IADIS International Journal on Computer Science and Information Systems
Vol. 21, No. 2, pp. 33-46
ISSN: 1646-3692

HARDWARE AS THE ARBITRATOR: AUTONOMOUS ESCROW LOGIC FOR LEDGERS WITHOUT SMART CONTRACTS

Luka Skeledžija
Technical University of Munich, Arcisstraße 2, 80333 Munich, Germany

ABSTRACT

This paper presents an approach to extending trustless, autonomous logic to ledgers without smart-contract support, using hardware-based Trusted Execution Environments (TEEs). We demonstrate this by upgrading an intent-based token swap protocol to facilitate secure, peer-to-peer transfers between an EVM chain and the XRP Ledger (XRPL). We first make explicit what the destination side of such a protocol has to provide, describing the escrow as a release condition over the preimage, the ledger time, and the party invoking it. Native conditional escrow primitives, including XRPL's, realize the first two arguments; the third carries the protocol's incentive structure. By running a hash-locked escrow facilitator inside a TEE, we are able to evaluate the full condition off-chain while preserving the order format, hashlock and timelock semantics of the source chain, guaranteeing code integrity and confidentiality for sensitive assets such as private keys. This approach incurs minimal performance overhead, unlike approaches such as Fully Homomorphic Encryption (FHE), which remain computationally infeasible for this class of operations. We situate the design within the literature on atomic swaps, intent-based protocols and trusted hardware, and analyze the critical limitation of relying on a single hardware root of trust, which leaves systems vulnerable to microarchitectural exploits.

KEYWORDS

Trusted Execution Environment (TEE), Confidential Computing, Intent-Based Swap, Cross-Chain Interoperability, Trustless Protocol, Autonomous Logic

1. INTRODUCTION

In 2019, a group of major technology companies (including Google, Microsoft, IBM, Intel, and others) founded the Confidential Computing Consortium under the Linux Foundation's umbrella. The consortium's objective is to accelerate the adoption of confidential computing and establish universal security standards for protecting data in use (Mithril Security, 2023). This technology secures data during computation by performing operations within a

hardware-based Trusted Execution Environment (TEE). As a high-security engineering solution, it offers minimal performance overhead compared to other privacy-preserving techniques, such as Fully Homomorphic Encryption (FHE). While FHE is cryptographically robust, existing implementations remain orders of magnitude more computationally expensive, making them currently infeasible for large-scale data processing.

Protecting user and corporate data is increasingly critical from ethical, regulatory, and economic perspectives. The financial incentive is particularly strong due to recent advancements in AI, which have made high-quality data more valuable than ever. TEEs offer several novel use cases: (1) confidential AI, where proprietary models can be trained on private datasets or perform secure inference without the infrastructure provider ever seeing the raw input or the model's weights (protecting intellectual property), (2) medical research, where researchers and platforms can collaborate on sensitive medical data to find cures or improve diagnostics while maintaining absolute patient anonymity through hardware-provided security, (3) government and sovereignty, where nations can migrate critical infrastructure and citizen records to the cloud while ensuring data remains under their exclusive control, shielded even from the cloud service providers themselves.

The development of this hardware is well underway across the industry. Major chip designers have introduced dedicated solutions, including Intel's SGX and TDX, AMD's SEV-SNP, NVIDIA's Hopper & Blackwell Confidential Computing, and ARM's TrustZone & CCA (Mithril Security, 2023).

Recently, interesting use cases for TEEs have emerged in the world of trustless, decentralized protocols. One such use case is discussed in this paper, which describes an intent-based token swap protocol that uses a TEE-enabled machine as one of its facilitators.

2. BACKGROUND AND RELATED WORK

2.1 Atomic Cross-Chain Swaps

The formal treatment of cross-chain swaps begins with Herlihy (2018), which models a swap as a directed graph of asset transfers and establishes when a protocol can guarantee that either all transfers occur or none do. The standard realization is the hash-time-locked contract (HTLC). A payer locks funds under a hash h, and the payee claims them by presenting a preimage s satisfying $H(s) = h$ before a deadline, after which the payer may reclaim them. HTLCs are atomic but not incentive-neutral. Xu et al. (2021) show game-theoretically that the party holding the secret possesses a free option, observing price movement during the timelock and choosing whether to complete while the counterparty bears the downside. The safety deposit used by the protocol studied here belongs to the same lineage of partial mitigations.

2.2 Intent-Based Protocols

Intent-based protocols replace user-executed transactions with signed declarations of a desired outcome, filled competitively by professional agents variously called solvers, fillers or resolvers. Empirical work reports measurable price improvement in such order-flow auctions while noting that they run on largely opaque backends (Bachu et al., 2024), which is itself part of the case for verifiable execution. Augusto et al. (2026) study 3.5 million cross-chain intents

representing $9.24 billion in volume and describe liquidity exhaustion attacks, which drain the working capital solvers front on the destination side. The protocol we build on is described in a whitepaper by Bukov et al. (2024), which remains the primary source for the mechanics described in Section 3.

2.3 Trusted Execution in Blockchain Systems

Trusted hardware has been proposed repeatedly as a trust-minimizing component in blockchain protocols. TrustCross (Lan et al., 2021) uses it for confidential cross-chain interoperability, and MERCURY (Wen et al., 2024) builds a complete cross-chain exchange on trusted hardware, removing both on-chain verification cost and the requirement that clients remain online. Our design differs in aim rather than in primitive, as Section 6.2 sets out.

The security literature is appropriately skeptical of the TEE primitive. Cloosters et al. (2020) demonstrate memory-corruption vulnerabilities in SGX enclaves, and Jean-Louis et al. (2024) find privacy flaws in TEE-based smart contract platforms that do not require breaking the TEE itself. Wilde et al. (2024) systematize twenty-nine TEE-and-consensus proposals and evaluate their resistance to forking and rollback, the relevant attack class for any escrow that keeps state across a restart. We return to these limitations later on.

3. INTENT-BASED SWAP PROTOCOL

For the purposes of this paper, we were building on top of 1inch's intent-based swap protocol, Fusion+. The protocol enables a trustless peer-to-peer swap of tokens A and B between two EVM-supported chains at a pre-determined price discovered by auction. The full protocol description is available in the whitepaper (Bukov et al., 2024) and shown schematically in Figure 1. In the following paragraphs, we will go over the most important parts for understanding how the swaps are made and where the critical security assumptions come from.

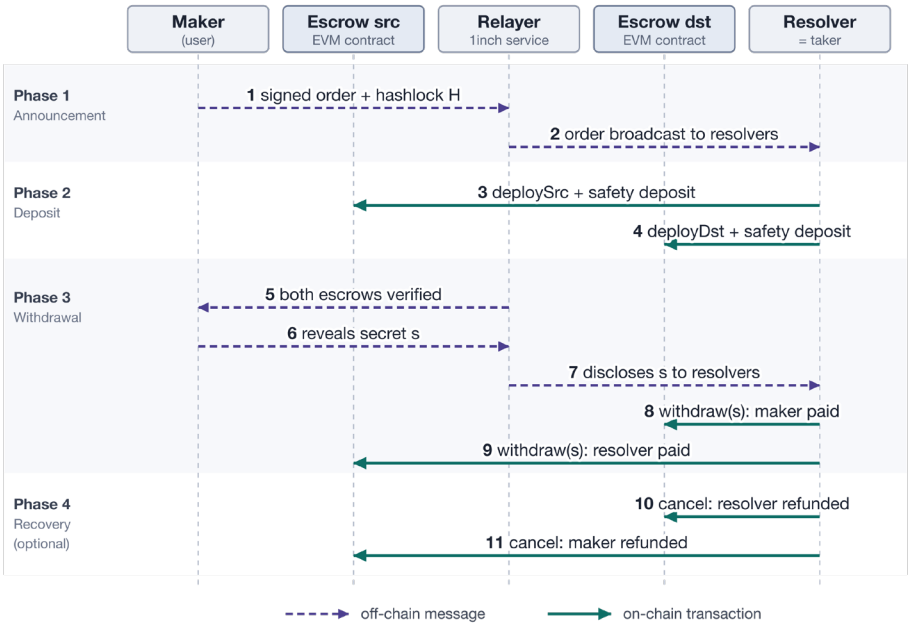


Figure 1. Schematic view of the four-phase 1inch Fusion+ protocol, after Bukov et al. (2024). Dashed arrows are off-chain messages; solid arrows are on-chain transactions. The resolver, and not the maker, submits every on-chain transaction on both chains

The main idea behind the protocol is to make deposits on both chains to a hash-locked escrow smart contract. After verifying that the deposits were made, it is safe to reveal the secret used to compute the hash and withdraw tokens to the opposite addresses (maker's deposit to the taker's address on the origin chain, and vice versa for the target chain). In Phase 1, the maker signs a Fusion order, essentially authorizing the deposit of their funds into escrow. In Phase 2, deposits are made to the escrow contracts on both chains. After the maker verifies that the deposits were made correctly, they can reveal the secret, and the taker will withdraw funds for both parties on the two chains. If no secret is revealed or an escrow is missing a deposit, the protocol includes a recovery phase that will cancel the order and return funds to their original owners.

3.1 Roles and Incentives

A simplified description could set the counterparty and relayer roles aside. However, they are worth stating properly, because the incentive layer they carry is what the destination side has to reproduce to make this work in practice.

Three roles participate. The **maker** is the end user: they sign an off-chain order and a secret hash and later reveal the secret, but they never have to submit a transaction on either chain. The **resolver** is the maker's counterparty, occupying the taker position of the underlying limit-order protocol. The whitepaper uses the two terms interchangeably (Bukov et al., 2024). It creates both escrows, fronts its own capital on the destination chain, pays gas on both chains,

and executes both withdrawals, including the one that pays the maker. Resolvers are professional firms admitted after KYC/KYB checks and identified on-chain by an access token the escrow contracts require (1inch, 2026). The **relayer** is a 1inch-operated off-chain service that broadcasts orders, verifies both escrows, and only then discloses the secret. Since it never holds funds nor signs a chain transaction, the implementation of Section 5 can omit it without weakening the security argument.

Resolvers are selected by a descending-price auction encoded in the signed order. The price begins above the prevailing market rate and falls in piecewise-linear segments toward the maker's stated minimum return. Resolvers compete to fill "as soon as it becomes profitable for them, otherwise they risk losing the profits to another resolver" (Bukov et al., 2024), so the fill time reveals the marginal resolver's cost.

The settlement itself is secured by two mechanisms. First, a resolver creating an escrow attaches a **safety deposit** in the chain's native asset. The deposit is paid to whoever executes the eventual withdrawal or cancellation (Bukov et al., 2024). It is a bounty for liveness: a resolver that fails to act in time forfeits it to whoever acts instead. It ensures that a swap abandoned midway is always completed.

Second, for that bounty to be claimable, the timeline is partitioned by *who* may act and not only by *when*. Fusion+ divides the two chains into seven stages (Figure 2). On each chain a finality lock runs first, during which no withdrawal is permitted; then an exclusive window in which only the resolver that created the escrow may withdraw; then a public window in which any other admitted resolver may complete the swap and claim the deposit. Cancellation mirrors this private-then-public structure (Bukov et al., 2024).

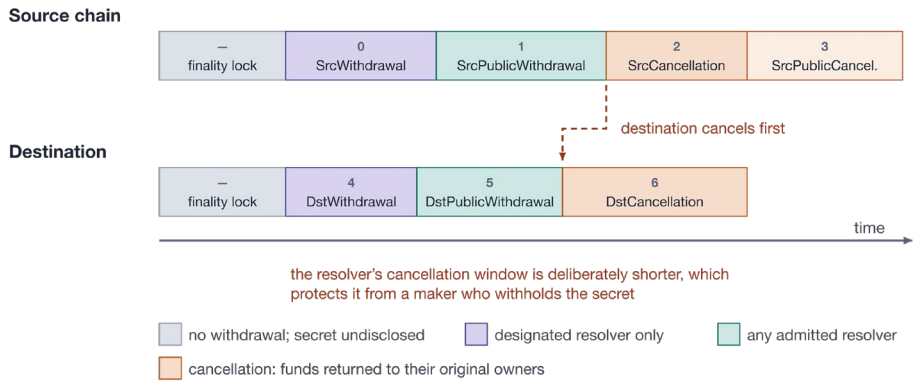


Figure 2. The seven-stage timelock ladder, source chain above and destination chain below. Shading distinguishes windows by permitted caller: finality lock, exclusive to the executing resolver, and public. The transition from the exclusive to the public window is what makes the safety deposit contestable

3.2 The Release Condition

It is worth talking about the release conditions, because they decide what kind of mechanisms the facilitator needs to support.

Let the destination escrow evaluate a release condition

$$R(s, t, c) \in \{\text{release}, \text{refund}, \perp\}$$

on the preimage s presented, the ledger time t, and the identity c of the party presenting it. The symbol $\perp$ denotes that no action is permitted and the escrow is left unchanged. A classical HTLC uses the first two arguments only,

$$R_{\text{HTLC}}(s,t) = \text{release} \Leftrightarrow (H(s) = h) \ \wedge \ (t < T),$$

with a refund available to the depositor once $t \geq T$. The caller does not enter the condition, because the funds move to an address fixed in advance: any party holding the preimage may trigger the release, and the payment still reaches the intended recipient.

The protocol of Section 3.1 conditions additionally on the caller. Writing $W(t)$ for the window in which time t falls, and $A(W)$ for the set of parties permitted to act during that window,

$$R_{\text{F+}}(s,t,c) = \text{release} \Leftrightarrow (\text{keccak256}(s) = h) \ \wedge \ (W(t) \in \Omega) \wedge \ (c \in A(W(t))),$$

where Ω is the set of windows in which a withdrawal may be made at all, namely the exclusive and the public withdrawal windows. The second conjunct therefore rules out the finality lock, during which nothing may be withdrawn, and the two cancellation windows, in which funds return to their depositor instead of moving to the maker. The third conjunct fixes who may act: A is empty during the finality lock, contains only the resolver that created the escrow during the exclusive window, and contains every admitted resolver during the public one. The third argument is important for the incentive structure described above. Without it there is no way to reserve a first attempt for the resolver whose capital is at risk, and the safety deposit has nothing to attach to.

Native escrow primitives generally realize R_{HTLC}. The XRP Ledger, for instance, provides genuine on-chain conditional escrow: an `EscrowCreate` may carry a `Condition` field holding a PREIMAGE-SHA-256 crypto-condition together with `FinishAfter` and `CancelAfter` timestamps, and the TokenEscrow amendment enabled in February 2026 extends this to issued tokens as well as XRP (XRP Ledger, 2026). This is a faithful realization of R_{HTLC}. What such primitives do not express is the caller argument, along with the keccak256 hash used by the source chain and a native-asset bounty payable to whoever executes. Evaluating the full condition therefore calls for a facilitator on the destination side, and what we propose and demonstrate below is how to use TEEs to act as that facilitator.

4. KEY PROPERTIES OF THE TEE

In confidential computing, a Trusted Execution Environment (TEE) is defined as a "Tamper-resistant processing environment that runs on a separation kernel. It guarantees the authenticity of the executed code, the integrity of the runtime states (e.g., CPU registers, memory, and sensitive I/O), and the confidentiality of its code, data, and runtime states stored on a persistent memory. In addition, it shall be able to provide remote attestation that proves its trustworthiness for third-parties. The content of TEE is not static; it can be securely updated. The TEE resists all software attacks and the physical attacks performed on the system's main memory. Attacks performed by exploiting backdoor security flaws are impossible" (Sabt et al., 2015).

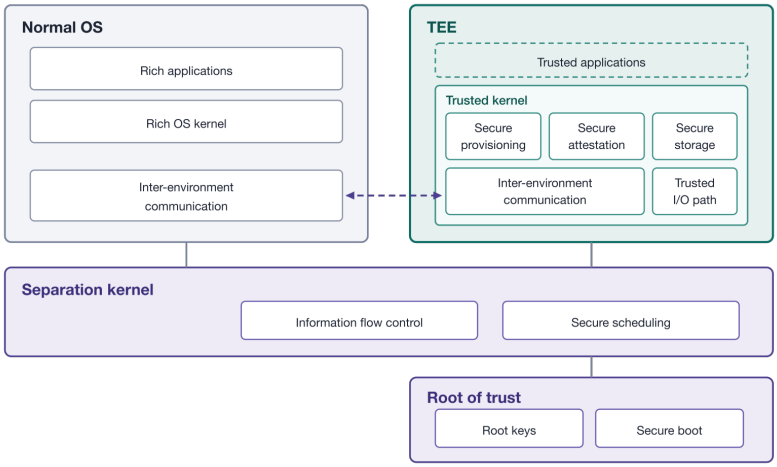


Figure 3. TEE alongside a normal runtime, after Sabt et al. (2015). Isolation is enforced by the separation kernel and anchored in a hardware root of trust

In practical terms, this means that TEEs are designed to defend against an attacker with OS kernel-level access (Figure 3). Due to the hardware isolation of these TEEs, an attacker should not be able to tamper with or observe the execution of a running TEE program. Because these guarantees only apply to a running TEE, any restart of the TEE will cause data to be lost. TEEs therefore make use of sealed data to encrypt state on disk and securely load it into the TEE. Remote attestation allows the user to verify that they are running a legitimate TEE with the correct software version. At a high level, remote attestation is an interactive protocol in which a measurement of the data and software running in the enclave is taken and signed using a secret key hidden in the hardware. This measurement and signature are checked and endorsed by the hardware manufacturer based on its secret knowledge of the hardware keys determined during manufacturing. The root of trust remains with the manufacturer, which is responsible for correctly manufacturing hardware and issuing certificates only for appropriately manufactured TEE-enabled CPUs. Some of these properties are crucial for the security of our autonomous intent swap resolver, and we will cover them in more detail below (Jean-Louis et al., 2024).

4.1 Confidentiality of Data

Since we will be performing token transfers on a public ledger, the autonomous resolver will need to hold many tokens on both chains. To make the transfers, a copy of the private key is required for transaction signing. This poses a significant risk to liquidity providers if the cloud provider (or one of its employees) becomes malicious. Running the resolver code in the TEE's encrypted memory alleviates this risk and shifts the root of trust to the hardware manufacturer.

4.2 Remote Attestation

The assumptions above hold only if we are talking to a true TEE, not an attacker pretending to be one. It is essential to verify authenticity using a procedure called remote attestation. The process works in the following way (Edera, n.d.):

1. **State measurement**: The TEE computes a measurement (or attestation) of its current state, covering application-code integrity (a cryptographic hash of the application code), configuration data (runtime input and output parameters and settings), TEE identity (specific hardware and firmware versions), and load-time measurements (verifying that the code was loaded correctly).
2. **Cryptographic signing**: The TEE signs the information from Step 1 using a hardware-embedded private key that cannot be extracted; therefore, the signature cannot be forged.
3. **Attestation delivery**: The signed attestation is sent to the verifier (any party that wants to verify its authenticity).
4. **Verification**: The verifier must then perform several checks, including signature verification (verifying against the public key obtained from the manufacturer), state comparison (i.e., checking the code hash against known-good reference values), and nonce verification (protecting against replay attacks).
5. **Secure channel establishment**: Once the attestation is verified, an encrypted channel can be established between the verifier and the TEE.

In the context of our XRPL swap protocol, the attestation verification is performed in *two* complementary ways. First, the Oasis Sapphire on-chain runtime verifies the ROFL node's attestation during registration, meaning only nodes with valid, up-to-date TEE measurements are admitted to the node pool and permitted to manage hot wallets. This ensures private key confidentiality. Second, the maker can independently verify the TEE's attestation before depositing funds: the enclave measurement carried in the node's registration is recorded on-chain and can be compared against the measurement published for the escrow application. This assures the maker that the expected code is running inside the TEE.

4.3 Code Integrity

In the world of smart contracts, code is law. It is important to know which code we will be running, because the actions often cannot be reversed. So, if we are sending tokens to one address, we need to be sure that we will get a payout on the other side. Even though ensuring code integrity is a part of the attestation process, it must be emphasized that this is the most important assumption that makes our protocol trustless. It is possible to mathematically verify that an input X produced an output Y (Edera, n.d.). Building on these security guarantees, we can now proceed to implement our protocol.

5. UPGRADING THE INTENT-BASED SWAP PROTOCOL

In our reference implementation, we will be performing a swap from an EVM chain to the XRP Ledger (XRPL). The whole idea behind this is to replace the smart contract side on XRPL with

a TEE-based implementation running on Oasis nodes. The resulting arrangement is shown in Figure 4: the resolver drives an unmodified 1inch resolver contract on the EVM side and the enclave-hosted escrow on the XRPL side.

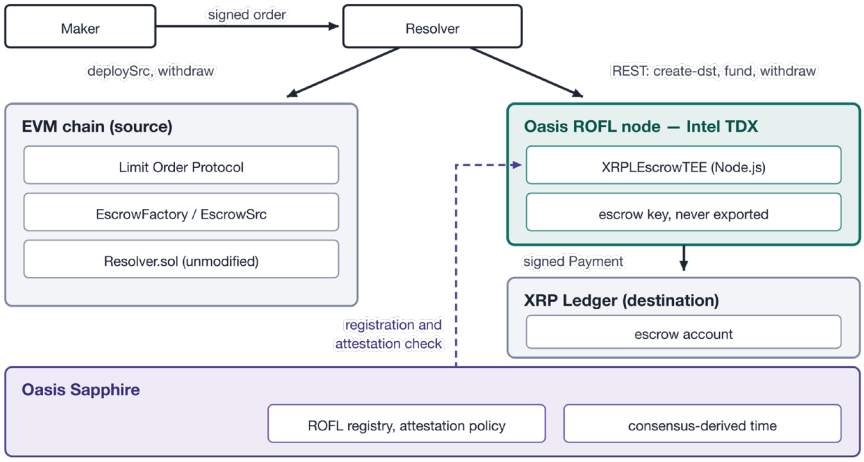


Figure 4. System architecture. The resolver drives both legs: an unmodified 1inch resolver contract on the EVM side, and a TEE-hosted escrow on the XRPL side whose enclave is registered and attested through Oasis Sapphire

5.1 Oasis Runtime Off-Chain Logic (ROFL)

For our hardware infrastructure, we chose Oasis Network's Runtime Off-Chain Logic (ROFL) nodes. ROFL-powered apps run inside TEEs on Oasis nodes and are managed through the Oasis Sapphire blockchain. ROFL supports (Oasis Protocol Foundation, 2026):

- Docker-like containers or single-executable apps
- Privacy and integrity through Intel SGX/TDX, including a fully auditable history of updates
- Uncensorable registration, management, and deployment on a permissionless pool of ROFL nodes, including billing
- Built-in Key Management Service (KMS) for storing secrets and securely deriving keys within TEEs

By doing this, we are enabling ourselves to build out our application layer as a Docker container and letting the Oasis protocol handle the deployment and attestation verification for us. The architectural consequence that matters here is that registration is itself an on-chain, policy-checked operation. The Sapphire runtime admits a node only if its attestation quote satisfies the application's published policy, so attestation is verified by the chain through its verifiers rather than by each counterparty separately.

5.2 Application Layer

Since we are now running inside a TEE and know our code cannot be tampered with, we are free to choose any programming language we want, not just Solidity. The reference implementation employs a Node.js environment executing within a TEE to provide the smart contract logic for the XRP Ledger (XRPL). The implementation can be found in our GitHub repository: https://github.com/lukaske/SwappaTEE.

5.2.1 Implementation Details

The *XRPLEscrowTEE* class acts as a trustless autonomous swap facilitator, mimicking EVM-style hash-locked escrow functionality for the XRP Ledger.

5.2.2 Cryptographic Identity and Initialization

- *generateEscrowWallet()*: Derives the XRPL account that holds the escrowed funds. The private key remains within the TEE's encrypted memory, ensuring that even a malicious host cannot intercept the key material. The design calls for a distinct account per swap instance, so that the balance backing one swap is never reachable by the release path of another.
- *initialize()*: Establishes the asynchronous WebSocket connection to the XRPL network, enabling the TEE to observe on-chain events and broadcast signed transactions.

5.2.3 Maintaining Compatibility with Solidity and the Fusion+ Protocol

To maintain parity with EVM-based smart contracts, the application must strictly enforce the same mathematical functions, data types, and temporal formatting of the Fusion+ protocol.

- *mykeccak256()*: Implements a Keccak-256 hashing algorithm identical to the Ethereum Virtual Machine (EVM) standard. This ensures that the hashlock generated on the source chain can be validated on the XRPL destination, which is the first conjunct of R_{F+}.
- *parseTimelocks()*: Decodes the 256-bit packed temporal data into a discrete schedule of seven protocol stages, ranging from initial withdrawal windows to final cancellation deadlines.
- *validateTimeWindow()*: Serves as the primary guardrail for state transitions. It verifies that the current time aligns with the authorized protocol phase before allowing any transaction to be signed, evaluating the second conjunct of R_{F+}. It is important to note that TEEs do not possess an independent hardware clock, so the choice of time source is itself a security property: the design specifies Oasis Sapphire's consensus-derived timestamps rather than the host operating system's clock. This prevents a malicious host from manipulating local system time to trigger premature cancellations or artificially delay withdrawal windows.

5.2.4 Swap Lifecycle

These endpoints manage the transfer of funds between the maker and taker, serving as a hash-locked escrow.

- *POST /escrow/create-dst*: In the “Announcement Phase,” this endpoint registers the swap intent, persists the hashlock, and calculates the required collateral (including the safety deposit).

- *POST /escrow/:escrowId/fund*: In the "Depositing Phase," the TEE auditably verifies on-chain payment transactions to ensure the taker has collateralized the escrow before the maker reveals the secret.
- *validateSecret()*: A critical security check that verifies the preimage (secret) against the stored hashlock. Only upon a successful match does the TEE authorize the release of funds.
- *POST /escrow/:escrowId/withdraw*: In the "Withdrawal Phase," the TEE validates the secret and temporal window, then autonomously signs and broadcasts a payment transaction to the maker's XRPL address.

5.2.5 Recovery Phase

The recovery phase is an optional phase of the protocol that ensures no deposited funds get locked and permanently lost should the swap fail. It activates after the rescue delay has passed.

- *POST /escrow/:escrowId/cancel*: Implements the "Recovery Phase". If the protocol conditions are not met within the allotted time, the TEE ensures funds are returned to their original owners, effectively acting as an automated refund mechanism.
- *POST /escrow/:escrowId/rescue*: A secondary safety fallback that permits the taker to reclaim "stuck" funds after an extended period (the rescueDelay), providing a final layer of economic security against unforeseen protocol halts.

By executing this logic within an Oasis ROFL node, the system ensures code integrity. This means the "code is law" principle, usually reserved for smart contracts, is extended to general-purpose Node.js logic through hardware-enforced attestation.

For demonstration purposes, the testnet implementation derives its escrow account from a fixed seed, reads the host clock, and holds escrow records in process memory. This would obviously need to be handled before this could become a production candidate. Jean-Louis et al. (2024) mention that TEE-based platforms are typically undermined through the interfaces a TEE leaves open rather than through the TEE itself.

A video demonstration is available on our YouTube channel: https://www.youtube.com/watch?v=d5ZZGaTBrWU

6. DISCUSSION

While TEEs offer a high-security engineering solution for protecting data in use, they are not exempt from hardware vulnerabilities. Historical exploits, such as side-channel attacks against Intel SGX (Cloosters et al., 2020), demonstrate that reliance on a single hardware manufacturer introduces a potential single point of failure. If a vulnerability is discovered in the microarchitecture of one vendor, the security guarantees regarding code integrity and confidentiality could be compromised.

To mitigate these risks and produce a more resilient autonomous logic, an architecture upgrade should be considered. A natural extension is a hybrid TEE-MPC architecture in which K-of-N ROFL nodes, each running on different hardware (e.g., two ARM TrustZone nodes, two Intel TDX nodes and one AMD SEV-SNP node), must independently validate the escrow conditions and co-sign the withdrawal transaction using threshold signatures (e.g., Shamir-based ECDSA). This eliminates the single-hardware-vendor point of failure: an attacker would need to simultaneously exploit distinct microarchitectures. While this

introduces additional latency proportional to the consensus round, preliminary estimates suggest the overhead remains under 500ms for K=2, N=3, which is well within acceptable bounds for cross-chain swap finality windows that typically span minutes.

6.1 Performance Characterization

To contextualize our choice of TEEs over alternative privacy-preserving techniques, we briefly consider the feasibility of implementing the escrow facilitator using Fully Homomorphic Encryption (FHE). The two most critical operations in our protocol, hash-preimage validation via Keccak-256 and ECDSA transaction signing, are particularly ill-suited for FHE. Homomorphic evaluation of hash functions with comparable circuit depth to Keccak-256 currently requires minutes even on multi-core hardware (Wei and Lu, 2024), while no practical FHE implementation of ECDSA exists due to the extreme multiplicative depth of elliptic curve arithmetic. Beyond computation, FHE cannot autonomously interact with external systems: reading on-chain state and broadcasting signed transactions would require a trusted intermediary to decrypt results, defeating the trustless autonomy our protocol provides. TEEs, by contrast, introduce overhead of less than 7% over native execution (Zhu et al., 2024), making them the only currently viable approach for real-time cross-chain swap facilitation on chains whose escrow primitives stop at R_{HTLC}. Results are presented in Table 1.

Table 1. Speed of the two operations the escrow performs on every swap

Operation	TEE (plaintext in enclave)	FHE (TFHE on ciphertext)	Slowdown
Keccak-256 hash *validateSecret()*	< 1 µs; ~12 cycles/byte, native CPU	~5–23 min; TFHE-rs, SHA-256 class circuit, 16 cores	$\sim 10^{8}\times$
ECDSA signature *secp256k1 tx signing*	~0.5–2 ms; OpenSSL, single core	Hours+ (est.); no practical FHE impl. exists; extreme multiplicative depth	$\sim 10^{6}\times+$

TEE overhead <7% over native (Zhu et al., 2024). TFHE gate bootstrapping ~13ms/gate (Chillotti et al., 2016). SHA-256 under TFHE 5–23 min depending on parallelization (Wei and Lu, 2024). FHE ciphertext operations 10^{2}–$10^{5}\times$ slower than plaintext (Jiang and Ju, 2022). ECDSA under FHE has no published implementation; the figure is estimated from the multiplicative depth of EC scalar multiplication. Keccak-256 has circuit complexity comparable to SHA-256, both ~64 rounds of bitwise operations on a 256-bit state.

6.2 Positioning Relative to Existing Approaches

Existing cross-chain swap solutions generally fall into four categories:

1. bridge protocols relying on multi-sig committees (e.g., Wormhole, Axelar), which introduce trust assumptions in the validator set;
2. light-client relay approaches (e.g., IBC), which require both chains to support on-chain verification of the counterparty's consensus, which is unavailable on XRPL;

3. hash-time-locked contracts (HTLCs), which require the destination ledger to express the escrow condition natively;
4. TEE-based exchanges such as MERCURY (Wen et al., 2024), which move the matching and settlement logic into trusted hardware.

Our TEE-based approach addresses the gap left by category (3). As Section 3.2 sets out, a ledger may offer native conditional escrow and still not express the staged, caller-dependent condition an intent protocol requires; the facilitator supplies the remaining arguments while avoiding the committee trust assumptions of category (1). It differs from category (4) in aim: rather than a standalone exchange, this is a destination leg that preserves an existing protocol's order format, hashlock and timelock semantics, so a resolver need not know which side is contract-hosted and which enclave-hosted. The trade-off, in both cases, is the introduction of hardware trust.

7. CONCLUSION

This paper has demonstrated the successful application of Trusted Execution Environments (TEEs) to achieve autonomous, trustless logic, extending high-integrity operations beyond the constraints of traditional smart contracts. The core advantage is the ability to provide trustless automation and robust data confidentiality for sensitive assets such as cryptographic keys in settings where this was previously impossible. However, the system's reliance on a single hardware manufacturer for its root of trust represents a critical limitation, introducing a potential single point of failure vulnerable to microarchitectural exploits like side-channel attacks. Looking forward, this technology lays a foundation for development of other trustless and autonomous protocols.

ACKNOWLEDGEMENT

Special thanks to Tanner from 1inch for his helpful discussions and feedback on integrating TEEs. The author acknowledges the use of Claude (Anthropic) for assistance with proofreading, language polishing and preparation of figures. The author reviewed and edited all generated text and takes full responsibility for the content of this publication.

REFERENCES

1inch (2026). *Resolvers: Introduction*. 1inch Business Documentation. Available at: https://business.1inch.com/portal/documentation/resolvers/introduction

Augusto, A. et al. (2026). Exploiting Liquidity Exhaustion Attacks in Intent-Based Cross-Chain Bridges. *arXiv preprint arXiv:2602.17805*

Bachu, B. et al. (2024). Quantifying Price Improvement in Order Flow Auctions. *arXiv preprint arXiv:2405.00537*

Bukov, A. et al. (2024). *1inch Fusion+: Intent-Based Atomic Cross-Chain Swaps*. 1inch Network whitepaper. Archived at: https://web.archive.org/web/20250105202844/https://1inch.io/assets/1inch-fusion-plus.pdf

Chillotti, I. et al. (2016). Faster Fully Homomorphic Encryption: Bootstrapping in Less Than 0.1 Seconds. *Advances in Cryptology – ASIACRYPT 2016*. Hanoi, Vietnam, pp. 3–33.

Cloosters, T. et al. (2020). TeeRex: Discovery and Exploitation of Memory Corruption Vulnerabilities in SGX Enclaves. *Proceedings of the 29th USENIX Security Symposium*. Boston, USA, pp. 841–858.

Edera (n.d.). *Explaining Remote Attestation in Confidential Computing*. Available at: https://edera.dev/stories/remote-attestation-in-confidential-computing-explained

Herlihy, M. (2018). Atomic Cross-Chain Swaps. *Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC)*. Egham, United Kingdom, pp. 245–254.

Jean-Louis, N. et al. (2024). SGXonerated: Finding (and Partially Fixing) Privacy Flaws in TEE-based Smart Contract Platforms Without Breaking the TEE. *Proceedings on Privacy Enhancing Technologies*, Vol. 2024, No. 4, pp. 1–18.

Jiang, L. and Ju, L. (2022). FHEBench: Benchmarking Fully Homomorphic Encryption Schemes. *arXiv preprint arXiv:2203.00728*

Lan, Y. et al. (2021). TrustCross: Enabling Confidential Interoperability across Blockchains Using Trusted Hardware. *arXiv preprint arXiv:2103.13809*

Mithril Security (2023). *Confidential Computing: A History*. Mithril Security Blog. Available at: https://blog.mithrilsecurity.io/title-confidential-computing-a-history-ft-apple-intel-and-the-linux-foundation/

Oasis Protocol Foundation (2026). *Runtime Off-Chain Logic (ROFL) | Oasis Documentation*. Available at: https://docs.oasis.io/build/rofl/

Sabt, M. et al. (2015). Trusted Execution Environment: What It is, and What It is Not. *2015 IEEE Trustcom/BigDataSE/ISPA*. Helsinki, Finland, pp. 57–64.

Wei, B. and Lu, X. (2024). Improved homomorphic evaluation for hash function based on TFHE. *Cybersecurity*, Vol. 7, No. 1, p. 14.

Wen, X. et al. (2024). MERCURY: Practical Cross-Chain Exchange via Trusted Hardware. *arXiv preprint arXiv:2409.14640*

Wilde, A. et al. (2024). The Forking Way: When TEEs Meet Consensus. *arXiv preprint arXiv:2412.00706*

XRP Ledger (2026). *Escrow*. XRP Ledger Documentation. Available at: https://xrpl.org/docs/concepts/payment-types/escrow

Xu, J. et al. (2021). A Game-Theoretic Analysis of Cross-Chain Atomic Swaps with HTLCs. *Proceedings of the 41st IEEE International Conference on Distributed Computing Systems (ICDCS)*. Washington DC, USA, pp. 584–594.

Zhu, J. et al. (2024). Confidential Computing on NVIDIA Hopper GPUs: A Performance Benchmark Study. *arXiv preprint arXiv:2409.03992*