Encifher: A Trusted-Execution Coprocessor for Confidential Computation on Solana

Nitanshu Lokhande
RizeLabs
nitanshu@rizelabs.io

Rishabh Gupta
RizeLabs
rishabh@rizelabs.io

Rachit Chahar
RizeLabs
rachit@rizelabs.io

Arun Jangra
RizeLabs
arun@rizelabs.io

Aniket Prajapati
RizeLabs
aniket@rizelabs.io

Muskan Pandey
RizeLabs
muskan@rizelabs.io

Abstract

Public blockchains expose all state and computation by default, which is incompatible with financial applications that require confidentiality. Solana achieves high throughput and sub-second confirmation, making it an attractive settlement layer, yet it offers no general mechanism for computing over encrypted state: fully homomorphic encryption (FHE) remains orders of magnitude too slow for interactive use, secure multi-party computation (MPC) incurs heavy communication, and Solana's native confidential-transfer extension hides only token amounts and supports no programmable logic. We present Encifher, a confidentiality *coprocessor* for Solana that brings general, programmable computation over encrypted state to a high-throughput chain. Encifher adopts the ciphertext-handle and symbolic-execution interface of confidential-computing coprocessors (e.g., Zama's fhEVM), but resolves it inside a Trusted Execution Environment (TEE) rather than with a fully homomorphic evaluator: on-chain Solana programs manipulate only 128-bit handles to ciphertexts, while an off-chain coprocessor running inside a hardware enclave fetches the corresponding ciphertexts from a public data availability layer, decrypts and computes on plaintext within the enclave, re-encrypts, and commits attested, Merkle-anchored results back on chain. A key observation is that Solana's account model and parallel (Sealevel) scheduler turn the transaction's declared read/write set into the dependency graph of the confidential computation, yielding parallel, correctly-ordered execution of encrypted operations without a bespoke scheduler. To avoid the single-point-of-failure of an enclave holding a master decryption key, Encifher distributes trust across a threshold-decryption committee and verifies enclave attestation on chain. Encifher is deployed in production, powering confidential payments, swaps, and a cross-chain bridge. In production, a confidential swap settles in a single Solana transaction costing 157,000 compute units; the AES-256-GCM symmetric-encryption layer runs at over a million operations per second and threshold decryption completes in single-digit to tens of milliseconds (near-plaintext speed, against the many-orders-of-magnitude overhead of FHE), and Encifher has served over 5,000 users and 50,000 confidential operations. We are explicit about the cost of this design point: Encifher reduces the trust base to TEE integrity, honest threshold key management, and the cloud attestation root, rather than to cryptographic hardness alone.

1 Introduction

Public blockchains replicate every transaction and every byte of state across all participants. This radical transparency is what makes them verifiable, but it is fundamentally at odds with the confidentiality that financial activity requires: a payment whose amount, counterparty, and purpose are visible to the world; a trade that is front-run because the order book is public; a lending position that is liquidated because its collateralization is legible to all. The tension between public verifiability and private state is the central obstacle to bringing mainstream financial workloads on chain.

Solana is, in many respects, an ideal settlement substrate for such workloads. Its pipelined consensus, a verifiable-delay-style hash chain (Proof of History) layered with a stake-weighted voting protocol (Tower BFT) [1], together with a parallel execution runtime (Sealevel) [2] delivers sub-second confirmation and throughput far beyond that of single-threaded chains. Yet Solana provides no general mechanism for computing over encrypted state. Every account's data is stored in cleartext and is world-readable [3]; the only native confidentiality primitive, the Token-2022 confidential-transfer extension [4], hides token *amounts* but not senders, receivers, or the transaction graph, supports no programmable logic, and, at the time of writing, is disabled on mainnet following a soundness vulnerability in its zero-knowledge proof program [5].

Why existing approaches do not close the gap. The cryptographic toolkit for confidential computation on blockchains is well developed, but each established approach is a poor fit for general-purpose, interactive confidential applications on a high-throughput chain like Solana. *Fully homomorphic encryption (FHE)* computes directly on ciphertext and offers the strongest, purely cryptographic confidentiality, and recent FHE coprocessors [6] make it programmable. But even with the fastest bootstrapping schemes [7], homomorphic operations remain roughly nine orders of magnitude slower than their plaintext counterparts, limiting deployed FHE coprocessors to on the order of tens of transactions per second [6]. *Secure multi-party computation (MPC)* removes any single trusted party by secret-sharing state across nodes, and is the basis of Solana's most prominent confidential-computing project [8]; but its communication cost grows with circuit complexity and the number of parties, and interactive workloads pay a per-operation round-trip tax. *Zero-knowledge (ZK) proofs* provide excellent transaction privacy, hiding a user's *own* inputs over a commitment set, as in shielded-payment systems, but do not natively support computation over state that is secret to *all* parties simultaneously, which is precisely what a shared confidential ledger or a sealed order book requires. The practical consequence is a missing layer: Solana lacks a deployed, general, performant way to run application logic over encrypted, shared state.

Our approach. We present Encifher, a confidentiality *coprocessor* for Solana. Encifher follows the now-standard coprocessor pattern (§2.3), execute the expensive work off chain, commit a cheaply verifiable result on chain, but specializes it for *confidentiality* rather than scalability. From FHE coprocessors we borrow the interface that has proven ergonomic for developers: encrypted values are represented on chain by 128-bit *handles* (pointers to ciphertexts), and on-chain programs perform *symbolic execution* [6], recording operations over handles and deferring the real computation. Where an FHE coprocessor would resolve those operations homomorphically, Encifher resolves them inside a *Trusted Execution Environment* (TEE): an off-chain coprocessor running in a hardware enclave fetches the ciphertexts referenced by the handles from a public data-availability (DA) layer, decrypts and computes on plaintext *within the enclave*, re-encrypts the result, and commits an attested, Merkle-anchored digest back on chain. Plaintext never leaves the enclave. This is, in effect, the architecture of an FHE coprocessor with the FHE evaluator replaced by a TEE (Figure 1), a

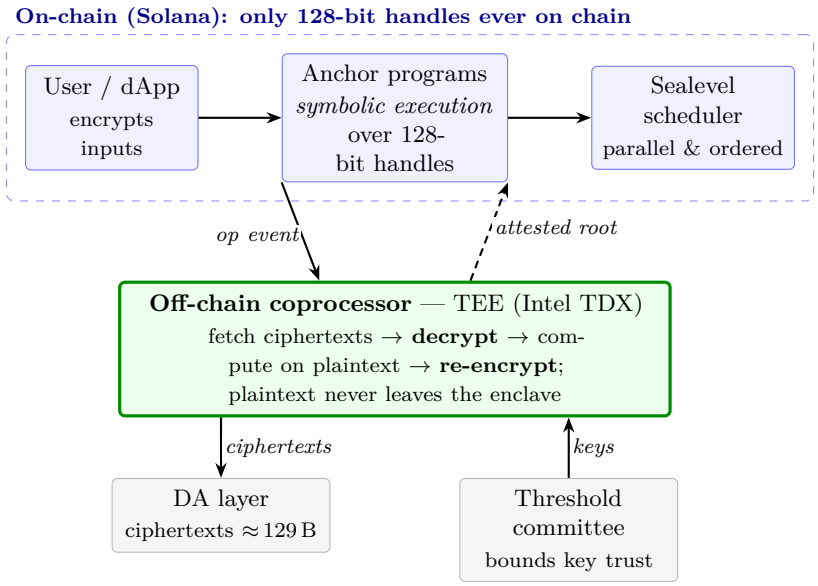

Figure 1: High-level design. On-chain Solana programs manipulate only 128-bit handles via symbolic execution; an off-chain TEE coprocessor resolves the operations on plaintext inside the enclave and commits an attested, Merkle-anchored result. Ciphertexts live in a public DA layer and decryption authority is split across a threshold committee. This is the architecture of an FHE coprocessor with the FHE evaluator replaced by a TEE.

deliberate move along the trust-versus-performance frontier, trading FHE's purely cryptographic confidentiality for hardware-rooted confidentiality in exchange for near-native performance and support for arbitrary computation.

The Solana substrate is not incidental. Porting the handle/symbolic-execution model from the EVM, where every prior confidentiality and FHE coprocessor has been built, to Solana is not a mechanical exercise, and the differences are where much of our design contribution lies. Three of Solana's defining properties jointly force ciphertexts *off* chain and only 128-bit handles *on* chain: a tight per-transaction compute budget that makes heavy on-chain cryptography infeasible, a 1,232-byte transaction-size limit that cannot carry a ciphertext but easily carries many handles, and a public-by-default, rent-priced account model under which on-chain ciphertext storage is expensive and yields no privacy. More interestingly, Solana's execution model turns a liability into an asset. Because every Solana transaction must *declare its read and write set of accounts in advance*, and because the Sealevel scheduler runs transactions with disjoint write sets in parallel while serializing conflicting ones, storing each handle in its own program-derived account makes the transaction's declared account set *equal to* the dependency graph of the confidential computation. Producer/consumer relationships between encrypted operations are then ordered correctly, and independent confidential operations are executed in parallel, *by the base-layer scheduler*, with no bespoke coordination logic, something the single-threaded EVM cannot offer.

Bounding the trust we place in hardware. A naive TEE confidentiality system concentrates catastrophic risk in a single enclave that holds a master decryption key: if that enclave is compromised, by a microarchitectural side channel, a firmware flaw, or an insider, every ciphertext it ever protected becomes retroactively decryptable. This is not hypothetical; it is exactly the failure mode

demonstrated against a deployed TEE blockchain whose network-wide master key was extracted from a single enclave [9, 10]. Encifher is designed to bound this blast radius. Decryption authority is split across a *threshold-decryption committee* of independently operated enclaves, so that confidentiality survives the compromise of any single enclave; and the on-chain programs verify enclave *attestation* before honoring coprocessor outputs, pinning computation to a specific, audited enclave image. We are explicit throughout about what Encifher trusts and what it does not.

Contributions. Encifher is, to our knowledge, the first deployed general-purpose confidential-computing coprocessor for Solana. Concretely:

- We design and **deploy** Encifher, a TEE-based confidentiality coprocessor that brings programmable computation over encrypted state to Solana, and use it in production for confidential payments, swaps, and a cross-chain bridge (§4, §8).
- We show how to map the ciphertext-handle / symbolic-execution coprocessor model onto Solana's account model and Sealevel scheduler, so that the transaction read/write set *is* the confidential-computation dependency graph, yielding parallel, correctly-ordered execution of encrypted operations from the base layer, and we design the supporting on-chain programs, per-handle program-derived accounts, and DA-backed verifiable batching (§4).
- We design a bounded-trust key-management and verification scheme, a threshold-decryption committee plus on-chain attestation verification, that avoids the single-enclave-master-key failure mode, and we give an honest security analysis situating our chosen TEE within the side-channel literature (§3, §7).
- We present the first end-to-end measurement of such a system in production (157,000-CU confidential swaps, sub-microsecond symmetric and millisecond threshold-decryption costs, and the batching and committee-scaling behavior), with a comparison to FHE and MPC alternatives and engineering lessons from operating it (§6, §10).

Roadmap. §2 reviews Solana, TEEs, and the coprocessor abstraction. §3 states the threat model. §4 presents the design and §5 the implementation. §6 evaluates Encifher, §7 analyzes its security, and §8 describes the deployed applications. §9 surveys related work and §10 discusses limitations and concludes.

2 Background

2.1 Solana, in the properties that matter for confidentiality

We summarize the Solana mechanisms that shape Encifher's design; [1–3] give full detail.

Accounts and programs. All persistent state on Solana lives in *accounts*, each identified by a 32-byte address and carrying a raw byte array (`data`), a lamport balance, and an `owner` program. Programs are *stateless*: a program is an account holding executable bytecode and stores no application state of its own; instead it creates and owns separate *data accounts*. Only an account's owner program may modify its `data`, but *any party may read any account*: state is public by default [3]. This inverts the Ethereum model, in which contract storage is encapsulated behind contract code; on Solana, hiding a value requires storing it *encrypted*, because the storage layer offers no read protection. Accounts pay a refundable, size-proportional *rent-exemption* deposit, and account data is capped at 10 MiB.

Program-derived addresses (PDAs). A PDA is an account address deterministically derived from a program ID and a set of seeds, placed off the signature curve so that it has no private key. PDAs give a program a deterministic namespace for its data accounts and let the program authorize writes to them programmatically. Encifher uses one PDA per ciphertext handle (§4).

Sealevel: declared read/write sets and parallel execution. Every Solana transaction declares, in advance, the complete list of accounts it will access and whether each is read-only or writable. The Sealevel runtime [2] exploits this to execute transactions with disjoint writable sets in parallel, taking a write lock on each writable account; transactions that write the same account are serialized. The consequence relevant to us is twofold. Negatively, a single "hot" account that many transactions must write becomes a serialization bottleneck, so a confidential design that funnels all encrypted state through one account would forfeit Solana's parallelism, and because the scheduler cannot interpret ciphertext it cannot shard such state automatically. Positively, the declared read/write set is a precise, statically known dependency specification, which Encifher turns into the dependency graph of its confidential computation (§4.9).

Compute and size limits. On-chain execution is metered in *compute units* (CU): by default 200,000 CU per instruction, capped at 1,400,000 CU per transaction, with a 12,000,000-CU ceiling per writable account per block [3]. Programs run in a restricted sandbox (sBPF) with small heap and stack and no hardware cryptographic acceleration. A transaction is limited to 1,232 bytes. Together these make heavy cryptography (FHE evaluation, large proof generation, multi-round MPC) infeasible on chain, and they make it impossible to carry a ciphertext inside a transaction; a 128-bit handle, by contrast, fits comfortably and many handles can be referenced per transaction.

Finality. Solana exposes three commitment levels: *processed* (a block has been received but may still be skipped), *confirmed* (a supermajority of stake has voted, reached in well under a second), and *finalized* (rooted after ∼32 subsequent slots, ∼12.8 s) [1]; the forthcoming Alpenglow upgrade, expected on mainnet in late 2026, is set to reduce finalization to ∼150 ms [11]. A confidentiality coprocessor that releases a decryption or an irreversible side effect must choose which level gates the release; we return to this in §4.

2.2 Trusted execution environments

A TEE is a hardware-isolated environment that protects the *confidentiality* and *integrity* of the code and data inside it from the host operating system, hypervisor, and operator, and that can produce a *remote attestation*: a hardware-signed statement of exactly which code is running, which a remote party can verify. TEEs fall into two structural families. *Process-level* TEEs (Intel SGX) isolate a region within a process and have the smallest trusted computing base, but execute on the adversary's core and share its microarchitectural state, the root cause of a long record of side-channel attacks [9, 12]. *VM-level* TEEs (Intel TDX, AMD SEV-SNP, AWS Nitro Enclaves) isolate a whole virtual machine; they have a larger trusted base but a coarser, more defensible boundary and a far smaller published microarchitectural attack surface.

Encifher's coprocessor runs in an *Intel TDX* Trust Domain, deployed and managed through the *Phala dStack* framework [13, 14]. TDX isolates an entire guest virtual machine, a *Trust Domain (TD)*, from the host hypervisor, the host operating system, and other tenants: a firmware component, the TDX Module, runs in a dedicated CPU mode (SEAM) and mediates every transition between the untrusted hypervisor and the TD, while the domain's memory is transparently encrypted and *integrity-protected* with a per-domain key (MKTME, AES-XTS-128). The trusted

base is thus the CPU, the Intel-signed TDX Module, and Intel as the attestation root; notably, the cloud operator and its host software are *outside* it. dStack runs the coprocessor as an ordinary container image inside the TD and adds a blockchain-governed key-management service that we use directly (§4.7) [14, 15].

Attestation in TDX is a two-step *TD Report* → *TD Quote* flow over Intel's Data Center Attestation Primitives (DCAP). The TDX Module produces a report binding a caller-chosen 64-byte value to the domain's measurements, *MRTD* (the build-time image) and *RTMR0–3* (runtime measurements of kernel, root filesystem, and application, extended with SHA-384), which a quoting enclave re-signs as a TD Quote with a certified ECDSA key on the *NIST P-256 (secp256r1)* curve, chaining to an Intel-rooted certificate [13, 16]. A relying party (here, a Solana program and the committee) checks the quote and compares MRTD/RTMR against the expected, on-chain-registered measurement of the audited coprocessor image, rejecting debug or mismatched domains. The signing curve matters for the on-chain design (§4.8): Solana gained a native secp256r1 precompile in 2025 [17], so a TD Quote's signature is verifiable on chain natively.

2.3 Blockchain coprocessors

A blockchain *coprocessor* lets an on-chain program delegate expensive or special-purpose computation to an off-chain system and receive a result together with a guarantee the chain can check cheaply: *trigger on chain* → *compute off chain* → *produce a guarantee* → *verify on chain*. The guarantee distinguishes the families. *ZK coprocessors* (e.g., for historical-state queries) prove the *correctness* of a computation over *public* data with a validity proof; they provide integrity and scalability, not confidentiality. *Confidentiality coprocessors* keep the inputs themselves secret, using FHE [6], TEEs [18], or MPC [8] as the off-chain engine.

This family shares a common programming interface, introduced by FHE coprocessors on the EVM [6]: on-chain programs never touch plaintext. Each encrypted value is named by a short, fixed-size *ciphertext handle*, and an on-chain operation over encrypted values records only the operation and its input/output handles rather than carrying it out. This deferred, reference-only record is a *symbolic* program: an off-chain engine later resolves each handle to its ciphertext, performs the actual computation, and binds the result back to its output handle. The on-chain handle graph is thus a symbolic-execution trace that the off-chain engine evaluates.

Encifher is a confidentiality coprocessor with a TEE engine. It inherits this handle-and-symbolic-execution interface, together with off-chain ciphertext storage and threshold decryption gated by on-chain permissions, but replaces the homomorphic evaluator with an enclave; from the TEE lineage of confidential off-chain execution, most directly Ekiden [18], which separates consensus from confidential execution and settles an attested signature on chain, it inherits the use of an enclave plus attestation as the confidentiality and integrity mechanism. §9 develops these connections; §4 presents the design.

2.4 Notation and cryptographic preliminaries

We write λ for the security parameter and call $\mathsf{negl}(\lambda)$ *negligible* if it is $o(\lambda^{-c})$ for every constant $c > 0$. For a finite set S, $x \xleftarrow{\$} S$ denotes sampling x uniformly at random from S (the superscript over the arrow marks the choice as randomized, standard cryptographic notation). H is a collision-resistant hash function (modeled as a random oracle where convenient), and $\|$ denotes unambiguous concatenation. "PPT" abbreviates probabilistic polynomial time.

Types, plaintexts, ciphertexts, handles. Let $\mathcal{T}$ be a finite set of *types* (e.g., `u64`, `bool`), each with a plaintext domain $\mathcal{P}_\tau$ (e.g., $\mathcal{P}_{\texttt{u64}} = \mathbb{Z}_{2^{64}}$); the plaintext space is $\mathcal{P} = \biguplus_{\tau \in \mathcal{T}} \mathcal{P}_\tau$ and $\mathcal{C}$ is the ciphertext space. A *handle* is an element of the handle space $\mathcal{H} = \{0,1\}^{128}$ that encodes a type tag, written $\mathsf{type}(h) \in \mathcal{T}$. The off-chain *ciphertext store* is a partial map $\Gamma : \mathcal{H} \rightharpoonup \mathcal{C}$ maintained by the indexer (§4.5); on chain, only handles in $\mathcal{H}$ and commitments to Γ ever appear.

Threshold public-key encryption. Encifher's confidentiality rests on a (t, n)-threshold PKE scheme $\mathsf{TPKE} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{ShareDec}, \mathsf{Combine})$: $\mathsf{KeyGen}(1^\lambda, t, n) \to (\mathsf{pk}, (\mathsf{sk}_1, \ldots, \mathsf{sk}_n))$ produces a master public key and n secret-key shares; $\mathsf{Enc}(\mathsf{pk}, m) \to c$ encrypts $m \in \mathcal{P}$; $\mathsf{ShareDec}(\mathsf{sk}_i, c) \to d_i$ produces a *partial decryption*; and $\mathsf{Combine}(\{d_i\}_{i \in S}) \to m$ recovers the plaintext from any S with $|S| \geq t$. Correctness requires $\mathsf{Combine}(\{\mathsf{ShareDec}(\mathsf{sk}_i, \mathsf{Enc}(\mathsf{pk}, m))\}_{i \in S}) = m$ for every $|S| \geq t$; we write $\mathsf{Dec}(c)$ for this combined decryption. Security is the standard threshold IND-CPA notion: no PPT adversary corrupting up to $t-1$ shares distinguishes encryptions with advantage better than $\mathsf{negl}(\lambda)$. We treat TPKE abstractly here and instantiate it in §4.7.

Remark. Encifher instantiates TPKE as a *hybrid* scheme over the prime-order secp256k1 group: each value is encrypted with AES-256-GCM [19] under a fresh content key, and that content key is wrapped with threshold ECIES/ElGamal over secp256k1, so the committee can threshold-decrypt the content key without reconstructing the master secret [20, 21].

Merkle commitments. A Merkle commitment scheme $(\mathsf{MkRoot}, \mathsf{MkPath}, \mathsf{MkVfy})$ maps leaves $(\ell_1, \ldots, \ell_m)$ to a root $R = \mathsf{MkRoot}(\ell_1, \ldots, \ell_m)$ such that $\mathsf{MkVfy}(R, \ell_j, \mathsf{MkPath}(j)) = 1$, and is *binding*: under collision- resistance of H, no PPT adversary finds two distinct leaves verifying at the same position under one root, except with probability $\mathsf{negl}(\lambda)$.

Signatures and attestation. $(\mathsf{Sign}, \mathsf{Vfy})$ is an EUF-CMA signature scheme. A remote attestation binds an enclave measurement μ (its PCR set, §2.2) to a public key pk_{enc}; we write $\mathsf{Att}(\mu, \mathsf{pk}_{\text{enc}})$ for a valid attestation and assume attestation *unforgeability*: no PPT adversary produces a valid $\mathsf{Att}(\mu^\star, \cdot)$ for the measurement $\mu^\star$ of code it does not run, except with probability $\mathsf{negl}(\lambda)$. This is the formal content of the TEE-integrity assumption (§3).

3 Threat Model and Security Goals

3.1 Parties

The system comprises: *users*, who encrypt inputs client-side and submit Solana transactions; the *on-chain programs*, a set of Solana programs that hold handles, enforce access control, and verify attested results (§4); the *coprocessor*, off-chain services (a relayer, batcher, processor, indexer, and submitter), of which the *processor* runs inside a TEE and performs decryption and computation; the *decryption committee*, a set of independently operated enclaves that jointly hold the master decryption key as a threshold secret; the *data-availability (DA) layer*, which stores ciphertexts; and the *Solana validators*, who order and finalize transactions.

3.2 Security goals

We separate three goals.

Confidentiality. An adversary who observes all on-chain state, all DA contents, all network traffic, and the non-enclave portions of the coprocessor learns nothing about the plaintext values underlying handles, beyond what is revealed by the application's intended outputs (e.g., a publicly posted swap settlement) and by unavoidable metadata (§3.6).

Integrity. An accepted state transition corresponds to the correct evaluation of the requested operation on the correct ciphertexts; formally, it preserves the resolution invariant (Def. 4): the coprocessor cannot substitute, replay, or forge a result without detection by the on-chain programs.

Availability/liveness. Honored requests eventually produce results. We treat liveness as a weaker goal than confidentiality and integrity, and we state explicitly where Encifher currently relies on a single coprocessor instance for liveness (§10).

3.3 Trust assumptions

Encifher's guarantees hold under the following assumptions, which we state plainly rather than claim away. We give the two load-bearing ones formally.

Assumption 1 (Enclave integrity and attestation)**.** The TEE preserves the integrity of code and data executing in an enclave, and attestation is unforgeable (§2.4): a valid $\mathsf{Att}(\mu, \mathsf{pk}_{\mathrm{enc}})$ implies that code with measurement μ produced $\mathsf{pk}_{\mathrm{enc}}$, and an enclave running an approved image does not leak its in-memory plaintext beyond its intended outputs.

Assumption 2 (Threshold non-collusion)**.** Of the n committee enclaves holding shares of the master decryption key, fewer than the reconstruction threshold t are simultaneously compromised or colluding.

In addition we assume: a *vendor attestation root* that honestly certifies genuine Trust Domains (under TDX the trusted base is Intel and the TDX Module, and the cloud operator is excluded from the confidentiality TCB; §7 discusses the implication); *Solana safety and liveness*, on which Encifher relies for ordering, access-control enforcement, and the integrity of committed results; *DA availability*, that committed ciphertexts remain retrievable; and *client correctness*, that user software encrypts inputs and forms handles correctly (client key custody is out of scope).

3.4 Adversary capabilities (in scope)

We consider a PPT adversary $\mathcal{A}$ who may: read all on-chain state, all DA contents, and all network messages; submit arbitrary transactions and malformed inputs; operate or collude with the relayer, batcher, indexer, submitter, DA operator, and up to $t-1$ committee enclaves; attempt to replay or roll back coprocessor state, equivocate, or withhold; and mount denial-of-service against open endpoints. Integrity must hold against *any* such adversary, including one who fully controls the off-chain pipeline, because integrity is anchored on chain by attestation and Merkle commitments rather than by trusting the operators.

3.5 Security goals, formally

We cast the two cryptographic goals as games against $\mathcal{A}$.

Definition 1 (Confidentiality)**.** In game $\mathsf{Exp}^{\mathrm{conf}}_{\mathcal{A}}(\lambda)$, $\mathcal{A}$ adaptively issues operation requests and submits two equal-length plaintext vectors $(\vec{m}_0, \vec{m}_1)$ that induce *identical* public outputs and metadata

(operation structure, touched handles, timing). The challenger encrypts $\vec{m}_\beta$ under pk for a secret bit $\beta \xleftarrow{\$} \{0,1\}$ and runs the system; $\mathcal{A}$, given the full on-chain, DA, and network transcript and the internal state of up to $t-1$ enclaves, outputs β'. Its advantage is $\mathsf{Adv}^{\mathrm{conf}}_{\mathcal{A}}(\lambda) = \left|\Pr[\beta' = \beta] - \frac{1}{2}\right|$, that is, how much better than a blind coin-flip $\mathcal{A}$ guesses which of the two plaintext vectors was encrypted; confidentiality (§7) requires this advantage to be negligible.

Definition 2 (Integrity)**.** In game $\mathsf{Exp}^{\mathrm{int}}_{\mathcal{A}}(\lambda)$, $\mathcal{A}$ controls the entire off-chain pipeline and wins if the on-chain manager program accepts a committed result for some output handle h_{out} whose stored ciphertext violates the resolution invariant (Def. 4). Its advantage is $\mathsf{Adv}^{\mathrm{int}}_{\mathcal{A}}(\lambda) = \Pr[\mathcal{A} \text{ wins}]$.

§7 proves both advantages negligible under Assumptions 1–2 and standard cryptographic assumptions.

3.6 Out of scope and residual leakage

Side channels. A complete defense against microarchitectural side channels on the TEE is explicitly *out of scope*: we do not claim immunity to them, and we do not treat them as part of the adversary that Def. 1 quantifies over for a *single* enclave. Encifher's secret-handling code is written to avoid secret-dependent control flow and memory-access patterns where practical, and the VM-level enclave does not share a physical core with untrusted tenants; but residual channels (e.g., single-stepping or cache attacks on TDX [22, 23]) are possible. We bound their impact rather than assume their absence: the threshold committee (Assumption 2) ensures that leakage from any single enclave does not by itself reveal user secrets, so a side channel must defeat t enclaves at once to fall within the confidentiality game. Physical attacks on the host hardware are out of scope.

Metadata. Like other systems that hide values but not relationships, Encifher leaks *metadata*: the existence and timing of confidential operations, the identities of the accounts (and thus handles) they touch, and the resulting public outputs. Encrypting values does not encrypt the transaction graph; Def. 1 reflects this by quantifying only over plaintext pairs that fix this metadata. Applications requiring sender/receiver unlinkability must layer additional techniques (§10).

Trustlessness. Encifher does not claim trustlessness; it makes its trust assumptions explicit and keeps them minimal. It reduces the trust required to run confidential computation on Solana to the assumptions above; it does not eliminate trust in hardware. The appropriate comparison is not "trusted vs. trustless" but a quantified point on a frontier (§9): Encifher trades a hardware-and-threshold trust assumption for performance and generality that the purely cryptographic alternatives do not currently deliver on a high-throughput chain.

4 Design

4.1 Overview

Encifher separates a Solana application into an *on-chain* part that manipulates only references to encrypted values and an *off-chain* coprocessor that performs the confidential computation inside a TEE. The flow of a single confidential operation, summarized in Figure 2, is:

1. A user encrypts an input client-side, obtaining a ciphertext and a 128-bit *handle* that references it (§4.2); the ciphertext is stored off chain, the handle is passed into a Solana transaction.

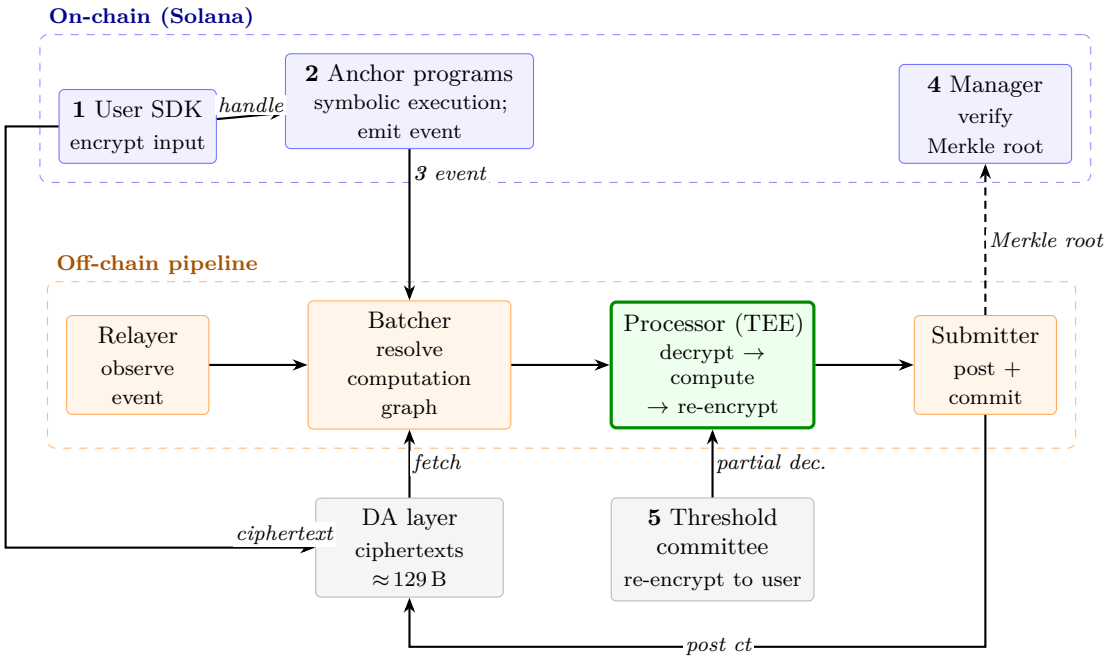

Figure 2: Lifecycle of one confidential operation (numbers match the steps above). The on-chain programs only manipulate handles and verify the committed Merkle root; the off-chain pipeline resolves the operation inside the TEE, posts the new ciphertext to the DA layer, and the threshold committee re-encrypts results to authorized users. Plaintext never leaves the enclave.

2. An on-chain program records the requested operation over handles (e.g., "$h_c \leftarrow \mathsf{add}(h_a, h_b)$") via *symbolic execution*, emitting an event; no plaintext or ciphertext is processed on chain (§4.3).
3. The off-chain *relayer* observes the event and enqueues the request; the *batcher* groups requests by handle dependency and time window and fetches the referenced ciphertexts; the *processor*, running inside the TEE, decrypts the ciphertexts, evaluates the operation on plaintext, re-encrypts the result, and produces a signed, Merkle-anchored commitment (§4.5).
4. The *submitter* posts the new ciphertexts to the DA layer and commits the batch's Merkle root on chain through the manager program; the *indexer* records the handle-to-ciphertext mapping for retrieval (§4.6).
5. When a user is authorized to learn a value, a *threshold-decryption* protocol over the committee re-encrypts it to the user, who decrypts client-side (§4.7).

The remainder of this section details each element, and then explains how the on-chain representation is laid out so that Solana's scheduler executes independent confidential operations in parallel while respecting dependencies (§4.9).

4.2 Encrypted values as handles

An encrypted value is represented on chain by a *handle* $h \in \mathcal{H} = \{0,1\}^{128}$ that maps to a ciphertext stored off chain, and that encodes a type tag $\mathsf{type}(h) \in \mathcal{T}$ so the on-chain programs can type-check operations before they are scheduled. Handles are the *only* representation of encrypted values that ever appears on chain; this is what keeps confidential applications within Solana's transaction-size and compute budgets (§2.1). A user obtains a handle in one of two ways: by *encrypting* a fresh

input client-side through the SDK (§5), producing $c = \mathsf{Enc}(\mathsf{pk}, m)$ together with its handle; or as the *output* of an operation on existing handles, in which case the handle is derived deterministically before the corresponding ciphertext exists (§4.3).

Ciphertexts are encrypted under a single system master public key pk, so that any two ciphertexts are mutually computable inside the enclave without per-pair key negotiation; the corresponding secret is held only as a threshold sharing across the committee (§4.7). Concretely, a value is sealed as a *hybrid* ciphertext: an AES-256-GCM payload under a fresh content key, with the content key wrapped to pk by threshold ECIES/ElGamal over secp256k1 (§4.7). The wrapped-key envelope, a 33-byte compressed ephemeral point plus the GCM nonce, tag, and wrapped key, is a small fixed overhead: an 8-byte balance becomes an $\approx$129-byte ciphertext, roughly eight times the 16-byte on-chain handle and sixteen times the plaintext. This asymmetry, a thin handle on chain, a fat ciphertext in the DA layer, is the quantitative basis for keeping ciphertexts off chain (§4.6).

4.3 Symbolic execution on chain

On-chain programs never compute on ciphertext. Instead they perform *symbolic execution*: an operation over handles produces a new *output handle* that is a deterministic function of the operation, its input handles, and a flag distinguishing ciphertext from scalar operands, and emits an event recording the operation. We make this precise.

Definition 3 (Operation signature and handle derivation)**.** An operation op of arity k carries a typing rule $\mathsf{op} : \tau_1 \times \cdots \times \tau_k \to \tau_{\text{out}}$. Given input handles $h_1, \ldots, h_k$ with $\mathsf{type}(h_i) = \tau_i$ and a scalar-operand flag $b \in \{0, 1\}$, the output handle is

$$h_{\text{out}} \;=\; \mathsf{H}_{\mathsf{der}}(\mathsf{op}, h_1, \ldots, h_k, b) \in \mathcal{H}, \qquad \mathsf{type}(h_{\text{out}}) = \tau_{\text{out}},$$

where $\mathsf{H}_{\mathsf{der}}$ truncates H to 128 bits and embeds the output type tag. The on-chain program records the tuple $(\mathsf{op}, h_1, \ldots, h_k, b, h_{\text{out}})$ and emits it as an event; it performs no cryptographic work.

Because output handles are deterministic, a program can reference the result of an operation *before* the coprocessor has computed the corresponding ciphertext, and can chain further operations on it, exactly as ordinary transactions chain on expected results. The on-chain cost of a confidential operation is therefore that of recording a handle and emitting an event, independent of the cost of the cryptographic work, and confidential operations compose across instructions and transactions.

A *confidential computation* is a finite directed acyclic graph $G = (V, E)$ whose vertices are recorded operations, with an edge $u \to v$ whenever v takes the output handle of u as an input. Let $[\![h]\!] \in \mathcal{P}$ denote the plaintext that handle h represents. The coprocessor's task is to maintain the store Γ so that the following invariant holds.

Definition 4 (Resolution invariant)**.** For every recorded operation $(\mathsf{op}, h_1, \ldots, h_k, b, h_{\text{out}})$,

$$\mathsf{Dec}\big(\Gamma(h_{\text{out}})\big) \;=\; \mathsf{op}^{\mathcal{P}}\big(\mathsf{Dec}(\Gamma(h_1)), \ldots, \mathsf{Dec}(\Gamma(h_k))\big),$$

where $\mathsf{op}^{\mathcal{P}}$ is the plaintext semantics of op and Dec is threshold decryption of the stored ciphertext (§4.7).

Equivalently, "decrypt then compute" and "derive the handle, resolve it, then decrypt" commute. The processor enforces Def. 4 by decrypting the inputs inside the enclave, evaluating $\mathsf{op}^{\mathcal{P}}$ on plaintext, and re-encrypting under pk to populate $\Gamma(h_{\text{out}})$ (§4.5); §7 shows that no other store can be made acceptable on chain except with negligible probability.

4.4 On-chain programs

Encifher's on-chain logic is divided into four Solana programs, written with the Anchor framework, each with a distinct responsibility.

Wrapper program. The user-facing entry point. It wraps a public SPL token into its confidential representation (and back), and routes user requests into the system, creating the handle accounts that subsequent operations reference.

Encrypted-token (EToken) program. Maintains confidential token state: per-account confidential balances represented as handles, and the confidential transfer/mint/burn operations expressed symbolically over those handles.

Executor program. Records the privacy-preserving operations (the symbolic computation requests) that the coprocessor must resolve, performing on-chain type-checking (Def. 3) and access-control enforcement before emitting the corresponding events.

Manager program. The trust anchor for results. It records the committed Merkle roots of processed batches together with the coprocessor's attested signature, and gates acceptance of a result on successful verification of that attestation (§4.8). It also mediates on-chain decryption requests (e.g., withdrawals, §4.7).

4.5 The off-chain coprocessor pipeline

The coprocessor is a pipeline of services; only the *processor* runs inside the TEE, which keeps the trusted computing base small.

Relayer. Subscribes to on-chain events, decodes each symbolic operation over handles, and enqueues it (in our implementation, onto a durable message queue) for batching.

Batcher. Groups queued operations into *batches* by handle overlap and a time window, resolving dependency order within a batch, a topological order of the induced subgraph of G, and fetches the input ciphertexts referenced by the batch's handles from the indexer.

Processor (in TEE). For each operation, obtains the input plaintexts $m_i = \mathsf{Dec}(\Gamma(h_i))$ via the threshold committee (§4.7), computes $m_{\text{out}} = \mathsf{op}^{\mathcal{P}}(m_1, \ldots, m_k)$ inside the enclave, re-encrypts $\Gamma(h_{\text{out}}) \leftarrow \mathsf{Enc}(\mathsf{pk}, m_{\text{out}})$, and accumulates the batch's results into a Merkle tree it signs (§4.6). Plaintext never leaves the enclave; the construction establishes Def. 4 by construction.

Indexer. Stores the handle-to-ciphertext mapping Γ (with the verification metadata of §4.6) and serves retrieval; it functions as a cache over the DA layer.

Submitter. Persists the batch's new ciphertexts to the DA layer (compressing them first) and commits the batch's signed Merkle root on chain through the manager program, with a reference to the DA record.

4.6 DA-backed verifiable batching

Ciphertexts live in a public DA layer, accessed through a per-kilobyte submission service, while the chain stores only handles and the signed Merkle roots that commit to batches of results.

Definition 5 (Batch commitment)**.** A batch is a set $B = \{(h_j, c_j, \rho_j)\}_{j=1}^{m}$ of output handles, their ciphertexts $c_j = \Gamma(h_j)$, and operation parameters ρ_j. The processor forms leaves $\ell_j = \mathsf{H}(c_j \,\|\, \rho_j \,\|\, h_j)$, computes the root $R = \mathsf{MkRoot}(\ell_1, \ldots, \ell_m)$, and signs $\sigma = \mathsf{Sign}(\mathsf{sk}_{\text{enc}}, R)$ with its attestation-bound key.

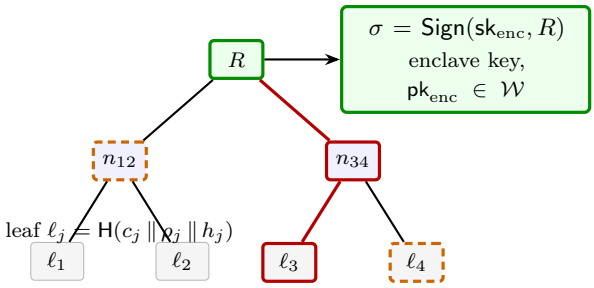

Figure 3: DA-backed batch commitment (Def. 5). The enclave Merkle-hashes a batch's result ciphertexts into a root R and signs it with its attestation-bound key ($\mathsf{pk}_{\mathrm{enc}} \in \mathcal{W}$). To check that ℓ_3 is committed, a consumer recomputes the path to the root (solid red: $\ell_3 \to n_{34} \to R$) using the proof siblings (dashed orange: ℓ_4, n_{12}), and accepts iff the signature, the attestation-bound key, and the inclusion check all hold (predicate VfyCT).

A consumer that retrieves (c_j, ρ_j, h_j) together with a path π_j, the root R, and the signature σ accepts the ciphertext iff

$$\mathsf{VfyCT} \;=\; \underbrace{\big[\mathsf{Vfy}(\mathsf{pk}_{\mathrm{enc}}, R, \sigma) = 1\big]}_{\text{signed by an enclave key}} \wedge \underbrace{\big[\mathsf{pk}_{\mathrm{enc}} \in \mathcal{W}\big]}_{\text{attestation-bound}} \wedge \underbrace{\big[\mathsf{MkVfy}\big(R,\, \mathsf{H}(c_j \,\|\, \rho_j \,\|\, h_j),\, \pi_j\big) = 1\big]}_{\text{inclusion under } R},$$

where $\mathcal{W}$ is the whitelist of attestation-bound keys pinned on chain (§4.8). The on-chain manager program performs the signature/attestation conjuncts when committing a root; clients and the batcher perform the inclusion conjunct on retrieval. Any tampering with c_j or ρ_j changes the recomputed leaf and fails inclusion. Batching amortizes the fixed per-batch costs, one on-chain commitment and one DA write, over m operations, which is central to Encifher's cost profile (§6).

4.7 Key management: a threshold-decryption committee

The master secret that decrypts ciphertexts is the system's most sensitive asset, and a single enclave holding it would be a single point of catastrophic, *retroactive* failure (§1, [9]). Encifher instantiates TPKE (§2.4) with a (t, n) threshold scheme held across a committee of independently operated enclaves. To decrypt a ciphertext c, whether the processor decrypting batch inputs or a user reading an authorized value, a quorum S with $|S| \geq t$ each computes a partial decryption $d_i \leftarrow \mathsf{ShareDec}(\mathsf{sk}_i, c)$ *inside its own enclave*, and the partials are combined, $m \leftarrow \mathsf{Combine}(\{d_i\}_{i \in S})$, without any party ever reconstructing the master secret. Compromising a single enclave thus reveals at most one share and no plaintext, so confidentiality survives up to $t - 1$ simultaneous enclave compromises (Assumption 2).

Provisioning and governance. Each committee node obtains its share only after its TDX attestation verifies against the coprocessor's on-chain-registered measurement. Encifher realizes this with Phala dStack's blockchain-governed key-management service: a key is released to a Trust Domain only if its attestation matches an approved code hash, and the root key is itself held as a Shamir sharing across the key-management nodes, with support for share rotation [14, 15]. This makes the non-collusion assumption (Assumption 2) concrete and binds committee membership to attested, audited code rather than to operator identity.

Remark (ElGamal instantiation). Encifher instantiates the committee key over *secp256k1*, a prime-order group ($\mathbb{G} = \langle g \rangle$, order q, cofactor 1). The master secret is (t, n)-shared by distributed key

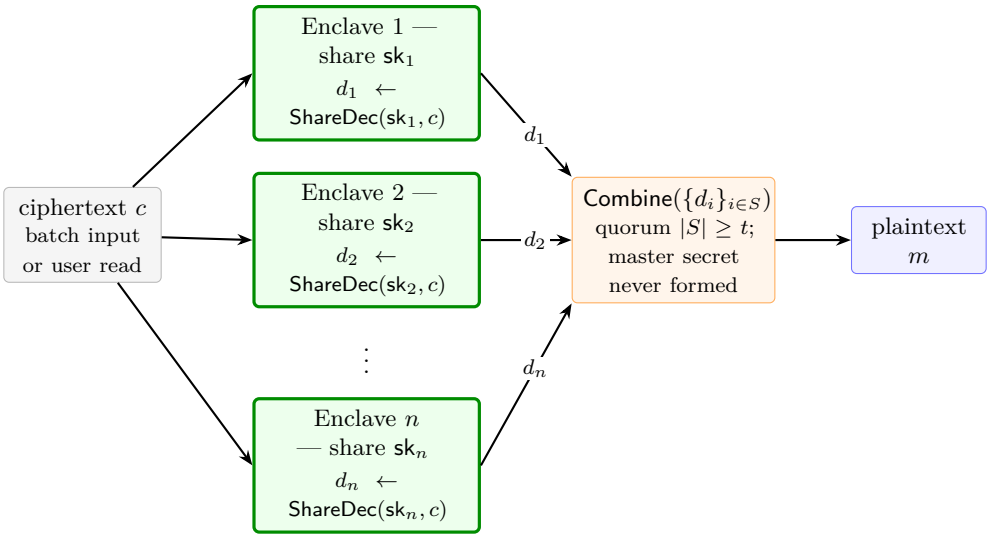

Figure 4: Threshold-decryption committee. The master secret is shared (t, n) across independently operated enclaves; each holds only its share sk_i and emits a partial decryption $d_i \leftarrow \mathsf{ShareDec}(\mathsf{sk}_i, c)$ from inside its own enclave. Any quorum S with $|S| \geq t$ recovers the plaintext via $\mathsf{Combine}(\{d_i\}_{i\in S})$ without any party ever materializing the master secret, so fewer than t compromised enclaves reveal nothing (Assumption 2).

generation [24–26], giving shares $\mathsf{sk}_i = f(i)$ of a degree-$(t-1)$ polynomial with $f(0) = \mathsf{sk}$ and $\mathsf{pk} = g^{\mathsf{sk}}$. To unwrap a content key whose ECIES envelope carries ephemeral point $R = g^r$, each member of a quorum S publishes a partial decryption $d_i = R^{\mathsf{sk}_i}$, and the shared secret is recovered in the group by Lagrange interpolation in the exponent, $R^{\mathsf{sk}} = \prod_{i\in S} d_i^{\lambda_{i,S}}$ with $\lambda_{i,S} = \prod_{j\in S\setminus\{i\}} \frac{j}{j-i} \bmod q$, never reconstructing sk at any single node.

The threshold (t, n) is configurable per deployment through dStack's key-management service, which performs the distributed key generation and supports share rotation without changing the public key [14]; a new member is admitted, and its share provisioned, only after its TDX attestation verifies against the approved measurement (§4.8).

User-facing decryption (re-encryption). When an application authorizes a user to learn a value, the user opens a short-lived *re-encryption session* (Figure 5): the user proves authorization (ultimately gated by on-chain permission state), the committee threshold- re-encrypts the requested ciphertext to an ephemeral session key the user controls, and the user decrypts client-side. On-chain decryption, for example, unwrapping a confidential balance back to a public SPL token on withdrawal, follows the same authorization path but is mediated by the manager program.

4.8 Anchoring trust on chain via attestation

The integrity of every result reduces to the claim that it was produced by the correct, audited coprocessor image. Encifher makes this checkable on Solana: the expected TDX measurement μ (the MRTD/RTMR values, §2.2) is registered on chain; the coprocessor's signing key is admitted to the whitelist $\mathcal{W}$ only through a TD Quote whose measurements match μ ($\mathsf{Att}(\mu, \mathsf{pk}_{\mathrm{enc}})$); and the manager program accepts a committed Merkle root only if signed by a key in $\mathcal{W}$. Debug or mismatched Trust Domains are rejected, so μ is always a genuine measurement of approved code, the "code-is-law" key-release discipline dStack enforces through its governance contracts [14, 15].

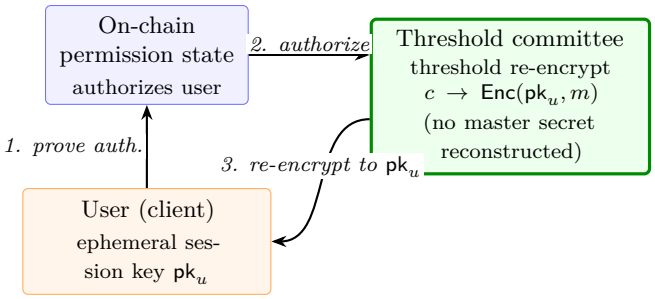

Figure 5: User-facing decryption as a re-encryption session. The user proves authorization (gated by on-chain permission state); the committee then threshold-re-encrypts the requested ciphertext to a user-controlled ephemeral key pk_u, which the user decrypts client-side. The master secret is never reconstructed, and on-chain decryption (e.g., withdrawal) follows the same path mediated by the manager program.

A TD Quote is signed with ECDSA on the secp256r1 (P-256) curve. Full DCAP verification is more than one signature: it also checks the Intel certificate chain and TCB collateral and parses the quote body for MRTD/RTMR. Encifher therefore uses an *attestation-verifier service* that performs the full DCAP verification once and re-signs a compact assertion (the approved measurement together with the coprocessor's public key) under an `ed25519` key; the manager program then need only check that assertion through Solana's native ed25519 precompile before admitting the key to $\mathcal{W}$. The verifier is itself attested, so this adds a checkable, not a blind, party. Solana also activated a native secp256r1 (NIST P-256) precompile on mainnet in 2025 [17]; since a TD Quote is ECDSA-signed over that curve, verifying the quote's signature *directly* on chain is feasible. As with the ed25519 and secp256k1 precompiles, it is checked at the transaction level rather than callable as a syscall, and it verifies only the signature, so TD Quote parsing and certificate-chain validation would still run off chain; we treat this fully on-chain path as a heavier alternative we did not require.

4.9 Exploiting Sealevel: per-handle accounts and free dependency ordering

The decisions above are shared, in spirit, with FHE coprocessors on the EVM. The element specific to Solana, and, we argue, the most interesting part of the design, is how the on-chain representation cooperates with Sealevel rather than fights it. Encifher stores each handle in its *own* account, at a program-derived address $\mathsf{acct}(h) = \mathsf{PDA}(pid, h)$ deterministically derived from the handle.

We model the on-chain realization. Operation v with output handle h^v_{out} and input handles $\{h^v_i\}$ is realized by a transaction tx_v that declares writable set $W(v) = \{\mathsf{acct}(h^v_{\text{out}})\}$ and read set $R(v) = \{\mathsf{acct}(h^v_i)\}$. Two facts follow directly from Solana's execution model.

First, *the transaction's declared read/write set is the dependency graph of the confidential computation.* Solana's runtime must already parse this set to schedule the transaction; in doing so it learns exactly which encrypted operations depend on which. Encifher obtains the dependency graph G of §4.3 for free, as a first-class part of the transaction, where an EVM coprocessor must reconstruct it.

Second, the base-layer scheduler executes confidential operations in parallel and in dependency order, with no coordination logic of our own.

Proposition 1 (Sealevel realizes the symbolic-execution order)**.** *Let $G = (V, E)$ be a confidential computation under the per-handle account assignment above, with collision-free handles. Then under Sealevel's account-lock scheduling:* (i) *transactions $\mathrm{tx}_u, \mathrm{tx}_v$ share a locked account iff $(u, v) \in E$ or $(v, u) \in E$, and any non-aborting execution orders the producer before the consumer; and* (ii) *any set of pairwise non-adjacent operations (an antichain of G) may execute concurrently.*

Proof sketch. Sealevel serializes two transactions iff their account sets conflict, i.e., one's write set intersects the other's read-or-write set. With one PDA per handle and collision-free handles, tx_u and tx_v touch a common account iff they share a handle, which by Def. 3 occurs iff one's output handle is the other's input, i.e., iff $(u, v) \in E$ or $(v, u) \in E$; this gives the conflict characterization. For an edge $(u, v) \in E$, tx_v reads $\mathsf{acct}(h^u_{\mathrm{out}})$, which is held for write by tx_u, so the two never run concurrently; and since the executor program requires an input handle account to be populated before consuming it (Def. 4), the only non-aborting order runs u before v, giving (i). If u, v are non-adjacent their account sets are disjoint, so no lock is shared and the scheduler may run them in parallel, giving (ii). □

A global ciphertext registry would instead place every h_{out} in one account, making every pair of operations conflict and collapsing G's execution to a total order, the serialization the per-handle layout avoids. Per-handle PDAs also confine Solana's per-account local fee market, so heavy use of one set of handles does not raise costs for unrelated users. The off-chain batcher's dependency resolution (§4.5) mirrors this on-chain order.

4.10 Choosing a finality level for reveals

Because Solana exposes both fast *confirmed* and slower *finalized* states (§2.1), Encifher must decide which gates an irreversible action. Internal handle-to-handle computation may proceed optimistically on confirmed state, since a reorg merely reschedules deterministic work; but *reveals*, threshold re-encryption to a user, or unwrapping to a public token, are gated on *finalized* state, so a value is never disclosed or a balance unwrapped on the basis of a transaction that could still be rolled back.

5 Implementation

Encifher is implemented as a set of Solana programs and an off-chain coprocessor and is deployed in production, where it has served over 5,000 users and more than 50,000 confidential operations (§8). The on-chain programs and the client SDK are open-source; the coprocessor is currently closed.

On-chain programs. The four Solana programs (§4.4) are written in Rust with the Anchor framework; the measured on-chain cost of a confidential operation is reported in §6.1.

Coprocessor. The coprocessor is written in Rust and structured around a single trust boundary: only the *processor*, the one component that handles plaintext (threshold-assisted decryption, in-enclave evaluation, re-encryption, and signing of Merkle-accumulated results), runs inside the enclave. The surrounding services, which ingest on-chain events, validate permissions, assemble dependency-ordered batches, index the handle-to-ciphertext mapping, and submit ciphertexts and committed roots, run outside it and treat the enclave as a plaintext-evaluation oracle. This keeps the trusted computing base small: everything that touches secrets lives inside the measured image, while orchestration and storage do not.

TEE deployment and attestation. The processor (and each committee enclave) is packaged as a reproducibly built container image and run inside Intel TDX Trust Domains via Phala dStack [14]: dStack launches each image from its `docker-compose` specification inside a confidential VM, measures the image and runtime into MRTD/RTMR, and provisions key material through its blockchain-governed KMS, which releases an application's keys only to a Trust Domain whose attestation matches the on-chain-approved code hash and holds the root key as a Shamir sharing across KMS nodes [14, 15]. An upgraded-but-approved image re-derives the same key material, so the committee survives restarts and controlled upgrades. The committee threshold is configurable through dStack's KMS, and code-hash governance is enforced by dStack's KmsAuth/AppAuth contracts (§4.7).

Client SDK. Applications integrate through a TypeScript SDK that performs client-side encryption, turning a plaintext and a declared type into a ciphertext and its handle, and manages the re-encryption sessions through which a user decrypts an authorized value (§4.7). This keeps key custody and plaintext on the client and exposes encrypted types to application developers without requiring them to handle ciphertexts directly. The SDK exposes encrypted unsigned-integer types on which applications express arithmetic (addition, subtraction), comparison, and conditional select/branching, enough for general confidential logic, not only transfers.

6 Evaluation

We evaluate Encifher along four axes: (Q1) what a confidential operation costs on chain; (Q2) the cost of the cryptographic layer, and how it compares to an FHE backend; (Q3) how cost scales with batching and committee size; and (Q4) the end-to-end latency of the deployed system. On-chain figures are read from production transactions on Solana mainnet; cryptographic figures are microbenchmarks of Encifher's components.

Cost model. For a batch of N operations, the amortized on-chain-plus-DA cost per confidential operation is

$$\mathcal{C}_{\mathrm{op}}(N) \;=\; c_{\mathrm{hdl}} \;+\; \frac{C_{\mathrm{root}} + C_{\mathrm{DA}}}{N},$$

where c_{hdl} is the per-operation cost of recording a handle and emitting its event, and $C_{\mathrm{root}}, C_{\mathrm{DA}}$ are the fixed per-batch costs of one signed Merkle-root commitment and one DA write; as $N \to \infty$, $\mathcal{C}_{\mathrm{op}}(N) \to c_{\mathrm{hdl}}$. The remainder of this section instantiates the terms.

Setup. Cryptographic microbenchmarks run on commodity CPUs: an AMD Ryzen 7 5800U and an Apple M4 Pro for the threshold layer, and AWS EC2 instances (8–192 vCPU) for the symmetric layer; executing inside an Intel TDX domain adds an estimated ∼15% on CPU and $< 5\%$ on a confidential GPU [27]. Reported cryptographic latencies *exclude* Solana confirmation, which adds ∼0.4 s (confirmed) to ∼12.8 s (finalized) per §4.10.

6.1 On-chain cost (Q1)

Because a confidential operation touches the chain only to record 128-bit handles and emit events, its on-chain footprint is small and composes with ordinary DeFi. A representative production *confidential swap*, two anonymous-transfer legs wrapped around a Jupiter route (§8), consumes

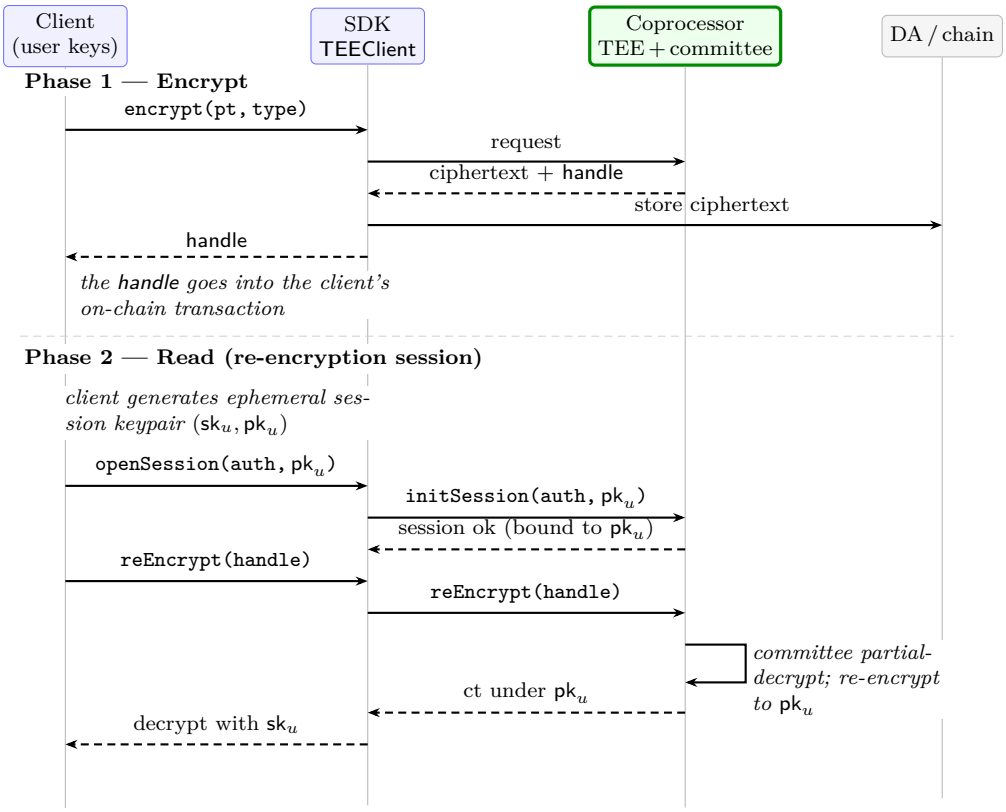

Figure 6: SDK message flow. **Encrypt:** the client turns a plaintext and a declared type into a ciphertext (posted to the DA layer) and a handle (used in on-chain transactions); key custody and plaintext stay on the client. **Read:** the client first generates an ephemeral session keypair $(\mathsf{sk}_u, \mathsf{pk}_u)$ and opens an authorized re-encryption session bound to the session public key pk_u; the coprocessor, with the threshold committee, then re-encrypts the requested handle to pk_u, which the client decrypts locally with sk_u. Plaintext and the session secret never leave the client, and plaintext never leaves the enclave.

157,323 compute units (CU) in a *single* Solana transaction with a 15,000-lamport fee.[1] Table 1 breaks this down: Encifher's program accounts for 85,121 CU across its two anonymous-transfer invocations, and the underlying Jupiter swap for 72,202 CU. The whole transaction uses only ≈11% of the 1.4M-CU per-transaction budget, so a confidential swap fits comfortably on chain beside the public swap it wraps. Transaction sizes stay far under the 1,232-byte limit (391 bytes for an anonymous transfer, 646 for a deposit, 633 for a withdrawal), precisely because only handles, never ciphertexts, ride on chain.

6.2 Cost of the cryptographic layer (Q2)

The confidentiality cost is dominated by the threshold key operations, not by bulk encryption. The symmetric (AES-256-GCM) layer is effectively free: it runs at over 1.1×10^6 encryptions/s and 4.0×10^6 decryptions/s on 8 vCPUs, rising to 1.28×10^6 / 4.31×10^6 on 192 vCPUs, under a microsecond per value. The threshold layer over secp256k1 costs 0.08 ms for encryption-key generation (independent of the number of parties t) and, for partial decryption plus aggregation,

[1] Mainnet transaction `wzwXb8...ojZD6P`.

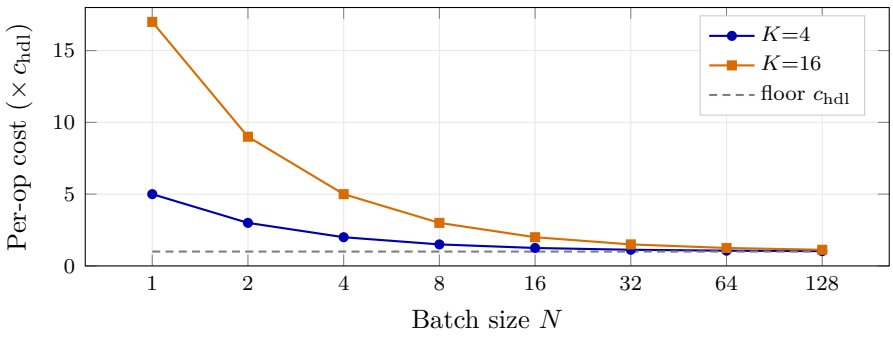

Figure 7: Amortization of the cost model $\mathcal{C}_{\text{op}}(N) = c_{\text{hdl}} + (C_{\text{root}} + C_{\text{DA}})/N$: the fixed per-batch cost (one signed Merkle root plus one DA write) is spread over N operations, so per-operation cost falls as $1/N$ toward the floor c_{hdl}. Curves show two ratios $K = (C_{\text{root}} + C_{\text{DA}})/c_{\text{hdl}}$; the exact constants are deployment-specific.

Table 1: On-chain cost of a production confidential swap (Solana mainnet).

Component	Compute units
Encifher program (2 anonymous-transfer legs)	85,121
Jupiter swap route (incl. DEX sub-calls)	72,202
Total transaction	157,323
Fee	15,000 lamports

scales with t as in Table 2: from 0.95 ms at $t = 2$ to 26.7 ms at $t = 32$.

Comparison to FHE and MPC. These costs are orders of magnitude below an FHE backend. Zama's fhEVM, the closest FHE coprocessor, runs at ∼20 transactions/s on CPU and leaves homomorphic operations many orders of magnitude slower than plaintext (§9); Encifher instead computes on plaintext inside the enclave, so a confidential operation pays only sub-microsecond symmetric encryption plus single-digit-to-tens-of-milliseconds threshold decryption, near-plaintext speed. Against an MPC backend such as Arcium, Encifher avoids per-operation interaction entirely: the only multi-party step is a single round of partial decryptions, whereas MPC pays communication rounds *throughout* the computation. Against Solana's native Token-2022 confidential transfers, which support only transfers and were disabled on mainnet at the time of writing (§9), Encifher runs general confidential logic and composes with external DeFi (Q1).

6.3 Scaling with batching and committee size (Q3)

Batching amortizes the fixed per-batch on-chain commitment and DA write (the $C_{\text{root}} + C_{\text{DA}}$ term). Table 3 reports batched partial decryption: time grows roughly linearly in batch size, 13 ms at batch 8 to 88 ms at batch 64 for a 4-node committee, so per-operation latency is bounded and the fixed costs amortize. Critically, enlarging the committee from 4 to 16 nodes barely changes the time, so confidentiality can be spread across more operators with little penalty, the property that makes the threshold committee (§4.7) practical rather than aspirational.

On the DA side, a value's ciphertext is ≈129 bytes (§4.2). DA layers typically bill per kilobyte with a per-submission floor, so batching is what drives the amortized DA cost down: in our

Table 2: Threshold key-operation cost over secp256k1 (ms), by number of parties t. Encryption-key generation is 0.08 ms for all t.

t	Partial dec.	Aggregation	Total
2	0.381	0.482	0.945
4	0.767	1.00	1.9
8	1.50	2.50	4.2
16	3.10	6.80	9.9
32	6.10	20.5	26.7

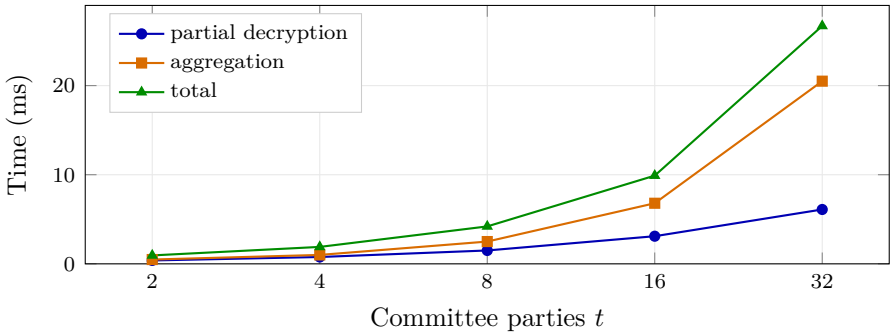

Figure 8: Threshold key-operation cost vs committee size t (secp256k1; same data as Table 2). Aggregation dominates as t grows, yet the total stays in the tens of milliseconds through $t = 32$.

deployment (one credit per kilobyte with a one-credit floor), batching ∼8 ciphertexts fills a single credit.

6.4 End-to-end latency (Q4)

We decompose user-perceived latency in the deployed system, measured on an Apple M3 Pro and excluding Solana confirmation. An anonymous transfer takes ≈360 ms of coprocessor processing plus ≈293 ms to batch-encrypt its (sender, receiver, mint) tuple (averaged over 100 runs; min 280 ms, max 556 ms); a deposit takes ≈221 ms and a confidential-balance read ≈650 ms. A confidential swap is two anonymous-transfer legs plus the relayer's decryption, consistent with the on-chain structure of §6.1.

6.5 Production deployment (Q4)

Encifher has served over 5,000 users and more than 50,000 confidential operations on Solana mainnet, production evidence that competing, non-deployed systems cannot provide.

7 Security Analysis

We analyze Encifher against the goals and adversary of §3, arguing each goal under the stated assumptions and then discussing residual risks we do not eliminate.

Table 3: Batched partial-decryption latency (ms, `u64`), by committee size and batch size. Enlarging the committee $4 \to 16$ has little effect.

Nodes	Batch size			
	8	16	32	64
4	13.1	23.0	45.3	87.5
8	13.2	22.8	42.4	84.5
16	13.6	20.4	43.7	82.0

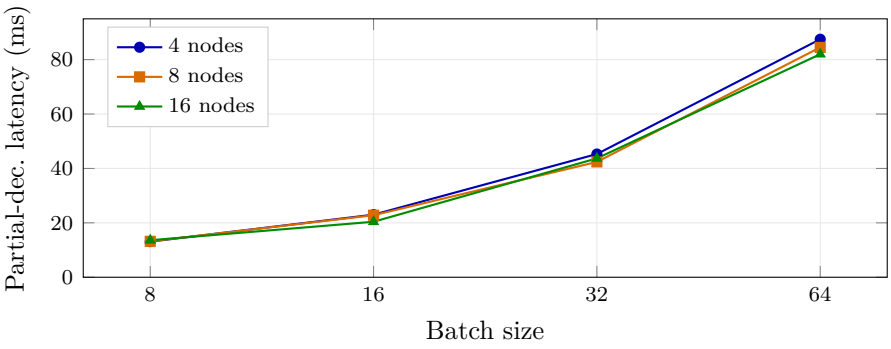

Figure 9: Batched partial-decryption latency vs batch size, for committees of 4, 8, and 16 nodes (same data as Table 3). The three curves nearly coincide: latency is roughly linear in batch size and almost insensitive to committee size, which is what makes spreading trust across more operators cheap.

7.1 Integrity

Theorem 1 (Integrity)**.** *Under collision-resistance of* H*, EUF-CMA security of* $(\mathsf{Sign}, \mathsf{Vfy})$*, the binding of the Merkle commitment, and Assumption 1,* $\mathsf{Adv}^{\mathrm{int}}_{\mathcal{A}}(\lambda) \leq \mathsf{negl}(\lambda)$ *for every PPT* $\mathcal{A}$*.*

Proof sketch. The manager program accepts a result only if the predicate VfyCT (§4.6) holds: the batch root carries a signature under a key $\mathsf{pk}_{\mathrm{enc}} \in \mathcal{W}$, and $\mathcal{W}$ admits only attestation-bound keys. To win, $\mathcal{A}$ must make the manager accept a result violating Def. 4, which requires one of: (a) forging a signature under some $\mathsf{pk}_{\mathrm{enc}} \in \mathcal{W}$, contradicting EUF-CMA; (b) exhibiting a second leaf that verifies at a committed position, contradicting binding/collision- resistance; or (c) obtaining a valid attestation $\mathsf{Att}(\mu, \cdot)$ for an enclave image that computes an incorrect result, contradicting Assumption 1, since the pinned measurement μ is that of the audited image which enforces Def. 4 by construction (§4.5). Each event occurs with probability $\mathsf{negl}(\lambda)$; a union bound gives the claim. Replay and substitution are excluded because each result is bound to its deterministically derived handle (Def. 3) and committed under the current root. □

Integrity is anchored on chain and does *not* rely on trusting the coprocessor operators. A result is accepted only if (i) its batch's Merkle root is signed by a key bound, through attestation, to an approved enclave image, and (ii) the ciphertext's leaf is proven to lie under that root (§4.6, §4.8). An adversary controlling the relayer, batcher, indexer, submitter, or DA operator can therefore delay or deny service, but cannot cause an incorrect result to be accepted: forging an accepted result would require either forging an enclave attestation (excluded by the TEE-integrity assumption) or a Merkle/signature collision. Substitution and replay are prevented because each result is bound to its

deterministically derived output handle and committed under a signed root; an attempt to serve a stale ciphertext fails the inclusion check against the current root. The reduction of integrity to "the right code ran on genuine hardware" is exactly what on-chain attestation verification establishes, and is the reason the verifier-key indirection of §4.8 must itself be sound.

7.2 Confidentiality

Theorem 2 (Confidentiality)**.** *Under threshold IND-CPA security of* TPKE*, Assumption 2, and Assumption 1 (enclave memory confidentiality),* $\mathsf{Adv}^{\mathrm{conf}}_{\mathcal{A}}(\lambda) \leq \mathsf{negl}(\lambda)$.

Proof sketch. We show that $\mathcal{A}$'s view is computationally independent of the challenge bit, so no PPT adversary distinguishes encryptions of $\vec{m}_0$ from those of $\vec{m}_1$. Recall that the two plaintext vectors must agree on all leaked *metadata*, the operation structure and the public outputs (Def. 1). $\mathcal{A}$'s view has four components, which we take in turn:

1. *Handles* are a deterministic function of the operation structure alone (Def. 3), not of the underlying plaintext. Since $\vec{m}_0$ and $\vec{m}_1$ induce the same structure, the handles are identical in both worlds and reveal nothing about the bit.
2. *Signed Merkle roots* commit only to the ciphertexts and their handles and parameters; given the ciphertexts they add no further information, as the commitment is binding rather than plaintext-revealing.
3. *Corrupted-enclave state.* By Assumption 2 the adversary controls at most $t-1$ committee members and so learns at most $t-1$ key shares; threshold IND-CPA guarantees these confer no decryption capability and no distinguishing advantage. By Assumption 1 the honest enclaves leak no plaintext beyond the metadata-fixed outputs that both worlds already share.
4. *Ciphertexts under* pk*.* All that remains is to distinguish the encrypted values themselves. A standard hybrid argument replacing the encryptions of $\vec{m}_0$ with those of $\vec{m}_1$ one value at a time reduces any such advantage to the threshold IND-CPA advantage of TPKE.

Combining the four steps, $\mathsf{Adv}^{\mathrm{conf}}_{\mathcal{A}}(\lambda) \leq \mathsf{negl}(\lambda)$. The bound is conditional on the assumptions: a side channel that violates Assumption 1 in t or more enclaves falls outside the model (§3.6), which is precisely why the committee threshold t is the security-critical parameter. □

Plaintext exists only inside enclaves: client-side before encryption, and within the processor and committee enclaves during computation and partial decryption. An adversary with the full on-chain, DA, and network view, and control of the non-enclave pipeline, sees only ciphertexts and handles. The critical question is therefore what a *compromised enclave* yields. Encifher is designed so that no single enclave compromise is catastrophic: because the master secret key is a t-of-n threshold sharing and is never reconstructed (§4.7), extracting the state of one enclave reveals at most one key share and the plaintext of the operations that enclave was actively processing, not the master key, and not the historical corpus of ciphertexts. This is a deliberate departure from the single-master-key design whose compromise enabled retroactive disclosure of an entire confidential ledger's history [9, 10]. Confidentiality holds as long as fewer than t committee enclaves are simultaneously compromised.

7.3 The TEE trust assumption, stated honestly

Encifher's confidentiality ultimately rests on the enclave keeping its memory secret, and we do not claim this is unconditional. We make three points explicit.

What is in the trusted base. Under Intel TDX, the confidentiality trusted base is the CPU, the Intel-signed TDX Module, and Intel as the attestation root; the host hypervisor, host OS, and cloud operator are *excluded* [13]. This is a stronger starting point than a memory-cleartext VM enclave (where the operator's hardware and firmware must be trusted to keep memory secret), because TDX encrypts and integrity-protects domain memory in hardware. The residual trust is in Intel and the comparatively young TDX Module rather than in a cloud provider, a real but *different* assumption, and one the threshold committee (§4.7), ideally spanning more than one operator, is designed to de-risk.

Side channels. TEEs have a substantial side-channel literature: process-level SGX enclaves have suffered repeated full breaks including attestation-key extraction [9, 28–30], and the memory-encrypted VM-level platforms have their own ciphertext and fault-injection attacks [31, 32]. TDX is no exception, and we do not claim otherwise: independent review found (and Intel fixed) vulnerabilities before release [33], the academic literature demonstrates reliable *single-stepping* and *cache* side channels that recover keys from non-constant-time code [22, 23], and *physical* DDR memory-bus interposition can extract secrets, including attestation keys, from TDX, SGX, and SEV-SNP alike [34]. We therefore treat TDX confidentiality as a strong-but-fallible assumption: we write the in-enclave secret-handling code to avoid secret-dependent control flow and memory-access patterns where practical, and, most importantly, bound the blast radius of a single-domain compromise (up to and including full key extraction by a TEE.Fail-class physical attacker) with the threshold committee, so that confidentiality holds unless at least t of the n committee members are compromised at once.

Attestation and upgrades. Integrity depends on the manager program pinning the correct enclave measurement and rejecting debug or mismatched-measurement attestations (§4.8). Enclave upgrades must rotate the pinned measurement under a controlled, auditable governance process rather than a silent signing key, so that the set of code that can ever decrypt is publicly known and multi-party-approved; this mirrors the recommended discipline for TEE key-sealing [35]. In Encifher this is enforced by dStack's governance contracts: a new coprocessor image must be approved, by code hash, in the AppAuth contract before the KMS will release keys to it, so the set of code that can ever decrypt is recorded on chain and changes only through that approval [14, 15].

7.4 Availability, rollback, and DA withholding

Encifher's liveness currently depends on the coprocessor and the committee being responsive, and on the DA layer serving committed ciphertexts; §3 classifies liveness as the weakest goal, and §10 discusses the present centralization. Two specific risks merit note. *Rollback/replay*: because enclaves are stateless across restarts, the design must prevent feeding an enclave stale state; Encifher binds results to deterministic handles and committed roots so that a replayed ciphertext fails verification, while freshness of the *committee key state* across restarts is ensured by dStack's KMS, which re-provisions an approved Trust Domain's key material deterministically after attestation rather than relying on enclave-local sealed state [14]. *DA withholding*: if the DA operator withholds a committed ciphertext, the corresponding value becomes unavailable (a liveness, not a confidentiality or integrity, failure); using a DA layer with its own availability guarantees mitigates this.

Table 4: Privacy modes for confidential payments; an application selects one per use case. *Shielded* and *anonymous* additionally break sender–recipient linkability by commingling balances in a shared encrypted pool (§3.6).

Mode	Amount	Tx graph	Linkability
Confidential (cf. Token-2022)	hidden	public	linkable
Shielded	hidden	hidden	broken
Anonymous	public	hidden	broken

7.5 What Encifher does not protect

Encifher hides values, not relationships: the existence, timing, and account adjacency of confidential operations remain public (§3.6). It does not defend against compromise of a user's client or keys, against physical attacks on host hardware, or against simultaneous compromise of t or more committee enclaves. These boundaries are stated so that the guarantees are not overread.

8 Deployed Applications

Encifher is not a prototype: it backs a production application offering confidential payments, swaps, and a cross-chain bridge on Solana. We describe each in terms of the primitives of §4, since each is a thin application over the same handle/symbolic-execution substrate. In production it has served over 5,000 users and more than 50,000 confidential operations to date, for which §6.5 reports the corresponding trace.

Confidential payments. A user wraps a public SPL token into its confidential representation through the wrapper program (§4.4), after which balances are held as handles and transfers are expressed symbolically over the encrypted-token program: a transfer subtracts an encrypted amount from the sender's balance handle and adds it to the recipient's, with the coprocessor resolving the arithmetic on plaintext inside the enclave. Encifher exposes this at three privacy levels that an application selects per use case: a *confidential* mode that hides amounts while leaving sender, recipient, and the transaction graph public (matching Token-2022 confidential transfers); a *shielded* mode that hides both amounts and the transaction graph; and an *anonymous* mode that hides the graph while leaving amounts public. The default confidential mode has the leakage profile of §3.6; the shielded and anonymous modes additionally break sender–recipient linkability by commingling balances in a shared encrypted pool over the encrypted-token program, so that a deposit and a later spend cannot be linked on chain.

Confidential swaps. A swap routes an encrypted input amount through the Jupiter aggregator for liquidity while keeping the traded amounts confidential: the amounts in and out are hidden, whereas the token pair and the routing venue are public.

Cross-chain bridge. The bridge moves value between Solana and external chains while keeping the transferred amounts confidential. Settlement itself is delegated to an external cross-chain protocol (NEAR Intents [36, 37]), whose mechanics are outside our scope; what matters for our trust model is that this imports an assumption *external* to Encifher's own TEE/threshold design. The safety of cross-chain settlement rests on that protocol's own signer set and contracts, so a

Table 5: Confidential-computation approaches for blockchains. ✓=yes, −=no; "general" means arbitrary programmable computation over shared encrypted state, not only transfers.

System	Engine	Trust root	Amount	Graph	General	Mainnet
Encifher (this work)	TEE+thresh.	HW+thresh.	✓	✓	✓	Solana
Zama fhEVM [6]	FHE	cryptographic	✓	–	✓	EVM
Arcium [8]	MPC	non-collusion	✓	–	✓	Solana (alpha)
Secret Network [10]	TEE	hardware	✓	✓	✓	own L1
Token-2022 CT [4]	ZK	cryptographic	✓	–	–	disabled

compromise there could missettle funds regardless of Encifher's confidentiality guarantees. We therefore treat the bridge's settlement layer as a separate trust domain from the coprocessor (§3); Encifher's role is confined to keeping the bridged amounts confidential on the Solana side.

Why these are a good fit. Each application is dominated by simple arithmetic over confidential balances (add, subtract, compare, conditional select) chained across transactions, precisely the workload for which symbolic execution plus an in-enclave evaluator is efficient, and for which an FHE backend would pay the largest relative penalty. The applications also exercise the per-handle account layout of §4.9: independent users' confidential operations touch disjoint handle accounts and so execute in parallel, which is what allows the confidentiality layer to inherit Solana's throughput rather than serialize behind it.

9 Related Work

We organize related work by the mechanism used to achieve confidentiality, and close by placing Encifher precisely. A useful axis throughout is the distinction between *transaction privacy*, hiding a single user's own inputs over a commitment set, and *confidential computation over shared encrypted state*, which multiple mutually distrusting parties read and write. Encifher, like the TEE/FHE/MPC systems below, targets the latter.

TEE-based confidential smart contracts. The use of trusted hardware to execute contracts confidentially originates with Town Crier [38] and, most influentially, Ekiden [18], which separates consensus from confidential execution: enclaves run contracts over encrypted state and settle an attested signature that consensus nodes verify without trusted hardware. Encifher inherits this execution model. Subsequent systems explored single-operator (FastKitten [39]), pooled and fault-tolerant (POSE [40]), and publicly verifiable multi-party (Cloak [41]) variants. In production, Secret Network [10] and Oasis Sapphire [42] (the commercial descendant of Ekiden) offer confidential smart contracts, and TEN [43] and Phala [44] apply TEEs to L2 confidentiality and decentralized off-chain compute. Two lessons from this lineage are central to Encifher's design. First, a single enclave holding a network-wide master key is a catastrophic single point of failure: such a key was extracted from a deployed system, which would have permitted retroactive decryption of its entire confidential history [9, 10]; Encifher's threshold committee (§4.7) is a direct response. Second, systems whose confidentiality degrades gracefully, reducing to the underlying integrity guarantee if the TEE fails, as in MobileCoin [45] and TEN [43], are more robust than those that collapse.

FHE coprocessors. FHE offers confidentiality from cryptographic hardness alone, and FHE coprocessors [6, 46, 47] have made it programmable on the EVM, introducing exactly the ciphertext-

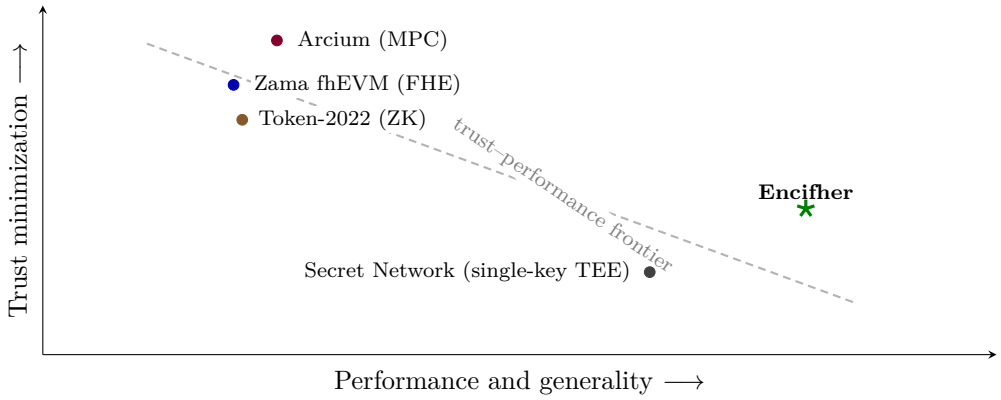

Figure 10: Confidential-computation approaches on a trust-versus-performance frontier (§6.2 quantifies the performance axis). Cryptographic engines (FHE, MPC) minimize trust but pay in performance and, for Token-2022, in generality; single-key TEEs are fast but concentrate trust in one enclave. Encifher takes the TEE performance point yet pushes back up the trust axis with a threshold committee, rather than dominating either extreme.

handle and symbolic-execution interface that Encifher adopts, together with threshold decryption. Encifher's relationship to this line is its defining design choice: we keep the interface but replace the FHE evaluator with a TEE. The motivation is performance: even the fastest gate- bootstrapping schemes [7] leave homomorphic operations several orders of magnitude slower than plaintext [48], which limits deployed FHE coprocessors to tens of transactions per second on CPU [6] (with a 2026 GPU roadmap targeting 500–1,000 TPS per chain). The cost is a change in trust root, from cryptographic to hardware-plus-threshold. Notably, the FHE coprocessors' own key-management services already run inside TEEs as defense-in- depth [6], so the two approaches meet at the trust boundary; Encifher moves the evaluator there as well.

MPC. MPC removes any single trusted party by secret-sharing state, from Enigma [49] to Arcium [8], the most prominent confidential- computing network on Solana. Arcium is Encifher's closest peer in goal and host: both push confidential computation off chain and use Solana for coordination and settlement, differing in the engine. The trade-off is clean. Arcium's MPC offers a stronger trust model, no single trusted party, with confidentiality holding as long as one participant is honest in its dishonest-majority protocol, at the cost of inter-node communication per operation; Encifher's TEE offers near-native, communication-free evaluation at the cost of a hardware trust assumption. We regard these as different points on a trust-versus-performance frontier rather than one dominating the other, and §6.2 quantifies the performance side.

Native Solana confidentiality. Solana's Token-2022 confidential-transfer extension [4] uses twisted-ElGamal encryption with zero-knowledge range and validity proofs, verified by a native runtime program, to hide token *amounts*. It is the natural baseline for the transfer operation, but it differs from Encifher in scope and status: it supports no general computation (only transfers), does not hide the transaction graph, and was disabled on mainnet following a soundness vulnerability in its proof program [5]. Light Protocol's ZK compression [50] is sometimes grouped with privacy work but is a *state-compression* primitive that makes public state cheaper, not a confidentiality mechanism. Elusiv [51] provided shielded transfers on Solana before sunsetting; its team's subsequent move to MPC (Arcium) mirrors the broader trajectory of the field. A recent system-

atization of Solana's zero-knowledge ecosystem [52] corroborates that confidential *computation* on Solana lands in the off-chain coprocessor quadrant that Encifher occupies.

ZK transaction privacy. Shielded-payment and mixer systems provide strong, cryptographically backed transaction privacy over commitment sets, and Hawk [53] and zkay [54] formalize and program privacy-preserving contracts. These are complementary to Encifher rather than competing: they hide a user's own inputs and provide unlinkability, but do not natively compute over state that is secret to all parties at once. Encifher could, in principle, adopt such techniques to address the metadata leakage it does not currently hide (§3.6).

The coprocessor abstraction. Finally, Encifher is an instance of the broader "execute off chain, verify on chain" paradigm whose integrity-oriented (ZK and fraud-proof) forms trace to Pinocchio [55], Truebit [56], Arbitrum [57], and ACE [58]. Encifher is the confidentiality-oriented counterpart on Solana: where a ZK coprocessor proves the correctness of public computation, Encifher keeps the inputs secret and anchors correctness in attestation and DA-backed Merkle commitments instead of a validity proof.

10 Discussion, Limitations, and Future Work

We state Encifher's limitations plainly; several are consequences of the design point it occupies, and naming them is part of an honest account of a deployed system.

Single-vendor and single-platform hardware trust. A deployment whose committee enclaves all run on one TEE platform and one cloud provider concentrates hardware trust. Spreading the committee across distinct TEE platforms (e.g., Intel TDX and AMD SEV-SNP) and operators would make the non-collusion assumption of §3 rest on heterogeneous hardware and independent operators, so that a single vendor-level flaw or a single operator does not compromise a threshold. We regard multi-platform committees as the most important hardening step beyond the present design.

Metadata privacy. Encifher hides values but not the transaction graph (§3.6). Applications needing sender/receiver unlinkability must combine Encifher with complementary techniques; integrating shielded-pool or mixing constructions over the confidential-token layer is future work.

Residual TEE risk. As §7 discusses, enclave confidentiality is a strong but unproven assumption, and the threshold committee bounds, rather than eliminates, the consequence of a breach. Continued monitoring of the TEE side-channel literature, prompt patching, and the multi-platform committees above are the available mitigations; we do not claim a TEE-based system can offer the unconditional confidentiality of a (currently far slower) FHE or MPC design.

11 Conclusion

Confidentiality has been a missing primitive on Solana, despite its suitability as a high-throughput settlement layer, because the cryptographic approaches that provide it are either too slow (FHE), communication-bound (MPC), or limited to transaction privacy (ZK). Encifher closes this gap with a confidentiality coprocessor that adopts the ergonomic ciphertext-handle and symbolic-execution

interface of FHE coprocessors but resolves computation inside a TEE, achieving near-native performance and supporting arbitrary computation over encrypted, shared state. The design is shaped by, and exploits, Solana's substrate: ciphertexts are kept off chain behind 128-bit handles because Solana's compute, size, and rent constraints demand it, and per-handle program-derived accounts make the Sealevel scheduler execute confidential operations in parallel and in dependency order without bespoke coordination. Encifher avoids the catastrophic single-enclave- master-key failure mode through a threshold-decryption committee and anchors integrity on chain through attestation and DA-backed Merkle commitments. It is deployed in production for confidential payments, swaps, and a bridge. Encifher does not claim trustlessness; it offers a different, and we argue practical, point on the trust-versus-performance frontier for confidential computation on a high-throughput chain, one that, uniquely among its peers, is running in production today.

References

[1] Anatoly Yakovenko. Solana: A new architecture for a high performance blockchain (proof of history). `https://solana.com/news/proof-of-history`, 2017. Accessed 2026-06.

[2] Solana Labs. Sealevel: Parallel processing thousands of smart contracts. `https://solana.com/news/sealevel,parallel-processing-thousands-of-smart-contracts`, 2019. Accessed 2026-06.

[3] Solana Foundation. Solana documentation: Accounts, programs, compute budget, fees. `https://solana.com/docs/core`, 2025. Accessed 2026-06.

[4] Solana Foundation. Token-2022 confidential transfer extension; confidential balances. `https://solana.com/docs/tokens/extensions/confidential-transfer`, `https://www.solana-program.com/docs/confidential-balances/zkps`, 2024. Accessed 2026-06.

[5] Solana Foundation. Post-mortem: ZK ElGamal Proof Program soundness issue and disablement. `https://solana.com/news/post-mortem-june-25-2025`, 2025. Accessed 2026-06.

[6] Zama. Introducing the fhEVM coprocessor; Zama protocol litepaper. `https://www.zama.org/post/fhevm-coprocessor`, `https://docs.zama.org/protocol/zama-protocol-litepaper`, 2024. Accessed 2026-06.

[7] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. TFHE: Fast fully homomorphic encryption over the torus. *Journal of Cryptology*, 33(1), 2020.

[8] Arcium. Arcium: The encrypted supercomputer, architecture overview. `https://docs.arcium.com/getting-started/architecture-overview`, 2026. Mainnet Alpha (Feb 2026); full mainnet / token TGE pending mid-2026. Accessed 2026-06.

[9] Stephan Van Schaik, Alex Seto, Thomas Yurek, Adam Batori, Bader AlBassam, Christina Garman, Daniel Genkin, Andrew Miller, Eyal Ronen, and Yuval Yarom. SoK: SGX.Fail: How stuff gets exposed. In *IEEE Symposium on Security and Privacy (S&P)*, 2024.

[10] Secret Network. Secret network graypaper. `https://scrt.network/graypaper`, 2020. Accessed 2026-06.

[11] Anza. Alpenglow: A new consensus protocol for Solana. `https://www.helius.dev/blog/alpenglow`, 2026. Live on test cluster May 2026; mainnet expected late 2026. Accessed 2026-06.

[12] Alexander Nilsson, Pegah Nikbakht Bideh, and Joakim Brorsson. A survey of published attacks on Intel SGX, 2020. arXiv:2006.13598.

[13] Pau-Chen Cheng, Wojciech Ozga, Enriquillo Valdez, Salman Ahmed, Zhongshu Gu, Hani Jamjoom, Hubertus Franke, and James Bottomley. Intel TDX demystified: A top-down approach. *ACM Computing Surveys*, 56(9), 2024. arXiv:2303.15540.

[14] Shunfan Zhou, Kevin Wang, and Hang Yin. dStack: A zero trust framework for confidential containers, 2025. arXiv:2509.11555.

[15] Phala Network. TEEs secured by Ethereum: A decentralized root of trust for confidential compute. `https://phala.com/posts/TEEs-Secured-by-Ethereum`, 2025. Accessed 2026-06.

[16] Intel Corporation. Intel TDX DCAP: Quoting library API. `https://download.01.org/intel-sgx/latest/dcap-latest/linux/docs/Intel_TDX_DCAP_Quoting_Library_API.pdf`, 2023. Accessed 2026-06.

[17] Solana Foundation. SIMD-0075: Precompile for secp256r1 signature verification; precompiled programs. `https://github.com/solana-foundation/solana-improvement-documents/blob/main/proposals/0075-precompile-for-secp256r1-sigverify.md`, 2025. Activated on Solana mainnet 2025. Accessed 2026-06.

[18] Raymond Cheng, Fan Zhang, Jernej Kos, Warren He, Nicholas Hynes, Noah Johnson, Ari Juels, Andrew Miller, and Dawn Song. Ekiden: A platform for confidentiality-preserving, trustworthy, and performant smart contracts. In *IEEE European Symposium on Security and Privacy (EuroS&P)*, 2019. arXiv:1804.05141.

[19] Morris Dworkin. Recommendation for block cipher modes of operation: Galois/Counter Mode (GCM) and GMAC. Technical Report NIST SP 800-38D, National Institute of Standards and Technology, 2007.

[20] Certicom Research. SEC 1: Elliptic curve cryptography, version 2.0. Technical report, Standards for Efficient Cryptography Group (SECG), 2009. Defines ECIES and SEC 1 point encoding. `https://www.secg.org/sec1-v2.pdf`.

[21] Certicom Research. SEC 2: Recommended elliptic curve domain parameters, version 2.0. Technical report, Standards for Efficient Cryptography Group (SECG), 2010. Defines secp256k1: prime order n, cofactor $h = 1$. `https://www.secg.org/sec2-v2.pdf`.

[22] Luca Wilke, Florian Sieck, and Thomas Eisenbarth. TDXdown: Single-stepping and instruction counting attacks against Intel TDX. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 79–93, 2024. doi: 10.1145/3658644.3690230. CVE-2024-27457.

[23] Fabian Rauscher, Luca Wilke, Hannes Weissteiner, Thomas Eisenbarth, and Daniel Gruss. TDXploit: Novel techniques for single-stepping and cache attacks on Intel TDX. In *USENIX Security Symposium*, 2025. Distinguished Paper (Hon. Mention). CVE-2025-20613.

[24] Rosario Gennaro, Stanisław Jarecki, Hugo Krawczyk, and Tal Rabin. Secure distributed key generation for discrete-log based cryptosystems. *Journal of Cryptology*, 20(1):51–83, 2007. Conference version: EUROCRYPT 1999.

[25] Torben Pryds Pedersen. A threshold cryptosystem without a trusted party. In *Advances in Cryptology, EUROCRYPT '91*, volume 547 of *LNCS*, pages 522–526. Springer, 1991.

[26] Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[27] Phala Network. Confidential computing on NVIDIA H100 GPU: A performance benchmark study. `https://phala.network/posts/confidential-computing-on-nvidia-h100-gpu-a-performance-benchmark-study`, 2024. Accessed 2026-06.

[28] Jo Van Bulck, Marina Minkin, Ofir Weisse, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Thomas F. Wenisch, Yuval Yarom, and Raoul Strackx. Foreshadow: Extracting the keys to the Intel SGX kingdom with transient out-of-order execution. In *USENIX Security Symposium*, 2018.

[29] Stephan Van Schaik, Andrew Kwong, Daniel Genkin, and Yuval Yarom. CacheOut/SGAxe: Leaking data and attestation keys from Intel SGX. In *IEEE Symposium on Security and Privacy (S&P)*, 2021.

[30] Pietro Borrello, Andreas Kogler, Martin Schwarzl, Moritz Lipp, Daniel Gruss, and Michael Schwarz. ÆPIC Leak: Architecturally leaking uninitialized data from the microarchitecture. In *USENIX Security Symposium*, 2022.

[31] Mengyuan Li, Yinqian Zhang, Huibo Wang, Kang Li, and Yueqiang Cheng. CipherLeaks: Breaking constant-time cryptography on AMD SEV via the ciphertext side channel. In *USENIX Security Symposium*, 2021.

[32] Jesse De Meulemeester et al. BadRAM: Practical memory aliasing attacks on AMD SEV-SNP. In *IEEE Symposium on Security and Privacy (S&P)*, 2025.

[33] Google Cloud Security and Intel. Intel Trust Domain Extensions (TDX) security review. Technical report, Google, 2023. Follow-on TDX 1.5 review, 2025. `https://services.google.com/fh/files/misc/intel_tdx_-_full_report_041423.pdf`.

[34] Jalen Chuang, Alex Seto, Nicolas Berrios, Stephan van Schaik, Christina Garman, and Daniel Genkin. TEE.fail: Breaking trusted execution environments via DDR5 memory bus interposition. In *IEEE Symposium on Security and Privacy (S&P)*, 2026. To appear. `https://tee.fail/`.

[35] Andrew Miller. TEE-based smart contracts and sealing pitfalls. `https://medium.com/initc3org/tee-based-smart-contracts-and-sealing-pitfalls-eccd5d751329`, 2022. IC3. Accessed 2026-06.

[36] NEAR Protocol. NEAR Intents: Intent-based multichain settlement. `https://docs.near.org/chain-abstraction/intents/overview`, 2025. Accessed 2026-06.

[37] NEAR Protocol. Chain signatures: MPC-based multichain signing. `https://docs.near.org/chain-abstraction/chain-signatures`, 2025. Accessed 2026-06.

[38] Fan Zhang, Ethan Cecchetti, Kyle Croman, Ari Juels, and Elaine Shi. Town crier: An authenticated data feed for smart contracts. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2016.

[39] Poulami Das, Lisa Eckey, Tommaso Frassetto, David Gens, Kristina Hostáková, Patrick Jauernig, Sebastian Faust, and Ahmad-Reza Sadeghi. FastKitten: Practical smart contracts on Bitcoin. In *USENIX Security Symposium*, 2019.

[40] Tommaso Frassetto, Patrick Jauernig, David Koisser, David Kretzler, Benjamin Schlosser, Sebastian Faust, and Ahmad-Reza Sadeghi. POSE: Practical off-chain smart contract execution. In *Network and Distributed System Security Symposium (NDSS)*, 2023. arXiv:2210.07110.

[41] Qian Ren, Yue Li, Yingjun Wu, Yuanit Wu, Hong Lei, Lei Wang, and Bangdao Chen. Cloak: Transitioning states on legacy blockchains using secure and publicly verifiable off-chain multiparty computation. In *Annual Computer Security Applications Conference (ACSAC)*, 2022. arXiv:2106.13926.

[42] Oasis Protocol Foundation. Sapphire: A confidential EVM ParaTime. `https://oasis.net/sapphire`, 2023. Accessed 2026-06.

[43] TEN (formerly Obscuro). TEN whitepaper. `https://whitepaper.ten.xyz`, 2024. Accessed 2026-06.

[44] Phala Network. Phala Network documentation; dStack: A zero-trust framework for confidential containers. `https://docs.phala.com`, 2025. arXiv:2509.11555. Accessed 2026-06.

[45] MobileCoin. MobileCoin: Private payments. `https://github.com/mobilecoinfoundation/mobilecoin`, 2021. Accessed 2026-06.

[46] Fhenix. CoFHE: A coprocessor for fully homomorphic encryption on EVM chains. `https://cofhe-docs.fhenix.zone`, 2025. Accessed 2026-06.

[47] Inco Network. Inco: A modular confidential computing network. `https://www.inco.org`, 2025. Accessed 2026-06.

[48] Shruthi Gorantala, Rob Springer, and Bryant Gipson. Unlocking the potential of fully homomorphic encryption. *Communications of the ACM*, 66(5):72–81, 2023. doi: 10.1145/3572832.

[49] Guy Zyskind, Oz Nathan, and Alex Pentland. Enigma: Decentralized computation platform with guaranteed privacy, 2015. arXiv:1506.03471.

[50] Light Protocol. ZK compression whitepaper. `https://www.zkcompression.com/references/whitepaper`, 2024. Accessed 2026-06.

[51] Elusiv. Sunsetting elusiv: Transitioning towards the future of privacy. `https://medium.com/elusiv-privacy`, 2024. Accessed 2026-06.

[52] Jotaro Yano. Zero-knowledge extensions on Solana: A theory of ZK architecture, 2025. arXiv:2511.00415.

[53] Ahmed Kosba, Andrew Miller, Elaine Shi, Zikai Wen, and Charalampos Papamanthou. Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In *IEEE Symposium on Security and Privacy (S&P)*, 2016.

[54] Samuel Steffen, Benjamin Bichsel, Mario Gersbach, Noa Melchior, Petar Tsankov, and Martin Vechev. zkay: Specifying and enforcing data privacy in smart contracts. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2019.

[55] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. Pinocchio: Nearly practical verifiable computation. In *IEEE Symposium on Security and Privacy (S&P)*, 2013.

[56] Jason Teutsch and Christian Reitwießner. A scalable verification solution for blockchains, 2017. arXiv:1908.04756.

[57] Harry Kalodner, Steven Goldfeder, Xiaoqi Chen, S. Matthew Weinberg, and Edward W. Felten. Arbitrum: Scalable, private smart contracts. In *USENIX Security Symposium*, 2018.

[58] Karl Wüst, Sinisa Matetic, Silvan Egli, Kari Kostiainen, and Srdjan Capkun. ACE: Asynchronous and concurrent execution of complex smart contracts. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2020.