ originals [18,32] specialised to two parties and to an algebra with an exceptional set.

Algorithm 10 MacCheck over all values opened since the last check

Require: opened values $a_1, \dots, a_M$ with local MAC shares $\gamma_b(a_j)$

1: $\chi \leftarrow \mathcal{F}_{\mathsf{coin}}(\mathbb{K}^M)$; $\quad a \leftarrow \sum_j \chi_j a_j$

2: $\varsigma_b \leftarrow \sum_j \chi_j \gamma_b(a_j) - \Delta_b\, a$

3: Send $\mathsf{Com}(\varsigma_b)$; then open; accept iff $\varsigma_0 + \varsigma_1 = 0$.

Algorithm 11 CheckZero on authenticated values claimed to vanish

Require: $[\![v_1]\!], \dots, [\![v_M]\!]$

1: $\chi \leftarrow \mathcal{F}_{\mathsf{coin}}(\mathbb{K}^M)$; $\quad (\hat{v}_b, \hat{\gamma}_b) \leftarrow \big(\sum_j \chi_j v_{j,b},\ \sum_j \chi_j \gamma_b(v_j)\big)$

2: Send $\mathsf{Com}(\hat{v}_b \,\|\, \hat{\gamma}_b)$; then open.

3: Accept iff $\hat{v}_0 + \hat{v}_1 = 0$ **and** $\hat{\gamma}_0 + \hat{\gamma}_1 = 0$.

Checking both sums, rather than opening $\hat{v}$ and then verifying its MAC against Δ, keeps Δ out of every transcript: when the residuals do vanish the two revealed quantities are 0 and a uniformly random share sum that is also 0, and when they do not the protocol aborts before anything is used. Commitment before opening is what stops a rushing party from adapting its share to the honest one.

Lemma 10. *If some $v_j \neq 0$,* CheckZero *accepts with probability at most $1/|E| + \varepsilon_{\mathrm{com}}$; if the authenticated sharing of some v_j is inconsistent,* MacCheck *accepts with probability at most $1/|E| + \varepsilon_{\mathrm{com}}$.*

Proof. The commitment fixes both shares before χ is revealed, up to $\varepsilon_{\mathrm{com}}$. For a fixed nonzero vector v and uniform χ, $\sum_j \chi_j v_j$ is uniform on the ideal generated by the v_j of least valuation, which contains at least $|E|$ elements; the same argument applies to the MAC deviation. □

B Proof of Theorem 2

We give the hybrid argument in full. P_1 is corrupted; the environment $\mathcal{Z}$ chooses inputs, sees the honest party's output and the adversary's view, and outputs a bit.

H_0. The real execution of Π_{VeriFSS} in the $(\mathcal{F}_{\mathsf{prep}}, \mathcal{F}_{\mathsf{coin}})$-hybrid model.

H_1*: internalise the pool.* $\mathcal{S}$ plays $\mathcal{F}_{\mathsf{prep}}$ and $\mathcal{F}_{\mathsf{coin}}$. It therefore knows Δ, Λ, every triple, bit and mask, and every additive error that $\mathcal{A}$ requests on its own shares. This is a syntactic change: the two experiments are identical.

H_2*: replace the honest party's tag images.* By Eq. (6), the only quantities of P_0 that enter the transcript and are not already masked by pool values are the level sums $\delta_0^L, \delta_0^R, \epsilon_0^L, \epsilon_0^R$ and the leaf conversions $c_0^{\mathrm{val}}, c_0^{\mathrm{tag}}$, and each of these differs from the corresponding quantity of P_1 by exactly the on-path node's contribution. Replace each of the images $G(S_0[\alpha_{<i}])$ and $\mathsf{Cv}_*(S_0[\alpha])$ by uniform elements of $\mathbb{K}$. A distinguisher between H_1 and H_2 is a distinguisher for the correlation-robust pseudorandom generator, so $|\Pr[\mathsf{H}_1] - \Pr[\mathsf{H}_2]| \le \varepsilon_{\mathrm{prg}}$. In H_2 every correction word that P_0 helps to open is uniform and independent of θ and of Λ; this is Lemma 2.

H_3*: decide the verdict from the deviation vector.* $\mathcal{S}$ records, for every message of $\mathcal{A}$, the difference from the prescribed value, and classifies it by Lemma 8. Let S be the set of levels carrying a class (c) deviation with $\delta \neq 0$ and let bad be the event that a class (a) or (b) deviation occurred. H_3 replaces the actual outcome of $\mathsf{GenCheck}$ by

$$v = \begin{cases} \mathsf{reject} & \text{if } \mathsf{bad} \text{ or } \pi_S(\theta) = 0, \\ \mathsf{accept} & \text{otherwise.} \end{cases}$$

By Corollary 1, in the second case the sealed key pair is exactly an honest pair and the real $\mathsf{GenCheck}$ accepts with probability 1. In the first case the sealed pair differs from an honest pair, so by Theorem 1 the real $\mathsf{GenCheck}$ rejects except with probability $\varepsilon_{\mathrm{gen}}$; and by Lemma 10 a class (a) deviation additionally trips $\mathsf{MacCheck}$. Hence $|\Pr[\mathsf{H}_2] - \Pr[\mathsf{H}_3]| \le \varepsilon_{\mathrm{gen}} + \varepsilon_{\mathrm{com}}$.

H_4*: replace release by the ideal outputs.* On an evaluation request for points $x_1, \ldots, x_m$, H_4 sets the honest party's released shares to $v_j - y_1(x_j)$ and its tag shares to $\Lambda v_j - \tau_1(x_j)$, where $v_j = f_\theta(x_j)$ and (y_1, τ_1) are computed from $\mathcal{A}$'s key. When the verdict was accept the sealed pair is honest, so $v_j = g(x_j)$ and the substitution is the identity. The verdict of Π_{Release} is replaced by "accept iff $\mathcal{A}$ released the correct values"; by Lemma 9 this differs from the real verdict with probability at most $\varepsilon_{\mathrm{rep}}$.

H_5*: the ideal execution.* H_4 no longer uses θ_0 except through $\pi_S(\theta)$ and the values $f_\theta(x_j)$, both of which $\mathcal{F}_{\mathsf{aFSS}}$ supplies. Renaming the remaining computation as $\mathcal{S}$ gives the ideal execution. Summing the four differences yields the bound of Theorem 2. □

Remark 9 (Both corruption cases). The argument for a corrupted P_0 is symmetric: the construction is symmetric except for the root control values $T_0 = 1$, $T_1 = 0$ and the sign $s_b = (-1)^b$, neither of which enters any of the four hybrids. The one asymmetry worth noting is that in H_2 the replaced images belong to whichever party is honest, and Eq. (6) is symmetric in the two parties.

Remark 10 (Many keys under one Λ*).* Proposition 6 certifies m instances under one Δ and one Λ, and the evaluation exercises m up to 256. The hybrid extends without new ideas, but the reuse of Λ deserves an explicit word. Repeat H_2 once per instance: the images replaced are the honest party's on-path seeds and value words *of that instance*, drawn from independent root seeds, so the m replacements are independent and the loss is $m\,\varepsilon_{\mathrm{prg}}$. H_3 becomes a per-instance classification, and $\mathcal{S}$ submits the predicate π_{S_j} for instance j; since $\mathsf{CheckZero}$ is taken over the concatenation, the simulated verdict is the conjunction of the per-instance verdicts, which is what Corollary 1 and Proposition 6 jointly give. H_4 is unchanged: Lemma 9 is already stated for released points drawn from any number of keys under one Λ, and its proof uses only that Λ is uniform given the view, which Lemma 2 establishes for the whole session. The total loss is $m(\varepsilon_{\mathrm{gen}} + \varepsilon_{\mathrm{prg}}) + \varepsilon_{\mathrm{rep}} + \varepsilon_{\mathrm{com}}$.

Remark 11 (Concurrent sessions). Session identifiers are prefixed to every $\mathcal{F}_{\mathsf{prep}}$ request, every commitment and every challenge derivation, and pool units are never shared across sessions. Composition therefore follows from the composition theorem for the $\mathcal{F}_{\mathsf{prep}}$-hybrid model with joint state on the identifiers only [14,15].

C The Ring Instantiation

Fixed-point secure computation wants payloads in $\mathbb{Z}_{2^k}$. This appendix records the instantiation that supports them, together with the parameter constraint that Lemma 6 imposes.

C.1 Parameters

Take $\mathbb{K} = \mathrm{GR}(2^m, d) = \mathbb{Z}_{2^m}[X]/(h(X))$ with $m = k + s$, h a Hensel lift of a degree-d irreducible polynomial over $\mathbb{F}_2$, and challenges drawn from the

Teichmüller set $E = \{\sum_{i<d} e_i X^i : e_i \in \{0,1\}\}$. Payloads and MAC keys live in $\mathbb{K}$; the meaningful precision of a payload is k bits, i.e. two payloads that agree modulo 2^k are equal as fixed-point numbers. Three constraints fix the parameters.

1. $2^d \geq 2^\sigma \cdot 2n$, so that Lemma 6 gives soundness $2^{-\sigma}$;
2. $s \geq k$, which is what makes the valuation argument of Lemma 6 reach the payload precision;
3. $\varepsilon_{\mathrm{mac}} \leq 2/2^d$ from Lemma 10.

For $k = 32$, $\sigma = 40$ and $n \leq 20$ this is satisfied by $d = 48$, $s = 32$, $m = 64$; for $k = 64$ by $d = 48$, $s = 64$, $m = 128$. Elements then occupy $md/8$ bytes, which is the price the ring instantiation pays over the field one: the artifact's $\mathrm{GR}(2^{48}, 8)$ configuration is used for correctness testing rather than for the performance campaign.

C.2 Why $s \geq k$ is necessary and not merely convenient

Without it the conclusion of Case B in Lemma 6 fails. Take $m = 4$, $\beta = 0$ and $g = (4, 4, 8, 0, \ldots, 0)$ over $\mathbb{Z}_{16}$. Then $g_x(g_x - \beta) = g_x^2 = 0$ for every x and $\sum_x g_x = 16 = 0$, so the mass and support equations hold identically while $g \neq 0$. With $s \geq k$ the same vector is congruent to 0 modulo 2^k, so it is not a counterexample to the statement actually proved: the audit certifies $g \equiv \beta e_\alpha$ at the payload precision, which is all a fixed-point computation can observe. The example also shows why the field proof cannot simply be reused: "$g_x(g_x - z_1) = 0$ implies $g_x \in \{0, z_1\}$" is a field fact.

C.3 Schwartz–Zippel over the exceptional set

Lemma 1 is the version of [17]: for a nonzero Q of total degree D over a ring with exceptional set E, $\Pr_{\rho \leftarrow E^n}[Q(\rho) = 0] \leq D/|E|$. The proof is the usual induction, using only that a difference of two distinct exceptional elements is a unit, so that a nonzero univariate polynomial of degree D has at most D roots in E. Both applications in Lemma 6 are legitimate: the support polynomial T has degree at most $2n$ and a unit-free but nonzero coefficient, and the position polynomial P has coefficients in $\{0, \pm 1\}$, hence is nonzero modulo 2 and a fortiori over $\mathbb{Z}_{2^{k-v}}$.

C.4 Interaction with the tag plane

Lemma 3 needs $\Lambda\delta$ to be unpredictable for $\delta \neq 0$. Over a ring this requires care: if $\delta = 2^t\delta'$ with δ' a unit, then $\Lambda\delta$ is uniform on $2^t\mathbb{K}$, which has

$2^{d(m-t)} \geq 2^d$ elements, so the bound $1/|E|$ still holds for every $t < m$. A deviation with $\delta \equiv 0 \pmod{2^m}$ is no deviation at all. The same observation applies to Lemma 9.

C.5 What does *not* work

For the record, the following three shortcuts all fail, and the artifact contains a regression test for each.

- Keeping payloads in $\mathbb{Z}_{2^{64}}$ and authenticating their images under $x \mapsto (x, 0) \in \mathrm{GF}(2^{128})$: the map is not additive, so Eq. (1) fails on the first addition (Remark 2).
- Drawing challenges from all of $\mathrm{GR}(2^m, d)$ rather than from E: differences of non-exceptional elements can be zero divisors and the root count collapses.
- Writing the integer literal 2 where the field element $1 + 1$ is meant. In $\mathrm{GF}(2^\kappa)$ the literal denotes X, and the borrow chain of Algorithm 6 together with the ripple-carry adder of Algorithm 8 silently compute the wrong index.

D Gate Specifications

Table 22 instantiates Theorem 3 for the catalogue of [3] and for the table gate of Construction 1. "Instances" counts certified DPF/DCF keys; "constants" counts Λ-authenticated key constants, each costing one offline multiplication; online communication is always the single opening of the masked input, and online rounds are always one.

Reading the table. Every row is an instance of Theorem 3: the evaluation is a public affine combination of value planes at publicly derived points, so the identical combination of tag planes authenticates the result. The "constants" column counts the wrap corrections w_c of Eq. (9), which Algorithm 8 produces as carry-outs; each must be promoted to a Λ-sharing by one multiplication with $[\![\Lambda]\!]$ before it can be added to a gate output.

Splines. For an m-piece spline with public polynomials P_j of degree $\leq D$, the key carries $D + 1$ coefficient planes. The constant term depends on the offset, since $P_j(x) = P_j(\hat{x} - r_{\mathrm{in}})$ expands to a polynomial in $\hat{x}$ whose coefficients involve r_{in}; those coefficients are authenticated key constants and are covered by the same column. Evaluation computes $\sum_{k \leq D} a_k \hat{x}^k$ locally with public $\hat{x}$, which is affine in the a_k and therefore preserves the tag plane.

Table 22. Certified instantiations of the standard FSS gates. $N = 2^n$, ℓ is a public shift amount and m a number of pieces or intervals.

gate	realisation	instances	constants	online work
zero test $\mathbb{1}\{x = 0\}$	DPF at r_{in}	1	0	$O(n)$
comparison $\mathbb{1}\{x < c\}$	Eq. (9)	2	1	$O(n)$
interval $\mathbb{1}\{p \le x \le q\}$	Eq. (10)	2	1	$O(n)$
m-interval containment	shared endpoints	$m{+}1$	m	$O(mn)$
m-piece spline	MIC + coefficient planes	$m{+}1$	m	$O(mn)$
ReLU	two-piece spline	2	1	$O(n)$
logical shift by ℓ	interval on the high part	2	1	$O(n)$
arithmetic shift by ℓ	logical shift + sign	3	2	$O(n)$
bit decomposition	n parallel interval tests	$2n$	n	$O(n^2)$
public interval variants	no offset on the bound	1	0	$O(n)$
table $T : [N] \to \mathbb{K}$	Construction 1	1	0	$\Theta(N)$

Gates outside the theorem. A gate whose evaluation multiplies two FSS outputs—for example a fixed-point product of two activations without an intervening opening—is not a public affine post-processing and needs an authenticated Beaver triple in the online phase, at two openings and one round. This is the only place where the certified layer costs an online round that the semi-honest layer does not, and it is inherited from the arithmetic layer rather than from certification.

E Pseudocode for the Gate Catalogue

Table 22 states *what* each gate costs; this appendix states *how* each one is built, in the dealer-free two-party actively secure framework of Section 4. Every algorithm below is written against the same three-line interface, so that the reader can check the uniformity claim of Theorem 3 row by row rather than take it on trust.

The interface. A gate is declared once, offline, and evaluated many times, online.

– Declare runs inside the authenticated layer. It may call $\mathcal{F}_{\mathsf{prep}}$, ShiftIndex (Algorithm 8), Π_{GenTrace} (Algorithms 2 and 4) and the authenticated multiplication Π_{Mul} (Algorithm 3). It outputs a *gate descriptor*

$$\mathrm{G} = \big(\{(I_j, \varphi_j, c_j)\}_{j \le k},\ \{[\![d_i]\!]\}_{i \le m},\ \text{public coefficient maps}\big),$$

where I_j is a certified DPF or DCF instance, $\varphi_j : [N] \to [N]$ is a public map from the masked input to the evaluation point, c_j is a public coefficient (possibly a function of $\hat{x}$), and $[\![d_i]\!]$ are Λ-authenticated key constants.

- Certify appends the six residuals of each I_j to the global ledger. It is identical for all gates and is never restated below; it is the only place where a gate touches the network after declaration.
- $\mathsf{Eval}(\hat{x})$ is local: $o_b = \sum_j c_j(\hat{x})\, y_b^{(j)}(\varphi_j(\hat{x})) + \sum_i c'_i(\hat{x})\, d_{i,b}$ on the value plane and the same combination on the tag plane.

Throughout, $N = 2^n$; $[\![r_1]\!], \ldots, [\![r_n]\!]$ are the MSB-first authenticated bits of the input offset r_{in} drawn once per gate from $\mathcal{F}_{\mathsf{prep}}$; A_c abbreviates the certified DCF whose special index is $(r_{\text{in}} + c) \bmod N$, as in Eq. (9); and w_c denotes the carry-out of $\textsc{ShiftIndex}(\cdot; c)$, promoted to a Λ-sharing by one call to Π_{Mul} with $[\![\Lambda]\!]$. We write $\lfloor c/N \rfloor$ for the *public* wrap count that arises when a threshold exceeds the domain; it is a public constant and needs no multiplication.

E.1 The two families

The eleven rows of Table 22 fall into two families, and the split is not cosmetic.

The *threshold family* — zero test, comparison, interval containment, multiple interval containment, public-bound intervals, the shifts and bit decomposition — evaluates a constant number of DPF/DCF instances at $O(1)$ publicly derived points and combines them with public coefficients. Online work is $O(n)$ per instance, matching [3].

The *table family* — splines, ReLU, and arbitrary lookup — uses Construction 1: a single certified DPF, contracted against a public table. Online work is $\Theta(N)$, but the gate is exact over *any* payload algebra.

Remark 12 (Why splines are realised in the table family). The DCF route to a spline computes $\sum_j \mathbb{1}\{x \in J_j\}\,(a_j\hat{x} + c_j - a_j r_{\text{in}})$. Over a payload *ring* $\mathbb{Z}_N$ this is exactly Definition 7: $\hat{x} - r_{\text{in}} = x$ holds identically. Over a payload *field* $\mathbb{K}$ with $|\mathbb{K}| > N$ the identity holds only modulo N, and the correction term $a_j N \cdot \mathbb{1}\{\hat{x} < r_{\text{in}}\}$ is a product of two secret-shared quantities — the piece indicator and the wrap bit — hence bilinear, hence outside Definition 7. It would cost one online triple and one extra round. The table gate avoids the issue entirely by never performing arithmetic on $\hat{x}$, at the price of $\Theta(N)$ local work. Our artifact takes the table route for both profiles, so the semi-honest and certified numbers of Table 10 are

measured on identical circuits and the ratio is meaningful. For $n \geq 18$ the ring instantiation of Section 2.4 restores the $O(mn)$ route.

E.2 Threshold family

Algorithm 12 $\mathsf{Declare}^{\mathrm{ZT}}$ — zero test $\mathbb{1}\{x = 0\}$

Require: domain n, authenticated offset bits $[\![r_1..r_n]\!]$, key $[\![\Lambda]\!]$
1: $I \leftarrow \Pi^{\mathsf{DPF}}_{\mathsf{GenTrace}}([\![r_1..r_n]\!], [\![\beta{=}1]\!], [\![\Lambda]\!])$ ▷ Algorithm 2
2: **return** $\mathrm{G} = \big(\{(I,\ \varphi(\hat{x}) = \hat{x},\ 1)\},\ \emptyset\big)$

3: **function** $\textsc{Eval}(\hat{x})$
4: **return** $\big(y_b(\hat{x}),\ \tau_b(\hat{x})\big)$ ▷ $g(\hat{x}) = \mathbb{1}\{\hat{x} = r_{\mathrm{in}}\} = \mathbb{1}\{x = 0\}$
5: **end function**

Algorithm 13 $\mathsf{Declare}^{<}$ — comparison $\mathbb{1}\{x < c\}$, public c

1: $([\![u_1..u_n]\!], [\![w_c]\!]) \leftarrow \textsc{ShiftIndex}([\![r_1..r_n]\!];\ c \bmod N)$
2: $I_c \leftarrow \Pi^{\mathsf{DCF}}_{\mathsf{GenTrace}}([\![u_1..u_n]\!], [\![1]\!], [\![\Lambda]\!])$; $I_0 \leftarrow \Pi^{\mathsf{DCF}}_{\mathsf{GenTrace}}([\![r_1..r_n]\!], [\![1]\!], [\![\Lambda]\!])$
3: $[\![d]\!] \leftarrow \Pi_{\mathsf{Mul}}([\![w_c]\!], [\![\Lambda]\!])$ ▷ promote the carry to a Λ-sharing
4: **return** $\mathrm{G} = \big(\{(I_c, \hat{x}, +1), (I_0, \hat{x}, -1)\},\ \{([\![w_c]\!], [\![d]\!])\},\ \lfloor c/N \rfloor\big)$

5: **function** $\textsc{Eval}(\hat{x})$
6: $o_b \leftarrow y_b^{(c)}(\hat{x}) - y_b^{(0)}(\hat{x}) + w_{c,b} + \lfloor c/N \rfloor \cdot \mathbb{1}\{b = 0\}$
7: $\varpi_b \leftarrow \tau_b^{(c)}(\hat{x}) - \tau_b^{(0)}(\hat{x}) + d_b + \lfloor c/N \rfloor \cdot \Lambda_b$
8: **return** (o_b, ϖ_b)
9: **end function**

The public wrap count $\lfloor c/N \rfloor$ on line 5 of Algorithm 13 is the detail that a naive implementation misses. When the shifted threshold is reduced modulo N the adder's carry-out reports the *residual* wrap only; the quotient is a public integer that must be added separately. It is exactly 1 in the degenerate case $c = N$ used by multiple interval containment when the last interval reaches the top of the domain, and our reference implementation regressed on precisely that input until the quotient was reinstated.

Algorithm 14 $\mathsf{Declare}^{\mathrm{IC}}$ — interval containment $\mathbb{1}\{p \leq x \leq q\}$

1: declare A_{q+1} and A_p as in Algorithm 13, obtaining $[\![w_{q+1}]\!], [\![w_p]\!]$
2: $[\![W]\!] \leftarrow [\![w_{q+1}]\!] - [\![w_p]\!]$; $[\![d]\!] \leftarrow \Pi_{\mathsf{Mul}}([\![W]\!], [\![\Lambda]\!])$
3: **return** $\mathrm{G} = \big(\{(A_{q+1}, \hat{x}, +1), (A_p, \hat{x}, -1)\}, \{([\![W]\!], [\![d]\!])\},\ \lfloor (q{+}1)/N \rfloor - \lfloor p/N \rfloor\big)$

4: **function** $\textsc{Eval}(\hat{x})$
5: **return** $\big(A_{q+1,b}(\hat{x}) - A_{p,b}(\hat{x}) + W_b + \kappa \cdot \mathbb{1}\{b{=}0\},\ \ldots\big)$, κ the public wrap difference
6: **end function**

Algorithm 15 $\mathsf{Declare}^{\mathrm{MIC}}$ — multiple interval containment for $J_1, \ldots, J_m$, $J_j = [p_j, q_j]$

Require: intervals sorted so that consecutive endpoints coincide, $q_j + 1 = p_{j+1}$

$\mathcal{T} \leftarrow \{p_1, q_1{+}1, \ldots, q_m{+}1\}$ ▷ $m+1$ distinct thresholds, not $2m$

for $t \in \mathcal{T}$ **do**

  $(\llbracket u^{(t)} \rrbracket, \llbracket w_t \rrbracket) \leftarrow \textsc{ShiftIndex}(\llbracket r_1..r_n \rrbracket;\, t \bmod N)$; $A_t \leftarrow \Pi^{\mathsf{DCF}}_{\mathsf{GenTrace}}(\llbracket u^{(t)} \rrbracket, \llbracket 1 \rrbracket, \llbracket \Lambda \rrbracket)$

end for

$\llbracket d_j \rrbracket \leftarrow \Pi_{\mathsf{Mul}}(\llbracket w_{q_j+1} \rrbracket - \llbracket w_{p_j} \rrbracket, \llbracket \Lambda \rrbracket)$ for $j \le m$ ▷ all m in one round

return G with the m output coordinates $\mathrm{IC}_j = A_{q_j+1} - A_{p_j} + (w_{q_j+1} - w_{p_j}) + \kappa_j$

Sharing endpoints is what makes multiple interval containment cost $m+1$ instances rather than $2m$; the same economy appears in [3], and certification inherits it because the ledger is per instance, not per interval.

Algorithm 16 $\mathsf{Declare}^{\gg \mathrm{L}}$ — logical right shift by ℓ; $\mathsf{Declare}^{\gg \mathrm{A}}$ adds the sign correction

logical: declare the table gate of Algorithm 19 with $T[v] = v \gg \ell$ ▷ exact over any $\mathbb{K}$

arithmetic: $T[v] = \mathrm{sgn}_n(v) \gg \ell$ with sgn_n the two's-complement reading on n bits

$O(n)$-online variant, payload ring $\mathbb{Z}_N$ only:

$q \leftarrow 2^\ell$; declare A_{jq} for $j < N/q$ and output $\sum_j j \cdot \big(A_{(j+1)q} - A_{jq} + (w_{(j+1)q} - w_{jq})\big)$

arithmetic shift additionally subtracts $2^{n-\ell} \cdot \mathbb{1}\{x \ge N/2\}$, one further comparison instance

Algorithm 17 $\mathsf{Declare}^{\mathrm{BD}}$ — bit decomposition $x \mapsto (x_{n-1}, \ldots, x_0)$

for $i = 0$ **to** $n-1$ **do**

  output coordinate i is the multiple interval containment for the 2^{n-1-i} intervals on which bit i equals one

end for

implementation note: all coordinates share the *same* offset r_{in}, hence the same instance pool; the artifact realises the whole decomposition with one table gate whose entries are the packed bit vectors, at $\Theta(N)$ instead of $O(n^2)$

Algorithm 18 $\mathsf{Declare}^{\mathrm{PI}}$ — public-bound interval $\mathbb{1}\{\hat{x} < c\}$, no offset on the bound

$I \leftarrow \Pi^{\mathsf{DCF}}_{\mathsf{GenTrace}}(\llbracket r_1..r_n \rrbracket, \llbracket 1 \rrbracket, \llbracket \Lambda \rrbracket)$

return $\mathrm{G} = \big(\{(I,\ \varphi(\hat{x}) = \min(\hat{x}, c),\ 1)\}, \emptyset\big)$ ▷ one instance, no wrap constant

E.3 Table family

Algorithm 19 $\mathsf{Declare}^{\mathrm{LUT}}$ — arbitrary table $T : [N] \to \mathbb{K}$ (Construction 1)

$I \leftarrow \Pi^{\mathsf{DPF}}_{\mathsf{GenTrace}}(\llbracket r_1..r_n \rrbracket, \llbracket 1 \rrbracket, \llbracket \Lambda \rrbracket)$
$(y_b, \tau_b) \leftarrow \mathsf{EvalAll}(k_b)$ ▷ once, offline; $\Theta(N)$ memory

function $\textsc{Eval}(\hat{x})$
  $o_b \leftarrow \sum_{u \in [N]} T[(\hat{x} - u) \bmod N] \cdot y_b(u)$
  $\varpi_b \leftarrow \sum_{u \in [N]} T[(\hat{x} - u) \bmod N] \cdot \tau_b(u)$
  return (o_b, ϖ_b) ▷ no communication beyond the opening of $\hat{x}$
end function

Algorithm 20 $\mathsf{Declare}^{\mathrm{spline}}$ — m-piece spline; ReLU is $m = 2$

Require: breakpoints $b_1 < \cdots < b_{m-1}$ and public polynomials $P_1, \ldots, P_m$ of degree $\leq D$
$T[v] \leftarrow P_{j(v)}\big(\mathrm{sgn}_n(v)\big)$ where $j(v)$ is the piece containing $\mathrm{sgn}_n(v)$
return $\mathsf{Declare}^{\mathrm{LUT}}(T)$ ▷ one instance, exact over $\mathbb{K}$; see Remark 12

$O(mn)$-online variant, payload ring $\mathbb{Z}_N$ only:
declare the MIC of Algorithm 15 on the m pieces, once with payload 1 (slope plane) and once with payload $\llbracket \Delta c_j \rrbracket$ (constant plane)
$\mathsf{Eval}(\hat{x}) = \hat{x} \cdot \sum_j \Delta a_j \, \mathrm{IC}_{j,b} + \sum_j \Delta c_j \, \mathrm{IC}_{j,b}$, with $\Delta a_j = a_j - a_{j-1}$ telescoped so that only m terms survive

Algorithm 21 $\mathsf{Certify}$ — identical for every gate

Require: gate descriptors $\mathrm{G}_1, \ldots, \mathrm{G}_M$ with sealed instance pools
$\rho \leftarrow \mathcal{F}_{\mathsf{coin}}(E^{n_{\max}})$ ▷ one challenge for the whole batch of gates
for each instance I of each G_t **in parallel do**
  append $\mathsf{GenCheck}(I, \rho_{1..n_I})$ to the ledger $\mathcal{L}$ ▷ Algorithms 5, 6
end for
return $\mathsf{CheckZero}(\mathcal{L}) \wedge \mathsf{MacCheck}()$ ▷ constant rounds in M

Correctness of the catalogue. Each declaration above outputs a descriptor of the shape required by Definition 7, so Theorem 3 applies verbatim and the gate output is a Λ-authenticated sharing of $G(x)$ whenever certification accepts. The artifact checks this empirically: every gate is evaluated against a plaintext reference at 24 inputs spread over the domain, in both profiles and over both $\mathbb{F}_{p^2}$ and $\mathrm{GF}(2^{128})$, together with the invariant $\varpi_0 + \varpi_1 = \Lambda\,(o_0 + o_1)$ at every tested point. All eleven gates pass all four combinations; see the artifact for the commands.