KORD: Breaking the Key-Generation Bottleneck in Dealerless Function Secret Sharing via Protocol–Hardware Co-Design

Yijing Peng[†‡], Lin Liu[†*], Yujie Xue[†], Shaojing Fu[†], Shaoqing Li[†‡], Yaohua Wang[†‡], Rongmao Chen[†], Yang Guo[†‡]
† College of Computer Science and Technology, National University of Defense Technology, Changsha, China
‡ Key Laboratory of Advanced Microprocessor Chips and Systems, National University of Defense Technology, Changsha, China
∗ Corresponding author: liulin16@nudt.edu.cn

***Abstract*—Function secret sharing (FSS) has become a core primitive in privacy-preserving computation. However, each FSS invocation requires a fresh pair of function keys, typically produced by a trusted dealer—a dependency that expands the system's trust boundary and hinders practical deployment. Existing dealerless protocols eliminate this dependency, but incur substantial communication and a number of interaction rounds that grows linearly with the input bit-width, making key generation a major bottleneck.**

This paper presents KORD, a protocol–hardware co-design that dramatically reduces the cost of dealerless FSS key generation. At its core is a pair of chips that establish a common root of trust through mutual attestation and, within it, reconstruct FSS keys—eliminating the need for a dealer. This root of trust further forms a security boundary within which KORD restructures the generation protocol, collapsing the interaction of prior dealerless protocols into a single round, independent of GGM depth. A cross-key scheduling scheme then interleaves independent GGM-tree traversals, sustaining high computational throughput. KORD reduces key-generation communication per operation by $7{,}633$–$70{,}274\times$ over the state-of-the-art distributed FSS protocol. On a ZCU102 FPGA, cross-key interleaving lifts AES lane utilization from 8.3% to a board-measured 99.0%, for 11.60 million 32-bit DPF keys per second at 187.5 MHz on a 21.5K LUT engine (12.38 M at the separately validated 200 MHz operating point). On private ResNet-18 inference, key generation's share of end-to-end time falls to 10.1%, from 82.6% under a trusted dealer and over 96% under the dealerless baseline.

I. Introduction

Function secret sharing (FSS) has become a core primitive for privacy-preserving computation, widely adopted in secure two-party computation (2PC) and real-world privacy-sensitive systems [18], [21], [44] for its outstanding online efficiency and low communication overhead [5], [6], [15]. Similar to mainstream preprocessing-based 2PC frameworks, FSS-based secure computation decouples the computation into two stages, i.e., an offline preprocessing stage and an online evaluation stage. The offline stage performs all heavy cryptographic computation to generate input-independent correlated randomness, i.e., a key pair, whose production usually relies on a trusted dealer. The online stage only executes lightweight computing operations based on the pre-generated randomness, effectively reducing online interaction latency and communication pressure.

Despite this remarkable online performance, dealer-based FSS has inherent drawbacks: the secure generation and synchronized distribution of paired keys bottleneck end-to-end performance, and the mandatory trusted dealer introduces trust risks, a single point of failure, and deployment constraints that limit real-world applicability.

To eliminate the dealer dependency, researchers have proposed dealerless schemes [13], [17], [50]. Floram [13] pioneers this paradigm with the first dealerless distributed point function (DPF) construction. Half-Tree [17] further optimizes the DPF and presents a dealerless distributed comparison function (DCF). Unfortunately, neither supports arithmetic-shared inputs or outputs, which restricts their adoption in arithmetic-heavy scenarios such as privacy-preserving machine learning training and inference [12], [18].

Most recently, Xing et al. [50] propose a 2PC FSS scheme that generates DPF and DCF keys without a dealer, supporting both arithmetic-shared inputs and outputs. However, this scheme suffers from two fundamental performance bottlenecks that prevent its practical deployment in high-performance systems:

- *Linearly scaling communication overhead.* At the core of all practical FSS constructions lies the GGM tree [16], a binary tree of pseudorandom seeds expanded level by level over the ℓ-bit input domain. Each level of the tree embeds a small correction word, which in a dealer-based setting is directly generated by the trusted dealer. In a dealerless protocol, however, the correction word at each level must be jointly computed via interactive communication between the two parties. As a result, the number of interaction rounds and the total communication volume both grow linearly with the input bit width ℓ. As reported in [50], this leads to seconds of latency per key generation even over a wide-area network (WAN).
- *Full-tree expansion.* Since the target point of the FSS is secret-shared between the two parties, neither party knows which path in the GGM tree corresponds to the hidden point. The key-generation algorithm must therefore expand the full GGM tree with 2^ℓ leaf nodes for every key (Algorithm 2 of [50]). This results in exponential computational overhead that scales with the

input bit width, far outweighing the communication cost in practice.

In summary, state-of-the-art dealerless arithmetic FSS requires $\Theta(\ell)$ interaction rounds and $\Theta(\lambda\ell)$ communication bits per key (where λ is the security parameter), and incurs $\Theta(2^\ell)$ pseudorandom generator (PRG) calls due to full-tree expansion. These performance bottlenecks are the primary obstacle to deploying dealerless FSS in high-throughput, low-latency secure computing systems.

To address these fundamental bottlenecks, we rethink the design space of dealerless FSS key generation across protocol and hardware layers. Instead of emulating the dealer through repeated interaction, our approach enables local generation after a one-time coordination step. Moreover, rather than optimizing the latency of a single key, we exploit the independence among keys to maximize sustained key generation throughput. The price of this route is stated up front: KORD relocates trust from the dealer into minimal, fixed-function hardware, anchored in a hardware root of trust at setup.

This paper makes the following contributions:

- We propose KORD, a protocol–hardware co-design that generates FSS keys without a dealer by relocating trust into a pair of attested chips. Key communication drops to a single round, independent of the input length, and the computation per key falls from the $\Theta(2^\ell)$ expansion of the full tree in state-of-the-art distributed FSS to a $\Theta(\ell)$ walk of the special path. This is an algorithmic restructuring of dealerless generation itself rather than an acceleration of its primitives: reconstructing the point inside a protected boundary collapses the search for the hidden path. To our knowledge, KORD is the first dedicated hardware engine for dealerless FSS key generation.
- Within one key the GGM traversal is serial in depth, leaving the AES lanes at 8.3% utilization. KORD interleaves across keys to saturate the pipeline; a *provisioning law* in closed form sets the number of keys that suffices (the tiling knee M^*) and the saturated rate $fW/(c\ell)$, independent of latency. On silicon the law holds and utilization reaches 99.98%.
- We implement KORD and evaluate it across FSS primitives and private inference. Per operation, KORD cuts communication for key generation by 7,633–70,274× relative to the state-of-the-art dealerless protocol [50]. In ResNet-18 inference, key generation takes 82.6% of the total time under a trusted dealer and over 96% under the dealerless route; KORD brings the share to 10.1%.

II. Background and Threat Model

A. FSS, DPF/DCF, and the GGM Construction

A notation summary appears in the supplementary material. A two-party FSS scheme [6], [15] for a function family $\mathcal{F} = \{f : \{0,1\}^\ell \to \mathbb{G}\}$ over an Abelian group $\mathbb{G}$ consists of two algorithms, $(\mathrm{Gen}, \mathrm{Eval})$. The key-generation algorithm $\mathrm{Gen}(1^\lambda, \hat{f})$ produces a key pair (k_0, k_1), while $\mathrm{Eval}(b, k_b, x)$ denotes the local evaluation performed by party P_b. Correctness requires

$$\mathrm{Eval}(0, k_0, x) + \mathrm{Eval}(1, k_1, x) = f(x) \quad \text{for all } x \in \{0,1\}^\ell. \tag{1}$$

Security requires the distribution of either key in isolation to be simulatable given only the public parameters $(\lambda, \ell, \mathbb{G})$.

Distributed point functions (DPFs) [6], [15] and distributed comparison functions (DCFs) [2], [6] instantiate FSS for point and comparison functions, respectively:

$$f_{\alpha,\beta}(x) = \begin{cases} \beta & x = \alpha \\ 0 & \text{otherwise,} \end{cases} \qquad f^{<}_{\alpha,\beta}(x) = \begin{cases} \beta & x < \alpha \\ 0 & \text{otherwise.} \end{cases} \tag{2}$$

Here, $\alpha \in \{0,1\}^\ell$ is the hidden point and $\beta \in \mathbb{G}$ is the payload; throughout this paper, $\mathbb{G} = \mathbb{Z}_{2^{\ell_{\mathrm{out}}}}$. Equality and point lookup reduce to DPF evaluation, whereas comparison, ReLU, and truncation reduce to DCF evaluation [2], [19].

Both primitives use a binary GGM tree [16] of depth ℓ. A length-doubling PRG, $G(s) = (s^L, t^L, s^R, t^R)$, advances each party by one level for every input bit and is typically instantiated with fixed-key AES [33]. Traversal begins with an independent root seed s_b^0 and control bit $t_b^0 = b$ for each party. At level j, the generator expands *both* parties' level-$(j-1)$ seeds and computes the correction word

$$\mathrm{CW}_j = \big(s_0^{\overline{\alpha_j}} \oplus s_1^{\overline{\alpha_j}},\ t_0^L \oplus t_1^L \oplus \alpha_j \oplus 1,\ t_0^R \oplus t_1^R \oplus \alpha_j\big), \tag{3}$$

where α_j is the j-th bit of α and $\overline{\alpha_j}$ selects the child *off* the evaluation path (children are indexed $L \equiv 0$, $R \equiv 1$). Each party applies CW_j exactly when its control bit is set. Off the path, the parties' seeds and control bits coincide and cancel in Eq. (1); on the path, they diverge. The final correction word maps this divergence to shares of the payload:

$$\mathrm{CW}_{\ell+1} = (-1)^{t_1^\ell}\big(\beta - \mathrm{Convert}(s_0^\ell) + \mathrm{Convert}(s_1^\ell)\big), \tag{4}$$

where $\mathrm{Convert}$ is a pseudorandom map from seeds to $\mathbb{G}$. A DCF key additionally includes one value correction word CW_j^V per level, derived from the same joint seed state [6]; its correctness lemma appears in the supplementary material.

A DPF key share occupies $\lambda + \ell(\lambda + 2) + \ell_{\mathrm{out}}$ bits, while a DCF key share adds one ℓ_{out}-bit value correction word per level [2], [6]. Two structural properties determine key-generation cost. *(P1) Depth-serial expansion:* level j consumes the seed produced at level $j-1$, making each key a serial chain of ℓ dependent AES calls and creating a hardware latency wall. *(P2) One-time keys:* α is bound to the fresh mask of the value the gate consumes; reusing one mask (hence its key) to open two values breaks hiding, so any key source must sustain the workload's full gate rate.

The online phase needs only one local evaluation and one masked-value opening per gate [6], [7], [19]; all remaining costs arise during key generation, which dealerless designs augment with the interaction to construct Eq. (3) at every level.

B. Threat Model and System Assumptions

Parties and corruption model. KORD assumes a *hybrid* model that is explicit about where each guarantee comes from. At the protocol level the two parties P_0, P_1 are *semi-honest*—each follows the protocol but may infer from its local view, consistent with existing dealerless FSS—and, as in attested-hardware MPC [52], the cryptographic security is carried by the protocol, not by the silicon. On top of this, KORD adds a single *minimal trusted-hardware* assumption: each fixed-function chip enforces a non-extractable boundary around a few primitive operations. This hardware assumption is stated separately and discharged separately (below and Section VI); it is not folded into the word "semi-honest." Protocol-level deviation is out of scope for the default channel and is rejected fail-closed by KORD-mac.

System components. Each party holds one fixed-function KORD chip (C_0 at P_0, C_1 at P_1); the chips execute a fixed generation datapath and expose only the protocol-defined interface, while the hosts merely relay and never touch secret-dependent generation.

Adversary capabilities. We consider adversaries controlling a semi-honest party or observing the channel: a party sees its own inputs, shares, emitted key shares, and transcript; the network sees transmitted messages; cryptographic primitives remain secure. Unlike prior dealerless FSS protocols, whose security is indifferent to hardware probing because no device holds the counterparty's secrets, KORD places the counterparty's secrets inside a chip the local party physically possesses; excluding invasive and side-channel attacks against that chip is therefore part of the minimal trusted-hardware assumption above, with implications analyzed in Section VI.

Provisioning and pairing trust. Before deployment, C_0 and C_1 establish the shared secret K_{ch} through a one-time secure initialization (Section IV-A). Any principal that fabricates both chips or conducts this pairing and retains K_{ch} could derive every subsequent root and decrypt every message. KORD's claim is therefore scoped precisely: no *online or per-key* principal is trusted, while pairing-time provisioning trust is explicit, one-shot, and analyzed in Section VI.

III. Motivation: The Key-Generation Bottleneck

Because FSS keys are one-time objects (Section II-A), key generation sits on the critical path of system scalability: a practical system needs a sustained generator matching the application's consumption rate—and meeting it *without* a trusted dealer introduces fundamental bottlenecks in both communication and computation.

A. The Dealerless FSS Design Space

Existing FSS key-generation approaches occupy different points in a design space characterized by three requirements: **(1)** eliminating the trusted dealer, **(2)** supporting interaction independent of depth, and **(3)** sustaining key supply at application rate. No existing route satisfies all three (Table I).

A *trusted dealer* [19] centrally generates both key shares at high throughput with no interaction per key, but observes the joint generation state—an additional trusted entity and single point of compromise.

TABLE I: Existing FSS key-generation routes, per key; shaded cells mark each route's shortfall.

Route	Party	Rds	Comm.	Work	keys/s
Dealer [19]	3rd party	0	$\lambda\ell$	$\Theta(\ell)$	0.36M
Dealerless [50]	none	$\Theta(\ell)$	$\Theta(\lambda\ell)$	$\Theta(2^\ell)$	10^1–10^2

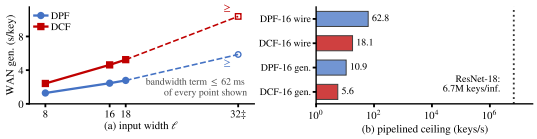


Fig. 1: The dealerless baseline [50] cannot supply keys at application rate, on its own published measurements (Table III; open markers and ‡: lower bound extrapolations past the artifact's ℓ=18 ceiling). (a) WAN latency per key at its own 40 Mbps/20 ms setting: bandwidth contributes $\leq$62 ms—the 9ℓ+5/16ℓ+7 round trip structure dominates. (b) Pipelined throughput from measured bytes and times per key (caliber B, §VII-A): 5.6–62.8 keys/s, against the 6.7 M keys one ResNet-18 inference consumes.

Dealerless FSS protocols [13], [50] remove this dependency by jointly computing each level's correction word, so interaction rounds grow with the tree depth ℓ and communication as $\Theta(\lambda\ell)$ per key—which their own measurements price at seconds per key over a WAN (Fig. 1a). Batching independent keys pipelines the rounds but not the bill: the bytes and generation time per key still cap pipelined throughput at tens of keys per second (Fig. 1b), more than five orders of magnitude short of what one private inference consumes.

A third route accelerates existing generation on CPUs, GPUs, or specialized cryptographic datapaths [18], [21], [49], speeding up primitives but leaving the structure untouched: the joint state stays where it was, and dealerless protocols still interact level by level.

No existing design simultaneously provides a dealerless deployment model, interaction independent of depth, and key generation at application rate. KORD reaches all three by adding an assumption the cryptographic routes avoid—a minimal hardware root of trust (Section II-B)—a condition that travels with every comparison in Section VII.

B. Structural Bottlenecks and Design Requirements

Three structural bottlenecks prevent dealerless FSS from scaling to high performance systems, and they yield four design requirements.

Observation 1: Joint generation state couples trust and communication. Generating a key requires a joint view of the function parameters and both GGM seed chains. Without a dealer, the parties must either place this state at another trusted entity—reintroducing external trust—or emulate it through distributed computation, whose communication grows with the GGM depth.

Design Requirements 1–2: Confine the joint generation state to a protected boundary without an external dealer, and interact

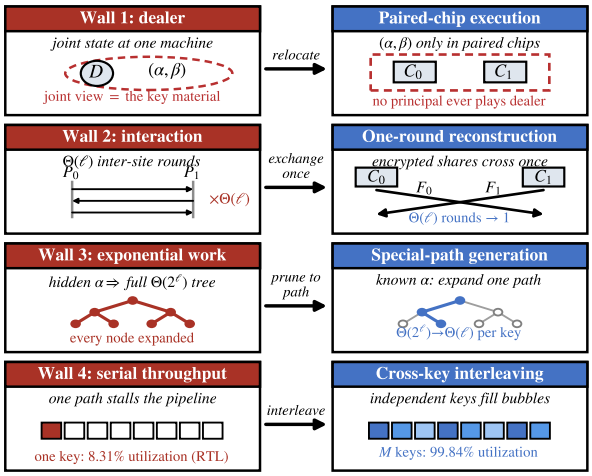


Fig. 2: From structural bottlenecks to KORD's design mechanisms: paired chips remove the external dealer, one encrypted exchange makes interaction independent of GGM depth, and interleaving independent keys fills the serial depth walk's dependency bubbles to sustain supply at application rate.

a constant number of rounds per key, independent of the depth ℓ.

Observation 2: A hidden path causes exponential generation work. The target point α remains secret shared, so neither party can identify the child each bit α_j selects and the generator must process the full GGM tree: $\Theta(2^\ell)$ work. Faster primitives cut the cost per node, not the nodes visited.

Design Requirement 3: Once α is available inside the protected boundary, expand only the selected path: $\Theta(2^\ell) \to \Theta(\ell)$ work.

Observation 3: The remaining linear traversal is serial in depth, but independent across keys. Special path generation removes the exponential work, yet level $j{+}1$ consumes level j's seed, so a single key cannot continuously feed a pipelined datapath; different keys, with independent chains, can progress concurrently.

Design Requirement 4: Interleave multiple keys in flight through one shared datapath, filling one key's dependency bubbles with another's ready work—optimizing sustained supply, not single key latency.

IV. KORD Protocol–Hardware Co-Design

KORD meets the four requirements of Section III-B in sequence: paired fixed function chips protect the joint state; one encrypted exchange lets both chips reconstruct the parameters locally; the reconstructed point enables linear special path generation; and cross-key interleaving hides the remaining serial depth. Section V gives the microarchitecture.

A. One Round Protocol Between Paired Chips

A host observes only its own input shares and the key share returned by its local chip C_b; it relays encrypted messages but cannot access the peer's shares, reconstructed parameters, roots, or intermediate GGM state. Each chip reconstructs joint state only transiently, removing the external dealer under a minimal hardware trust assumption.

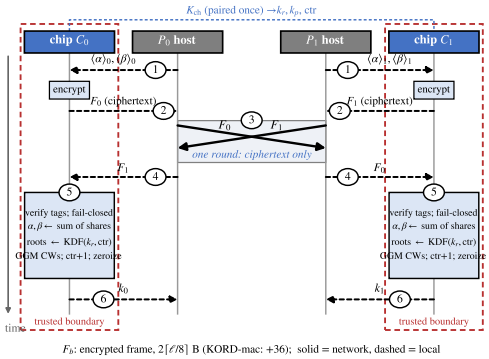


Fig. 3: The single exchange round in time order: each party submits its share pair to its own chip ①, which returns the ciphertext F_b—all that ever leaves the trust boundary ②; the hosts exchange F_0, F_1, the only network round ③, and forward the peer message inward ④; both chips verify, reconstruct, and expand alone ⑤, each emitting its own party's key ⑥. Red dashes mark the trust boundaries; the one-time pairing that provisions K_{ch} (§II-B) precedes this flow.

Before any key, C_0 and C_1 run the secure initialization of PPMLAC [52] once, establishing K_{ch}, channel state, and synchronized counters; domain separated derivation yields the root subkey k_r and channel keys. Given the same counter, both chips derive an identical root pair (s_0^0, s_1^0), so roots never enter host memory or cross the network.

The parties hold additive shares

$$\alpha = \langle\alpha\rangle_0 + \langle\alpha\rangle_1, \qquad \beta = \langle\beta\rangle_0 + \langle\beta\rangle_1.$$

Because the roots are derived locally, the peer's input shares are the only information still missing at either chip for each key. Figure 3 shows how KORD supplies these shares in one simultaneous exchange.

Under the default channel, C_b encrypts its two share fields as

$$F_b = (\langle\alpha\rangle_b \| \langle\beta\rangle_b) \oplus \mathrm{KS}(K_{\mathrm{otp},b}, ctr_b),$$

where $K_{\mathrm{otp},0} \neq K_{\mathrm{otp},1}$ are directional subkeys derived at pairing, so the two messages of one key never share a keystream block even under identical lockstep counters; the implemented SCU realizes this separation in the KDF domain. Each message occupies $\lceil \ell/8 \rceil + \lceil \ell_{\mathrm{out}}/8 \rceil$ bytes, or $4/8/16$ B when $\ell_{\mathrm{out}} = \ell = 16/32/64$. KORD-mac authenticates every message but preserves the same single round flow. After both messages are accepted, every remaining protocol step is local to the chips.

Algorithm 1 gives the core DPF steps; the DCF variant appears in the supplementary material. Lines 1–6 of Algorithm 1 correspond to Steps ①–⑤ in Figure 3: a party's input crosses only its own local interface, the channel secrets never leave the chips (so no host can open a message), and only ciphertext crosses the network. In KORD-mac, line 2 becomes $F_b \leftarrow \mathrm{AE}_{k_p}(\cdot)$ with mode, widths, and ctr_b bound as authenticated header fields, and line 5 verifies both tags and both transported counters—any failure aborts and fails closed, with no decryption and no engine dispatch. The roots

Algorithm 1 DPF Key Generation in One Round via Paired Trusted Hardware

Require: P_b holds $\langle\alpha\rangle_b, \langle\beta\rangle_b$; the paired chips hold the directional subkeys $K_{\mathrm{otp},0}, K_{\mathrm{otp},1}$, the subkeys k_p, k_r, and matching counters ctr
Ensure: each P_b obtains its standard DPF key share k_b
1: P_b**:** write $(\langle\alpha\rangle_b, \langle\beta\rangle_b)$ into the local chip C_b
2: C_b**:** $F_b \leftarrow (\langle\alpha\rangle_b \,\|\, \langle\beta\rangle_b) \oplus \mathrm{KS}(K_{\mathrm{otp},b}, ctr_b)$; return F_b to the host
3: $P_0 \leftrightarrow P_1$**:** exchange F_0 and F_1 (the single round)
4: P_b**:** forward the peer message into C_b
5: C_b**:** bind both messages to the lockstep counter ctr_b; on any mismatch abort, failing closed
6: C_b**:** decrypt both messages; $\alpha \leftarrow \langle\alpha\rangle_0 + \langle\alpha\rangle_1$, $\beta \leftarrow \langle\beta\rangle_0 + \langle\beta\rangle_1$
7: C_b**:** $(s_0^0, s_1^0) \leftarrow \mathrm{KDF}(k_r, ctr)$; $t_0^0 \leftarrow 0$, $t_1^0 \leftarrow 1$
8: **for** $j = 1$ **to** ℓ **do**
9: expand $G(s_0^{j-1})$, $G(s_1^{j-1})$; form $\mathrm{CW}_j = (s_{\mathrm{CW}}, t^L_{\mathrm{CW}}, t^R_{\mathrm{CW}})$ by Eq. (3)
10: advance each chain to the Keep child selected by α_j, applying the correction when the control bit is set: $s^j_{b'} \leftarrow s^{\mathsf{Keep}}_{b'} \oplus t^{j-1}_{b'} \cdot s_{\mathrm{CW}}$, $t^j_{b'} \leftarrow t^{\mathsf{Keep}}_{b'} \oplus t^{j-1}_{b'} \cdot t^{\mathsf{Keep}}_{\mathrm{CW}}$
11: **end for**
12: compute $\mathrm{CW}_{\ell+1}$ by Eq. (4)
13: C_b**:** $ctr \leftarrow ctr + 1$; $k_b \leftarrow (s_b^0, \mathrm{CW}_1, \ldots, \mathrm{CW}_{\ell+1})$; zeroize (α, β) and all seeds
14: **return** k_b to P_b only, over the local interface

derived in line 7 are identical in both chips by lockstep and are never transmitted. The generation loop is the algorithm's only $\Theta(\ell)$ stage, serial in depth within a key; independent keys interleave through it in hardware. DCF shares the entire admission, exchange, verification, and reconstruction path and differs only in its local correction word generation: each level expands four PRG blocks per chain, and the value lanes v^L, v^R feed the value corrections CW^V_j of each level.

KORD accepts α, β as additive shares because target workloads hold them only as joint secrets (a shared DORAM index, a compiled wire mask), letting KORD slot in as the key source unchanged; where one party may choose the mask, most residual communication disappears.

B. DPF and DCF Special Path Generation

Figure 4 shows the local phase that follows the exchange. Its Steps ①–⑤ correspond to the generation loops and final assembly in Algorithm 1 (DCF variant: supplementary material). Both chips execute the same generation procedure over identical roots and reconstructed parameters, produce the same correction word stream, and emit only their respective key shares.

In the distributed construction of Xing et al. [50], α remains hidden from both parties, preventing direct selection of the special GGM path and forcing $\Theta(2^\ell)$ full tree work. KORD reconstructs α only inside each protected chip. At level j, the chip expands both current seed chains, forms the correction from the children on the Lose side, and retains only the children selected by α_j.

For DPF, two chains and two children per chain require four PRG invocations per level, or 4ℓ in total. DCF uses the same selected path but additionally produces the value correction CW^V_j, requiring eight PRG invocations per level in the implemented datapath. KORD therefore reduces generation from $\Theta(2^\ell)$ full tree expansion to a $\Theta(\ell)$ special path traversal.

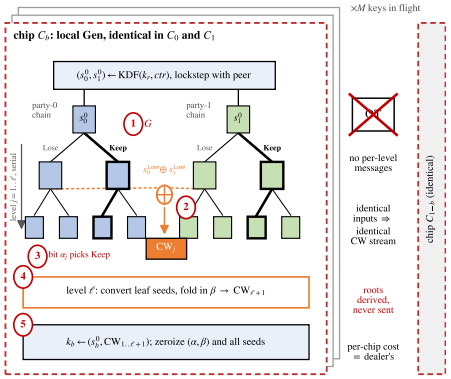


Fig. 4: After one exchange, each chip runs the identical local Gen over *both* parties' seed chains (4-leaf tree; roots derived in lockstep from (k_r, ctr), never transmitted): expand with AES-MMO G ①, XOR the Lose side into CW_j (Eq. (3)) ②, advance along α's path ③, convert the leaves (Eq. (4)) ④, return only the party's own key ⑤. Levels are serial within a key; nothing crosses between the chips.

The keys use the standard BGI layout [6]: after assembly each chip commits its counter, zeroizes reconstructed state, and returns only its party's share, so online evaluation and higher software layers remain unchanged.

C. Cross-Key Interleaving for Sustained Throughput

The loops remain serial in depth within one key (Figure 4). KORD therefore exploits independence *across* keys: hosts keep multiple keys in flight, the chips retain each key's secret state, and when one traversal waits on its previous level another issues to the shared PRG array—no generator per key. The hosts' role ends at submission, relay, and collection; every secret touching step executes inside the chips (Section V).

V. KORD Microarchitecture and Implementation

This section realizes the protocol of Section IV in hardware. Figure 5 shows the complete organization: the host software stack, the trust boundary of Section II-B (red dashed), and the eight stations ①–⑧ one key traverses—message submission, admission and unmasking (with authentication that fails closed under the KORD-mac extension), reconstruction, root derivation and slot admission, FCFS level issue, traversal of the W AES-MMO lanes, correction word extraction (Eqs. (3), (4)), and assembly with egress through the SCU key window—mapping one to one onto Algorithm 1. Three invariants govern the dataflow: *(i)* plaintext (α, β) exists only in the hatched reconstruction zone, latched once the SCU accepts both messages (in KORD-mac, after both tags verify) and zeroized once consumed; *(ii)* the dataflow never reverses—nothing downstream of the SCU emits data except through its egress, and the dispatch AND stays closed unless the SCU accepts the messages—in KORD-mac, until both tags verify—and the counter advances; *(iii)* every unit is a fixed function FSM with no loaded code and no DRAM. Section VI analyzes the boundary these invariants establish; Section VII-E reports all quantitative results.

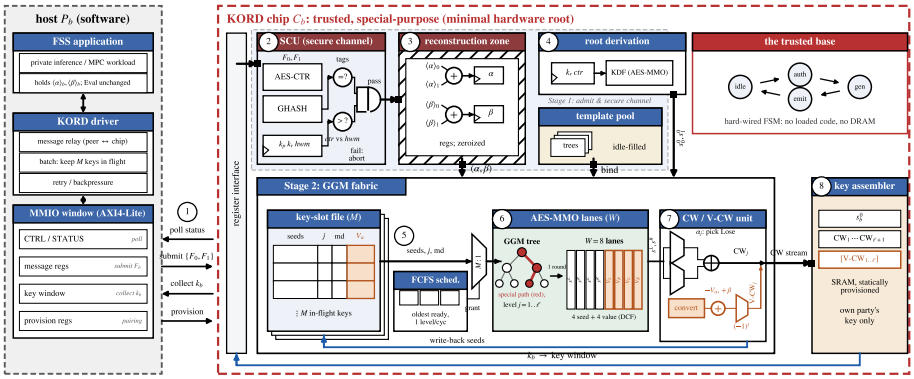


Fig. 5: The complete KORD hardware organization. The host software stack (gray dashed panel, semi-honest) sees only four memory mapped operations; the red dashed trust boundary encloses every hardware element on the key path. The D-shaped gate is the dispatch AND of invariant *(ii)*, failing closed until the SCU accepts both messages. Circled numbers 1–8 trace one key's dataflow; plaintext (α, β) exists only in the hatched reconstruction zone. Colors: red = control and trust boundary, blue = writeback rail, orange = DCF only (value lanes, V_α, value correction path), black = data; the lane microarchitecture opens up in Fig. 6(b).

A. Units at the Trust Boundary

The secure channel unit (SCU) is the sole message ingress and egress; every key crosses the boundary in the fixed ciphertext layout of Section IV-A. The prototype implements the default channel (session keystream XOR); KORD-mac hardens the same channel for actively tampered links with a GHASH/verify stage (AES-GCM per message) that fails closed: no decryption, no fabric dispatch, no observable engine state. The default datapath is a strict subset of the same hardware, so the engine is channel agnostic. In KORD-mac, mode and widths ride as associated data, so a message cannot replay against a different key type; all persistent channel state (pairing derived subkeys, monotonic counters) lives in the SCU, never in any key's GGM state.

The reconstruction zone holds the only plaintext copy of (α, β) that ever exists on either site, sized to one admission in flight. In parallel, root derivation runs the same AES-MMO primitive as the lanes under the dedicated domain key k_r, so both chips derive identical roots in lockstep and the roots are never transmitted. The front end admits a key only when a free slot and assembler space exist, otherwise backpressuring the register interface without partially modifying any generation state.

B. GGM Fabric and Host Integration

The fabric comprises three units (Figure 6a). The *key slot file* holds M keys in flight, each slot carrying one key's complete level context $(s_0^j, t_0^j, s_1^j, t_1^j, j, \mathrm{mode}, V_\alpha)$ plus readiness, age, and assembler tags, so no state migrates between units. The *W-lane AES-MMO array* accepts one block per cycle per lane (initiation interval 1); issued levels carry slot and mode tags, and the lanes know nothing of tree structure—one lane implementation serves root derivation, seed expansion, and value expansion through fixed keys on separate domains. The *CW/V-CW unit* selects Keep/Lose children by the reconstructed bit α_j, streams CW_j to the assembler, and writes the corrected child seeds back to the tagged slot; a completed key commits its counter and releases its slot only after its share reaches the output window.

The chip exposes four memory mapped operations—*provision*, *submit*, *poll*, *collect*—as the entire surface visible to the host (AXI4-Lite; Section VII-A). Host software does all flexible work touching no secret beyond its own party's (relay, batching to keep M keys in flight, retry); the chip does every step that touches one. Keys stream out as they complete, so generation and delivery overlap (egress: Section VII-E). The datapath reserves a split model that batches first and binds later; the prototype does not implement binding, and the verified path throughout this paper is direct generation.

C. Unified Architecture for DPF and DCF

Practical workloads consume both primitives concurrently. DPF and DCF differ in the data at each level, not the ordering: a DCF level is a DPF level plus a value chain (supplementary material), with identical admission, issue, return, and completion events, so one mode bit per slot suffices. A DPF level issues c=4 PRG calls; a DCF level fires all eight lanes (c=8), changing exactly the three orange components of Figure 5: admission ④ arms V_α, issue ⑤ adds the V_L/V_R lanes, and extraction ⑦ appends the value path, forming CW_j^V and folding the level into V_α. No other component distinguishes the modes. A DPF level on the unified engine idles the four value lanes—the honest cost of sharing one 4-domain AES block—so it issues one level per cycle in either mode; a build for DPF alone groups its lanes into independent 4-lane issue groups and retires proportionally more (§VII-E).

D. Cross-Key Interleaving Schedule

The protocol eliminated exponential work and rounds between the sites, but the expansion loop remains serial in depth *within* one key: level $j{+}1$ consumes the seeds of level j. In the implemented pipeline this dependency is a fixed latency at every level

$$L_{\mathrm{eff}} \;=\; L_{\mathrm{AES}} + 1 \;=\; 12 \text{ cycles}, \tag{5}$$

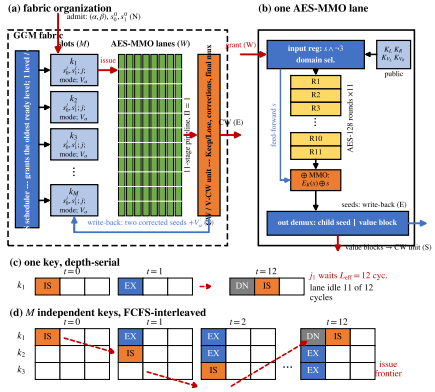


Fig. 6: The GGM fabric. (a) An FCFS scheduler issues one ready level per cycle from M key slots into the W-lane AES-MMO array; the CW unit writes corrected seeds back. (b) One lane: 11-stage AES round pipeline (II=1), MMO feedforward XOR, child demux. (c),(d) Cycle by cycle (IS/EX/DN = issuing/in lanes/done): one key idles the lane $L_{\text{eff}}-1$ of every L_{eff} cycles; M interleaved keys tile the array at the knee M^* (Eqs. (5), (6)).

the 11 lane stages plus one scheduling turnaround (Figure 6b): an isolated key issues at most one level per L_{eff} cycles and cannot feed a pipelined lane, independent of lane count or clock; replicating single chain engines scales area and state with replicas. The schedule instead fills each dependency bubble with a ready level from another key (Figure 6c): every cycle the FCFS scheduler grants the oldest ready slot, so utilization depends on the number of independent chains in flight, not the latency of one chain; FCFS suffices because all levels cost equally and age order bounds completion skew.

Provisioning M follows a closed form: the array is exactly tiled at

$$M^* \;=\; W \cdot L_{\text{eff}}/c \tag{6}$$

keys in flight (M^*=12 for the implemented c=W configurations; 24 for a W=8 build for DPF alone with 4-lane issue groups), beyond which the saturated rate is

$$\text{keys/s} \;=\; f\,W/(c\,\ell) \tag{7}$$

at clock f, with no latency term—pipeline depth moves the knee, not the throughput frontier. Since an FCFS grant cannot recover a slot lost to a phase collision between two ready keys, the implementation provisions four slots of headroom, M=16, at 52.6 B of context in flight per key (0.82 KiB total). Backpressure never perturbs the schedule: a full slot file or uncollected key window stalls admission at the SCU; admitted levels issue and retire in order. One limitation remains: the correction word extraction of station ⑦ is an unpipelined combinational reduction and is the engine's critical path, not the AES lanes; pipelining it would raise the clock and, by Eq. (6), need more slots without changing the rate law. The saturation curve, the knee, and the rate law's fit to measurement are evaluated in Sections VII-D and VII-E.

VI. Security Analysis

Under the threat model of Section II-B, KORD is *dealerless under a minimal hardware root*: after the one-time pairing, no principal outside the two chips observes the complete (α, β) or both output keys, and no online trusted party exists, nor one per key. Each chip reconstructs joint state only transiently inside its fixed function boundary. The DCF correctness lemma and the simulation security argument appear in the supplementary material.

Security claim. For each party P_b, an adversary controlling the party, its host, and the modeled communication path observes only P_b's input shares, public configuration, encrypted messages, and the returned key share k_b. This view is computationally simulatable by composing (i) authenticated pairing and attestation, (ii) the single exchange round (keystream confidentiality in the default channel; authenticated encryption in KORD-mac), and (iii) the ideal chip functionality releasing only k_b. The output simulation follows the FSS security of a single share, while multiple requests additionally rely on the PRF security of the derivation from K_{ch} across its instances. For fixed public widths and mode, the protocol exposes two messages of fixed length and one network round.

Channel and replay protection. The default channel matches the semi-honest model: it hides message contents but assumes faithful delivery; its directions are masked under $K_{\text{otp},0} \neq K_{\text{otp},1}$ (Section IV-A), so lockstep counters never reuse a keystream block. KORD-mac additionally authenticates mode, widths, direction, and counter; the SCU rejects invalid or replayed messages before reconstruction or dispatch. Neither guarantees availability: dropped messages may waste a key pair but reveal nothing.

Hardware trust boundary. KORD relocates rather than eliminates trust: each chip executes a fixed function, exposes only its party's key share, and zeroizes reconstructed state after generation. *Minimal* describes this bounded functionality and interface, not a formal lower bound. Like a fixed function attested root [52], the domain loads no code, so its attack surface is far narrower than a TEE dealer running loaded code; whether such relocation beats an online dealer is a deployment tradeoff, made explicit (supplementary trust comparison) rather than claimed universally. The relocated trust is symmetric and must hold against the chip's own host: each party's privacy rests on the boundary of the chip at the *other* party's site, and extraction of K_{ch} from either chip exposes both parties' masks and keys for the pairing's entire lifetime. Side channel silence toward the physically colocated owner is accordingly part of the assumed boundary, not an orthogonal exclusion: the persistent side channel targets are confined to the SCU's channel keys—GGM state for any key is single trace territory—so standard first order masking of the SCU datapath is the required hardening. This confines the hardening cost to a single AES core—a bounded fraction of the design (Fig. 5), not the generation fabric—rather than to the whole engine; we do not synthesize the masked variant here.

Provisioning ceremony. The residual trusted event is the

pairing itself (Section II-B); secure element practice shrinks it: independently sourced chips, an interactive pairing in which both chips contribute entropy so K_{ch} never exists outside them, and a witnessed transcript. What remains is device authenticity—the certificate root vouching the audited fixed function—one shot, unlike an online dealer observing every key.

Limitations. Pairing secrets and counters require persistent storage protected against rollback; the FPGA prototype guarantees monotonicity only within a session, and under the default tier a counter rolled back would reuse keystream—production parts need nonvolatile monotonic counters, and KORD-mac, which rejects replayed counters, is the recommendation wherever rollback protection is unavailable. Counter resynchronization requires an authenticated handshake advancing the counter; chip replacement requires a fresh pairing ($O(N^2)$ ceremonies for N sites). Invasive physical attacks remain out of scope, and the first order SCU masking that the assumed side channel boundary requires (§VI) is specified but not synthesized: the prototype demonstrates the datapath, not a production tamper boundary, and does not yet meet its own assumed physical boundary.

VII. Evaluation

We evaluate KORD against deployed key sources, asking four questions: does removing the key-generation bottleneck change behavior end to end (§VII-B); how do the protocol and the algorithm scale with input width (§VII-C); what does cross-key scheduling contribute (§VII-D); and what is the hardware cost, and does silicon confirm the modeled rates (§VII-E)?

A. Experimental Methodology

Platform. All hardware figures target one Xilinx Zynq UltraScale+ ZCU102 (XCZU9EG-2FFVB1156, Vivado 2022.2). Each engine, the secure channel unit, and the full online path are placed and routed out of context; $f_{\max}$ is extrapolated from WNS as $1000/(\text{period}-\text{WNS})$, and power is a vectorless estimate—synthesis estimates throughout; no ASIC is taped out. The same online path also runs on a physical ZCU102, driven over the processing system's JTAG debug access port with no operating system, storage, or DRAM on the key path. Board frequency is measured by retuning the PL clock in place across 100–250 MHz (each setting read back) and rerunning the golden vectors; a request generator on the chip keeps all slots full for saturated throughput.

References, baselines, and workloads. Key correctness is established in Vivado `xsim` against a C++ golden model linked to our `fss-ref` reference, and the same vectors are replayed on the board. All baseline numbers rerun published artifacts as genuine deployments of both parties on one host (Intel i7-12700KF, 20 threads, 32 GB): Xing et al. [50] as two simultaneous localhost processes, read from the framework's own byte and round counters (median of three); MP-SPDZ [23] as its `semi2k` offline phase at R=64, the smallest its binaries support; and the trusted dealer LLAMA stack [19] as the anchor for the full pipeline. Workloads: nine FSS building blocks and six trigonometric kernels at $\ell \in \{8, 16, 18\}$, private ResNet-18 [20] on ImageNet (ℓ=32, scale 10) over a 1 Gbps LAN and a 40 Mbps/100 ms application WAN (the baseline's bandwidth with our 100 ms system RTT; the primitive comparisons below use its own 40 Mbps/20 ms setting), and distributed ORAM at d=16.

TABLE II: Private ResNet-18 end to end (ImageNet, ℓ=32): trusted dealer LLAMA [19] vs. KORD, key-generation stage. Tags [M] measured, [S] synthesis-estimated, [D] derived (§VII-A); the online phase is identical by construction. †paired hardware root (§VI).

Key generation	LLAMA	KORD	Factor
Trusted party	yes	none†	—
Compute	18.54 s [M]	0.880 s [S]	21×
Wire per inf.	4.469 GB	50.4 MB	88.7×
Total, 1 Gbps [D]	54.29 s	1.283 s	42.3×
Total, 40 Mbps [D]‡	893.8 s	10.18 s	87.8×
Keygen share	**82.6%**	**10.1%**	—
Supply rate	0.363 M/s	6.25 M/s	17.2×
Keygen energy [E]	∼ 2.8 kJ	2.85 J	∼10^3×

Shared: 6,726,608 FSS trees; online 11.43 s / 60.6 MB [M]. ‡Wire time only (one 100 ms RTT for KORD), compute overlapped on both sides; charged serially as in the 1 Gbps row: 912.3 s vs. 11.06 s (82.5×). Energy [E] is order of magnitude only (vectorless power vs. whole server attribution).

Provenance and calibers. Every number carries a provenance tag—*measured*, *board-measured*, *synthesis-estimated*, or *board-confirmed* (rate law and clock both board-measured). Hardware speedups are claimed only at system scale, as lower bounds corroborated on the board (§VII-E); figures per operation report communication and latency reductions, not speedups. Before timing we removed one debug print from the baseline's timing region (neither logic nor byte counts change). The dealerless baseline uses two calibers: *A*, its own formula (9ℓ+5/16ℓ+7 rounds, λ=128), favorable to it; *B*, our reproduction (79.6/276.8 KB per DPF/DCF key at ℓ=16); above the artifact's ℓ=18 crash point, wider figures extrapolate (A by formula, B by a linear fit stable in ratio to A). Comparisons use the baseline's own links (10 Gbps/0.05 ms LAN; 40 Mbps/20 ms WAN); `EvalAll` on the party side is not modeled since KORD changes generation, not evaluation. The RTL, golden models, reproduction scripts, and frozen byte accounting will be submitted for artifact evaluation.

B. Bottleneck Relief End to End

KORD relieves the bottleneck it targets: on a private ResNet-18 inference—6,726,608 FSS trees (1.71 M ReLU, 1.71 M truncation, 1.61 M MaxPool, 512 division operators; truncation and division take two trees each), with the online phase matching cleartext bit for bit on 1000/1000 images—key generation falls from 82.6% of the total time under the trusted dealer LLAMA stack to 10.1% under KORD (Table II), and the online phase—identical in every deployment because KORD emits standard keys—again dominates at 89.9%, restoring the cost profile assumed by a decade of online optimizations. An online phase itself accelerated 10× would return key generation to ≈53%, sharpening the case for a fast key source. The dealer ships 8.87 GB over two

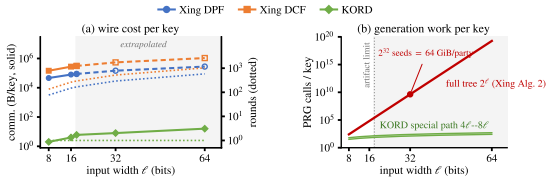


Fig. 7: Scaling with input width for the baseline's raw DPF/DCF primitives (§VII-A; Table III). (a) Wire cost and rounds grow linearly in ℓ; KORD sends $2\lceil \ell/8 \rceil$ B ($\ell_{\text{out}}{=}\ell$) in one round (open markers extrapolate past $\ell{=}18$). (b) Generation work: the baseline's full 2^{ℓ}-leaf tree vs. KORD's ℓ-level special path.

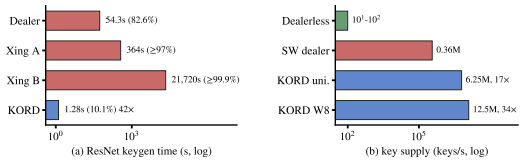


Fig. 8: Results at system level (log axes; Table II, §VII-E). (a) ResNet-18 key-generation time across deployments ($\geq$ marks extrapolated caliber A/B rows). (b) Sustained key supply rate: software baselines vs. KORD tiers ($\ell{=}32$ rate law points).

parallel links (4.469 and 4.403 GB per link), so its transfer time is governed by the larger payload. KORD's total is a synthesis-estimated serialized sum of generation and transfer without overlap credit, an upper bound within a model that excludes key delivery I/O at the host; the dealer's compute leg is measured (medians of 30 runs), its transfer modeled from bandwidth. The 0.880 s compute row is 175,990,696 fabric cycles at 200 MHz over the frozen tree inventory (mixed depths, 26.2 levels/tree average; per layer inventory in the supplement); at the 154.6 MHz signoff clock it is 1.139 s and the share 11.9% (the 200 MHz point is validated on the board, §VII-E). A dealer resident on the GPU [18], [21] would compress only the 18.54 s compute row: delivery, set by key size, stays $\geq$ 35.75 s even with compute free (27.9× KORD's total, third party trust intact), and shipping ahead cannot help since each inference consumes a fresh 4.469 GB. Under the KORD-mac extension (+36 B per message; Fig. 11) the wire total is 292.6 MB and the share 22.0%—still well below 82.6%.

Against the dealerless baseline the relief is larger still. Charged *in its favor* (its $\Theta(2^{\ell})$ generation and Beaver triples counted as zero), key generation still consumes $\geq$ 96.9%–99.9% of the total time (Fig. 8(a))—a 283×–16,920× reduction, decomposed by mechanism in Fig. 10 (top) and essentially flat across the LAN, the 40 Mbps WAN, and the DORAM $d{=}16$ anchor.

C. Protocol and Algorithmic Scaling

At the operation level, every generation cost axis collapses at once (Fig. 9): communication by four to nearly five orders of magnitude, latency to near one RTT regardless of protocol,

TABLE III: The baseline's own key-generation measurements [50] (its Table II) with KORD added, generation axis only, under the baseline's links (calibers: §VII-A).

		Xing gen. (s)		Gen. comm.		KORD reduction (×)		
Protocol	ℓ	LAN	WAN	Xing (MB)	KORD (B)	LAN	WAN	Comm
Π_{DPF}	8	0.050	1.288	0.046	2	1,000	64	23,142
	16	0.092	2.450	0.080	4	1,840	123	19,907
	18	0.124	2.780	0.088	6	2,480	139	14,661
Π_{DCF}	8	0.085	2.417	0.145	2	1,700	121	72,500
	16	0.180	4.625	0.277	4	3,600	231	69,250
	18	0.280	5.247	0.310	6	5,600	262	51,667

and DORAM and MP-SPDZ preprocessing by similar factors. Interaction falls from 154–38,659 messages one way to a single round, and the fabric's $\leq$5.60 µs expansion (a synthesis-estimated model value over all nine building blocks, not a speedup claim; the thirteen operations plotted in Fig. 9 stay $\leq$2.2 µs) keeps pace with the baseline's measured 37.8 ms–7.14 s per operation (LAN; the endpoints are EQ at $\ell{=}8$ and PriLUT, one of the two blocks beyond the figure). Online evaluation is excluded by construction—KORD emits standard keys, so the `Eval` phase is identical to the bit—and the trigonometric kernels additionally eliminate the Millionaire, B2A, and Beaver triple instances the baseline generates interactively (the full suite per block, calibers, KORD-mac message authentication, and the 15–40% measured versus reported discrepancies on ten of eighteen points are in the supplement).

Scaling with input width separates the routes (Fig. 7, Table III). The baseline's communication and rounds per key are strictly linear in ℓ, so the gap grows from 72,500× at the measured DCF $\ell{=}8$ point to 1,132×–67,611× at $\ell{=}32$, while KORD's message stays flat at one round. Generation *work* scales worse: full tree expansion (Fig. 7(b)) makes the artifact fail above $\ell{=}18$, so wider latency is charged as a communication lower bound, while KORD's walk stays linear in ℓ.

D. Ablation: Mechanisms and Cross-Key Scheduling

KORD's three mechanisms remove the bottleneck in sequence (Fig. 10, top): special path generation turns the dealerless $\Theta(2^{\ell})$ full tree expansion into a $\Theta(\ell)$ walk, the single round removes the communication wall, and cross-key interleaving fills the walk's serial bubbles. All four bars are dealerless with no online principal, so this 283× LAN reduction holds the trust model fixed and attributes the gain to protocol–hardware co-design alone, not to any change in what must be trusted; the leftmost bar charges the dealerless software's compute at zero (a communication lower bound)—at its measured 10^1–10^2 keys/s (§VII-A) it would sit at $\geq 10^4$ s—so the software reference KORD improves on is itself credited favorably.

Cross-key interleaving—not lane count, not clock—is what saturates the fabric: sweeping the number of keys in flight M on otherwise identical hardware yields a 12.01× throughput gain on the same lanes and the same clock (Fig. 10, bottom). With one key in flight the array idles at $1/L_{\text{eff}}$—the remaining slots are dependency bubbles no lane count or clock can fill—

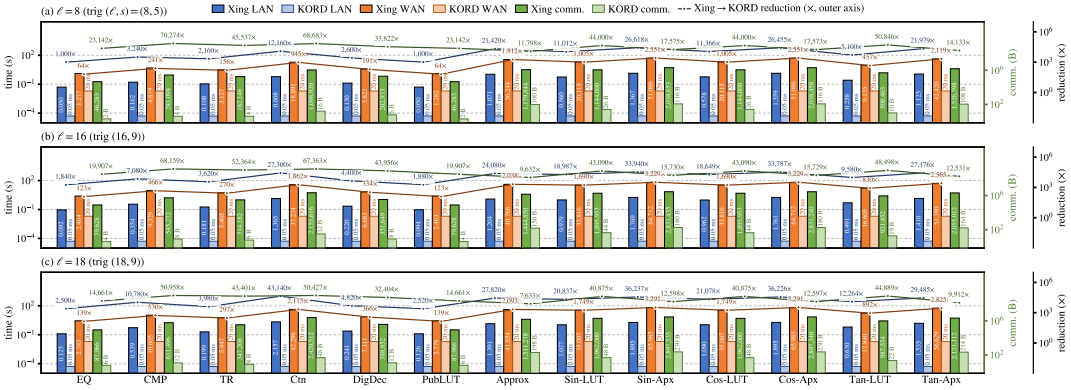


Fig. 9: Generation cost per operation, Xing [50] vs. KORD: seven of the nine building blocks and six trigonometric kernels at $\ell \in \{8, 16, 18\}$ (log scale; links and one RTT accounting of §VII-A). Solid bars: Xing latency (LAN/WAN) and communication; tinted bars: KORD—one RTT + $\leq$2.2 µs expansion + 2–230 B. Polylines give the exact Xing→KORD reduction per operation (7,633×–70,274× comm.).

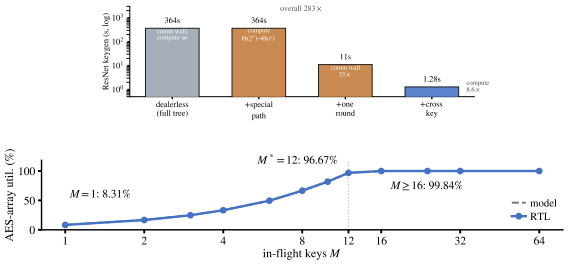


Fig. 10: Ablation. Top: ResNet-18 key-generation time as each mechanism is added—the special path makes compute finite ($\Theta(2^\ell) \to \Theta(\ell)$), the single exchange round collapses the communication wall (33×), and cross-key interleaving the compute wall (8.6×), 283× overall. Bottom: AES array utilization vs. keys in flight M—the RTL sweep rises from 8.31% (M=1) to 99.84% (12.01×) past the knee M^*=12; the schedule model (crosses) tracks it within 0.3%, and the board saturates at 99.98% (star). DPF W=4 and unified W=8 coincide.

and independent keys convert exactly those bubbles into work until saturation at the knee of Eq. (6). Both swept engines trace one curve, and Eq. (7) tracks the RTL closely enough to provision the fabric. Because the gain is bounded by L_{eff} by construction, we report it as model validation, not as a headline speedup: one interleaved array with 0.82 KiB of slot state replaces the 12 single chain engines a purely data parallel design would need for the same rate. Silicon confirms the saturated regime (§VII-E).

E. Hardware Cost and Validation

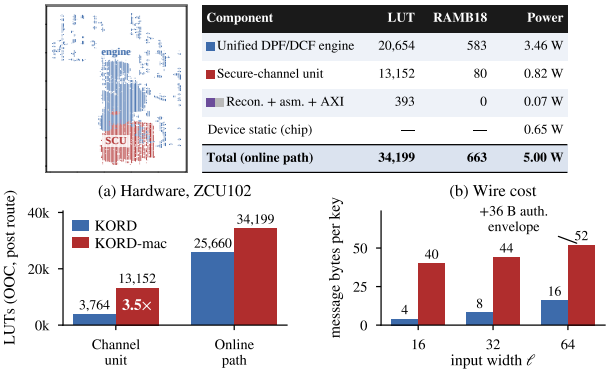
The complete online path fits in a corner of a midrange FPGA (Fig. 11, top: placed floorplan, resource and power bill of the default tier's board build); the generation engine accounts for 83.0% of the path's LUTs, and the secure channel unit—the hardware cost of the trust boundary—for 15.5% of LUTs and 11.9% of BRAMs. The OOC ledgers (Table IV, Fig. 11) keep hierarchy for like against like comparison; the routed board build optimizes across module boundaries and

TABLE IV: Synthesized key-generation engines on ZCU102 (XCZU9EG-2, OOC after routing), all at a common 200 MHz. †W16 is a capacity boundary (S-boxes fold into LUTs), not a point of linear scaling; Thpt@200 is the ideal ℓ=32 rate law point; keys/J counts 32-bit keys at vectorless power. ‡DPF W8 board-measured: 99.02% utilization and 11.60 M keys/s at 187.5 MHz (§VII-E).

Engine	$f_{\max}$ (MHz)	LUT	RAMB18	Thpt@200 (M/s)	keys/J (M)
DPF W4	234.1	12,906	305	6.25	2.99
DPF W8‡	204.0	21,540	601	12.50	2.75
DPF W16†	207.8	72,092	912	25.00	2.70
Unified W8	154.6	20,654	583	6.25	2.36

trims the online path from 25,660 to 21,553 LUTs, so the two accountings differ by construction, not by error. Hardening to KORD-mac for actively tampered links is also implemented and board validated, at a measured rather than bounded price (Fig. 11, bottom): the channel unit grows 3.5× in LUTs and each message gains a 36 B envelope; the engine behind the SCU is unchanged. Unification is nearly free: one datapath serving DPF and DCF adds 0.96% LUT over a design for DCF alone and saves 50.8%/50.3% of LUT/BRAM relative to provisioning separate engines.

The engines (Table IV) span the design points, all operated at a common 200 MHz—within every DPF engine's synthesis $f_{\max}$ and, for the unified engine, validated by the board measurements below. The engine for DPF alone, with 4-lane issue groups, retires two tree levels per cycle, twice the unified engine's one level per cycle in either mode, so the 2× tier gap is architectural, not a frequency artifact (a DPF level there uses four of the eight lanes), and a mixed stream satisfies $R_{\text{DPF}}+R_{\text{DCF}} \leq$ the table's endpoint per mode. One unified engine supplies the key inventory of 1.14 ResNet-18 inferences per second ($\approx$13 concurrent online sessions). These are rates at the fabric's assembler boundary: at the 540 B DPF-32 key share (§II-A) the headline tier is a $\approx$6.75 GB/s egress stream per chip, and sustaining it off the chip requires a DMA class interface we have not built—though the rate sits within a single



Component	LUT	RAMB18	Power
Unified DPF/DCF engine	20,654	583	3.46 W
Secure-channel unit	13,152	80	0.82 W
Recon. + asm. + AXI	393	0	0.07 W
Device static (chip)	—	—	0.65 W
Total (online path)	**34,199**	**663**	**5.00 W**

Fig. 11: Top: placed layout of the KORD online path on ZCU102 (routed board build)—each dot is one of 40,301 leaf cells at its SLICE location, colored by component; the eight AES-MMO lanes place as the tall vertical bands; LUT and RAMB18 cells give the routed hierarchy; power is vectorless (per component dynamic plus one chip static term). Bottom: the KORD-mac extension vs. the default tier it hardens. (a) Channel unit 3,764 → 13,152 LUTs ($f_{\max}$ 308.9 → 269.0 MHz), online path 25,660 → 34,199 (post-route OOC); on the board the tiers run exact to the bit through 214.3 and 250.0 MHz, above the 200 MHz operating point (each build's highest passing PLL step under its own timing closure; the mac build was constrained at 250 MHz). (b) A 36 B authentication envelope per message ($4/8/16 \rightarrow 40/44/52$ B at ℓ=16/32/64); the ResNet-18 wire grows from 50.4 to 292.6 MB, the keygen share from 10.1% to 22.0%.

PCIe Gen4 ×8 link (∼15.8 GB/s raw), an integration task outside this prototype, not a capability gap.

Validation runs from simulation to silicon. In `xsim` the DPF and unified engines match the software oracle bit for bit (108,276 and 226,920 checks), and the full online path passes 4,540 vectors per channel tier (KORD-mac: 60/60 authentications verified). On a physical ZCU102 over the processing system's debug access port, four results hold. *(i) Correctness:* 60 keys (30 DPF, 30 DCF; $\ell \in \{16, 32, 64\}$) reproduce the golden correction word stream across 13,560 field checks (six 32-bit fields per level over the keys' $\Sigma\ell$=2,240 levels, plus two final fields per key) with zero mismatches. *(ii) Channel misuse:* tampered ciphertext, a wrong channel key, or a diverged counter yields only an unusable key (its correction words diverge from golden) and a malformed configuration is rejected—misuse degrades keys, leaks nothing (§VI); KORD-mac instead rejects all four attack classes of that section with no engine output. Both checks exercise the pairing itself: the chip holds the pairing's two directional subkeys and lockstep counters while the reference model plays the peer chip under the same secrets, so an accepted key means both directions of the paired channel agree. *(iii) Frequency:* the online path runs exact to the bit through 214.3 MHz against a 144.1 MHz worst case OOC extrapolation ($\geq 1.48\times$), so the synthesis estimates are conservative on the measured device; derived rates assume the board validated operating point (one device, nominal conditions). *(iv) Saturation:* a request generator holding all M=16 slots full sustains 99.98% lane utilization, matching the rate law within 0.1% at every width and clock swept. It holds on the unified engine (DPF stream at 49.99%, half a block, as the endpoint per mode assumes) and on a W=4 DPF instance. We then ran the headline W=8 DPF tier on the board: at saturating provisioning M=28 (M^*=24 plus headroom) it holds 99.02% utilization and 11.60 M 32-bit keys/s at 187.5 MHz (the highest passing integer PLL divider; 214.29 MHz exceeds the full block design $f_{\max}$), while a separate build provisioned at M=16 returns 66.5%, matching the predicted 16/24 within the −0.3% admission overhead seen in simulation and confirming the knee $M^*=WL_{\text{eff}}/c$ on the board. The flagship saturation is therefore board-measured; this tier itself never ran at 200 MHz (no PLL step lands there), so at that operating point—validated by a separate W=8 build through 214.3 MHz—the rate law gives 12.50 M ideal, 12.38 M at the board-measured utilization. Silicon thereby confirms the saturation curve of Fig. 10 (bottom). As the rate law scales with clock and lanes, a conservative 1 GHz ASIC signoff (5× the board) would project the 0.880 s fabric row (Table II) to ≈0.18 s and 5× the supply; we tape out no ASIC, and the total time, bound by the link, is unchanged.

In key supply terms the compute wall stays down at system scale (Fig. 8(b)): the fabric outpaces the software dealer by up to 34.5× and dealerless software Gen by orders of magnitude. For DORAM at d=20 (the supplement's extension of the d=16 anchor) it generates 24× more supply than a 40 Mbps link can drain: the link, not key generation, again becomes the binding constraint.

VIII. Related Work

FSS systems and their key sources. FSS systems [18], [19], [21], [37] shift the online cost other 2PC pays per nonlinear layer [22], [30], [32], [35] into one-time keys from a dealer or helper [18], [19], [21], [37], [41], [47] or a client that knows the point [1], [9], [10], [48]—neither fits an MPC pipeline, where the point is a secret shared mask [14], [44]. KORD serves this setting, exposing masks and key shares to no principal (§VI).

Dealerless key generation. Our baseline, Xing et al. [50], follows Floram [13]; we do not treat its full tree Gen as a lower bound. Half-Tree [17] halves expansion but lacks arithmetic sharing. Noninteractive [**?**] and multiparty [**?**] DPFs reshape the tradeoff but do not target application rate supply; programmable DPFs [8], correlation generators [3], [4], [51], and generic 2PC [23] stay costlier at inference scale (Section VII-C); GPU batching [18], [21], [27], [34], [42], [49] accelerates a route without changing its trust. Cross-key interleaving is the fine multithreading of a barrel processor; our contribution is the closed form provisioning law (Eqs. (5)–(7)) tied to GGM's serial depth, validated to silicon.

Cryptographic hardware and trusted roots. FHE ASICs [24]–[26], [38], [39], HE co-design [36], garbled circuit [31], [40] and OT [29] accelerators target data parallel kernels, not the trust model. PPMLAC [52] roots secret sharing ML in attested hardware; KORD adopts only its

initialization. A TEE hosted dealer [11], [28], [43], [46] runs loaded code with its attack surface [45] and remains a principal; KORD's chips load none. Two paired TEEs would reach the same trust destination (Section VI) at a software key rate; the ablation of §VII-D isolates the co-design gain within the class, fixing the trust model and varying only the datapath.

IX. Conclusion

KORD pairs two fixed function chips once, generates keys in one round and a few bytes, and saturates one AES array by interleaving keys: where the key stream is the bottleneck, KORD is dealerless under a minimal hardware root.

References

[1] D. Boneh, E. Boyle, H. Corrigan-Gibbs, N. Gilboa, and Y. Ishai, "Lightweight techniques for private heavy hitters," in *IEEE Symposium on Security and Privacy (S&P)*. IEEE, 2021.

[2] E. Boyle, N. Chandran, N. Gilboa, D. Gupta, Y. Ishai, N. Kumar, and M. Rathee, "Function secret sharing for mixed-mode and fixed-point secure computation," in *Advances in Cryptology – EUROCRYPT 2021*. Springer, 2021, pp. 871–900.

[3] E. Boyle, G. Couteau, N. Gilboa, Y. Ishai, L. Kohl, P. Rindal, and P. Scholl, "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)*. ACM, 2019, pp. 291–308.

[4] E. Boyle, G. Couteau, N. Gilboa, Y. Ishai, L. Kohl, and P. Scholl, "Efficient pseudorandom correlation generators: Silent OT extension and more," in *Advances in Cryptology – CRYPTO 2019*. Springer, 2019.

[5] E. Boyle, N. Gilboa, and Y. Ishai, "Function secret sharing," in *Advances in Cryptology – EUROCRYPT 2015*. Springer, 2015.

[6] E. Boyle, N. Gilboa, and Y. Ishai, "Function secret sharing: Improvements and extensions," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS)*. ACM, 2016, pp. 1292–1303.

[7] E. Boyle, N. Gilboa, and Y. Ishai, "Secure computation with preprocessing via function secret sharing," in *Theory of Cryptography Conference (TCC)*. Springer, 2019.

[8] E. Boyle, N. Gilboa, Y. Ishai, and V. I. Kolobov, "Programmable distributed point functions," in *Advances in Cryptology – CRYPTO 2022*. Springer, 2022, pp. 121–151.

[9] B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan, "Private information retrieval," in *36th Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE, 1995, pp. 41–50.

[10] H. Corrigan-Gibbs and D. Boneh, "Prio: Private, robust, and scalable computation of aggregate statistics," in *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*. USENIX Association, 2017, pp. 259–282.

[11] V. Costan and S. Devadas, "Intel SGX explained," Cryptology ePrint Archive, Paper 2016/086, 2016.

[12] D. Demmler, T. Schneider, and M. Zohner, "ABY – a framework for efficient mixed-protocol secure two-party computation," in *Network and Distributed System Security Symposium (NDSS)*, 2015.

[13] J. Doerner and a. shelat, "Scaling ORAM for secure computation," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS)*. ACM, 2017, pp. 523–535.

[14] B. H. Falk, R. Ostrovsky, M. Shtepel, and J. Zhang, "GigaDORAM: Breaking the billion address barrier," in *32nd USENIX Security Symposium (USENIX Security)*. USENIX Association, 2023, pp. 3871–3888.

[15] N. Gilboa and Y. Ishai, "Distributed point functions and their applications," in *Advances in Cryptology – EUROCRYPT 2014*. Springer, 2014.

[16] O. Goldreich, S. Goldwasser, and S. Micali, "How to construct random functions," *Journal of the ACM*, vol. 33, no. 4, pp. 792–807, 1986.

[17] X. Guo, K. Yang, X. Wang, W. Zhang, X. Xie, J. Zhang, and Z. Liu, "Half-tree: Halving the cost of tree expansion in COT and DPF," in *Advances in Cryptology – EUROCRYPT 2023*, ser. Lecture Notes in Computer Science, C. Hazay and M. Stam, Eds., vol. 14004. Springer, 2023, pp. 330–362.

[18] K. Gupta, N. Jawalkar, A. Mukherjee, N. Chandran, D. Gupta, A. Panwar, and R. Sharma, "SIGMA: Secure GPT inference with function secret sharing," *Proceedings on Privacy Enhancing Technologies (PoPETs)*, vol. 2024, no. 4, pp. 61–79, 2024.

[19] K. Gupta, D. Kumaraswamy, N. Chandran, and D. Gupta, "LLAMA: A low latency math library for secure inference," *Proceedings on Privacy Enhancing Technologies (PoPETs)*, vol. 2022, no. 4, pp. 274–294, 2022.

[20] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2016, pp. 770–778.

[21] N. Jawalkar, K. Gupta, A. Basu, N. Chandran, D. Gupta, and R. Sharma, "Orca: FSS-based secure training and inference with GPUs," in *IEEE Symposium on Security and Privacy (S&P)*. IEEE, 2024.

[22] C. Juvekar, V. Vaikuntanathan, and A. Chandrakasan, "GAZELLE: A low latency framework for secure neural network inference," in *27th USENIX Security Symposium (USENIX Security)*. USENIX Association, 2018, pp. 1651–1669.

[23] M. Keller, "MP-SPDZ: A versatile framework for multi-party computation," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security (CCS)*. ACM, 2020, pp. 1575–1590.

[24] J. Kim, S. Kim, J. Choi, J. Park, D. Kim, and J. H. Ahn, "SHARP: A short-word hierarchical accelerator for robust and practical fully homomorphic encryption," in *Proceedings of the 50th Annual International Symposium on Computer Architecture (ISCA)*. ACM, 2023.

[25] J. Kim, G. Lee, S. Kim, G. Sohn, M. Rhu, J. Kim, and J. H. Ahn, "ARK: Fully homomorphic encryption accelerator with runtime data generation and inter-operation key reuse," in *Proceedings of the 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2022, pp. 1237–1254.

[26] S. Kim, J. Kim, M. J. Kim, W. Jung, J. Kim, M. Rhu, and J. H. Ahn, "BTS: An accelerator for bootstrappable fully homomorphic encryption," in *Proceedings of the 49th Annual International Symposium on Computer Architecture (ISCA)*. ACM, 2022, pp. 711–725.

[27] B. Knott, S. Venkataraman, A. Hannun, S. Sengupta, M. Ibrahim, and L. van der Maaten, "CrypTen: Secure multi-party computation meets machine learning," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

[28] D. Lee, D. Kohlbrenner, S. Shinde, K. Asanović, and D. Song, "Keystone: An open framework for architecting trusted execution environments," in *Proceedings of the 15th European Conference on Computer Systems (EuroSys)*. ACM, 2020.

[29] C. Lin, K. Yang, T. Xu, L. Liang, Y. Wang, Z. Chen, R. Wang, M. Gao, and M. Li, "Ironman: Accelerating oblivious transfer extension for privacy-preserving AI with near-memory processing," in *Proceedings of the 58th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. ACM, 2025.

[30] P. Mishra, R. Lehmkuhl, A. Srinivasan, W. Zheng, and R. A. Popa, "Delphi: A cryptographic inference service for neural networks," in *29th USENIX Security Symposium (USENIX Security)*. USENIX Association, 2020.

[31] J. Mo, J. Gopinath, and B. Reagen, "HAAC: A hardware-software co-design to accelerate garbled circuits," in *Proceedings of the 50th Annual International Symposium on Computer Architecture (ISCA)*. ACM, 2023.

[32] P. Mohassel and Y. Zhang, "SecureML: A system for scalable privacy-preserving machine learning," in *IEEE Symposium on Security and Privacy (S&P)*. IEEE, 2017, pp. 19–38.

[33] National Institute of Standards and Technology, "Advanced encryption standard (AES)," U.S. Department of Commerce, Tech. Rep. FIPS PUB 197, 2001.

[34] L. K. L. Ng and S. S. M. Chow, "GForce: GPU-friendly oblivious and rapid neural network inference," in *30th USENIX Security Symposium (USENIX Security)*. USENIX Association, 2021, pp. 2147–2164.

[35] D. Rathee, M. Rathee, N. Kumar, N. Chandran, D. Gupta, A. Rastogi, and R. Sharma, "CrypTFlow2: Practical 2-party secure inference," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security (CCS)*. ACM, 2020, pp. 325–342.

[36] B. Reagen, W.-S. Choi, Y. Ko, V. T. Lee, H.-H. S. Lee, G.-Y. Wei, and D. Brooks, "Cheetah: Optimizing and accelerating homomorphic encryption for private inference," in *IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2021, pp. 26–39.

[37] T. Ryffel, P. Tholoniat, D. Pointcheval, and F. Bach, "AriaNN: Low-interaction privacy-preserving deep learning via function secret sharing,"

Proceedings on Privacy Enhancing Technologies (PoPETs), vol. 2022, no. 1, pp. 291–316, 2022.

[38] N. Samardzic, A. Feldmann, A. Krastev, S. Devadas, R. Dreslinski, C. Peikert, and D. Sanchez, "F1: A fast and programmable accelerator for fully homomorphic encryption," in *Proceedings of the 54th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. ACM, 2021, pp. 238–252.

[39] N. Samardzic, A. Feldmann, A. Krastev, N. Manohar, N. Genise, S. Devadas, K. Eldefrawy, C. Peikert, and D. Sanchez, "CraterLake: A hardware accelerator for efficient unbounded computation on encrypted data," in *Proceedings of the 49th Annual International Symposium on Computer Architecture (ISCA)*. ACM, 2022, pp. 173–187.

[40] E. M. Songhori, S. U. Hussain, A.-R. Sadeghi, T. Schneider, and F. Koushanfar, "TinyGarble: Highly compressed and scalable sequential garbled circuits," in *IEEE Symposium on Security and Privacy (S&P)*. IEEE, 2015, pp. 411–428.

[41] K. Storrier, A. Vadapalli, A. Lyons, and R. Henry, "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)*. ACM, 2023, pp. 2143–2157.

[42] S. Tan, B. Knott, Y. Tian, and D. J. Wu, "CryptGPU: Fast privacy-preserving machine learning on the GPU," in *IEEE Symposium on Security and Privacy (S&P)*. IEEE, 2021, pp. 1021–1038.

[43] F. Tramèr and D. Boneh, "Slalom: Fast, verifiable and private execution of neural networks in trusted hardware," in *International Conference on Learning Representations (ICLR)*, 2019.

[44] A. Vadapalli, R. Henry, and I. Goldberg, "Duoram: A bandwidth-efficient distributed ORAM for 2- and 3-party computation," in *32nd USENIX Security Symposium (USENIX Security)*. USENIX Association, 2023.

[45] J. Van Bulck, M. Minkin, O. Weisse, D. Genkin, B. Kasikci, F. Piessens, M. Silberstein, T. F. Wenisch, Y. Yarom, and R. Strackx, "Foreshadow: Extracting the keys to the Intel SGX kingdom with transient out-of-order execution," in *27th USENIX Security Symposium (USENIX Security)*. USENIX Association, 2018, pp. 991–1008.

[46] S. Volos, K. Vaswani, and R. Bruno, "Graviton: Trusted execution environments on GPUs," in *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. USENIX Association, 2018, pp. 681–696.

[47] S. Wagh, "Pika: Secure computation using function secret sharing over rings," *Proceedings on Privacy Enhancing Technologies (PoPETs)*, vol. 2022, no. 4, pp. 351–377, 2022.

[48] F. Wang, C. Yun, S. Goldwasser, V. Vaikuntanathan, and M. Zaharia, "Splinter: Practical private queries on public data," in *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*. USENIX Association, 2017.

[49] J.-L. Watson, S. Wagh, and R. A. Popa, "Piranha: A GPU platform for secure computation," in *31st USENIX Security Symposium (USENIX Security)*. USENIX Association, 2022, pp. 827–844.

[50] P. Xing, H. Li, M. Hao, H. Chen, J. Hu, and D. Liu, "Distributed function secret sharing and applications," in *Network and Distributed System Security Symposium (NDSS)*, 2025.

[51] K. Yang, C. Weng, X. Lan, J. Zhang, and X. Wang, "Ferret: Fast extension for correlated OT with small communication," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security (CCS)*. ACM, 2020, pp. 1607–1626.

[52] X. Zhou, Z. Xu, C. Wang, and M. Gao, "PPMLAC: High performance chipset architecture for secure multi-party computation," in *Proceedings of the 49th Annual International Symposium on Computer Architecture (ISCA)*. ACM, 2022, pp. 87–101.