SPEEDSTER: An Efficient Multi-party State Channel via Enclaves

Jinghui Liao
RITAS and Department of CSE
SUSTech & Wayne State University & Neo Foundation
Shenzhen, Guangdong, China
liaojh2021@mail.sustech.edu.cn

Wenhai Sun
Department of Computer and Information Technology
Purdue University
West Lafayette, Indiana, USA
whsun@purdue.edu

Fengwei Zhang∗
Department of CSE and RITAS
SUSTech
Shenzhen, Guangdong, China
zhangfw@sustech.edu.cn

Weisong Shi
Department of Computer Science
Wayne State University
Detroit, Michigan, USA
weisong@wayne.edu

ABSTRACT

State channel network is the most popular layer-2 solution to the issues of scalability, high transaction fees, and low transaction throughput of public Blockchain networks. However, the existing works have limitations that curb the wide adoption of the technology, such as the expensive creation and closure of channels, strict synchronization between the main chain and off-chain channels, frozen deposits, and inability to execute multi-party smart contracts. In this work, we present SPEEDSTER, an account-based state-channel system that aims to address the above issues. To this end, SPEEDSTER leverages the latest development of secure hardware to create dispute-free certified channels that can be operated efficiently off the Blockchain. SPEEDSTER is peer-to-peer decentralized and provides better privacy protection than prior channel projects. It supports fast native multi-party contract execution, which is previously unavailable in TEE-enabled channel networks. Compared to the Lightning Network, SPEEDSTER improves the throughput by about 10,000× and generates 97% less on-chain data with a comparable network scale.

CCS CONCEPTS

• Security and privacy → Distributed systems security.

KEYWORDS

Multi-party State Channel; Layer-2; Scalability; Offchain Smart Contracts

ACM Reference Format:

∗Fengwei Zhang is the corresponding author

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

1 INTRODUCTION

Blockchain (aka layer-1 main chain) has been deemed a disruptive technology to build decentralized trust and foster innovative applications in both public and private sectors. However, scalability has become a great concern in practice when adopting the decentralized infrastructure. For example, the Bitcoin network [81] can only handle approximately 3, 500 transactions in every new block due to the block size limitation [18] and process 7 transactions per second (tps) on average [19, 81]. The issue has also haunted other major Blockchain networks which are based on a similar design principle, such as Ethereum [25]. Modifying the on-chain protocols helps alleviate the problem, for instance, using alternative consensus algorithms [79] and improving the information propagation [69, 102]. Nevertheless, changes at layer-1 Blockchain level may adversely affect the existing participants with undesired costs [46, 62]. Shifting to layer-2 payment channels [24, 68, 78, 84] is considered an effective remedy by carrying out micropayment transactions off the Blockchain to avoid the expensive on-chain overhead. State channels [1, 42, 78] further advances this off-chain innovation by enabling stateful transactions and smart contract execution. Promising as it is, the state channel also has the following limitations.

(L1) Opening a new channel requires freezing deposits of channel participants to lock in their collateral, which significantly affects liquidity rates and network efficacy. Every time a channel is created or closed, an associated transaction is required to send this signal to the main chain, thus incurring additional transaction fees and waiting time for main chain confirmation [1, 65, 68, 78, 84].

(L2) With the help of Hashed Timelock Contract (HTLC) [68], the architectural complexity is reduced and multi-hop transaction becomes feasible in the state channel network. However, HTLC also raises many privacy concerns with the intermediate nodes [40, 50, 52, 70] and leads to a multitude of attacks, such as wormhole attacks [71], bribery attacks [93], and DoS attacks [60, 90].

(L3) The current dispute resolution in the state channel is not robust and vulnerable to the denial-of-service (DoS) attack. A malicious channel participant can send an outdated channel state to the Blockchain while DoS-ing the victim to prevent the submission of the lasted channel state.
(L4) Despite the ambition of the instant processing of off-chain transactions [68], the complex routing and state updating mechanisms give rise to a non-negligible overhead, thus considerably degrading promised performance. The actual throughput of the state channels is still unsatisfactory (tens of tps measured in [40, 66, 78]).

(L5) The state exchange is confined within a pairwise channel, which poses fundamental challenges for creating and executing multi-party smart contracts. Though a multi-party state channel can be recursively established using the virtual channel techniques [31, 39, 40], the associated expensive cost is still a concern for implementation.

**Technical contributions.** We present **Speedster** to address the above limitations. The main idea of **Speedster** is that every user creates and funds an off-chain account protected by the enclave, an instance of a Trusted Execution Environment (TEE). As **Speedster** transfers the on-chain trust with the Blockchain to the off-chain trust with enclaves, we significantly reduce the design complexity to accomplish a plethora of innovations, such as multi-party channels, and lightweight protocols for channel confidentiality, authenticity, finalization, and dispute resolution. **Speedster** outperforms the conventional state channel networks in terms of security, performance, and functionality.

In **Speedster**, a node does not need to send an on-chain transaction to open/close a channel. Only one deposit transaction is needed to initialize a TEE-enabled account for each off-chain participant. Later, a participating node can directly create/close channels with any other nodes completely off the main chain with the balance in their enclave accounts, thereby turning **Speedster** into a peer-to-peer state channel network and resolving **L1**. **Speedster** addresses **L2** by eliminating the need for HTLC-based multi-hopping and routing [60, 71, 90, 93] via the use of the peer-to-peer state channel network.

**Speedster** adopts a novel certificate-based off-chain transaction processing model where the channel state is retained in the enclave. **Speedster** modifies the state before sending out or after receiving transactions to make sure the state submitted to the Blockchain is always up to date. As a result, **L3** is addressed as attackers cannot roll back to old states by DoS-ing counterparts and fool the Blockchain into biased decisions.

By leveraging the off-chain enclave trust, **Speedster** replaces the costly public-key algorithms with efficient symmetric-key operations for transaction generation and verification. Experimental results show that **Speedster** increases the throughput by four orders of magnitude compared to the Lightning Network, the most popular payment channel network in practice, thus, alleviating the concerns in **L4**.

Off-chain multi-party smart contracts for **L5**, can be enabled and efficiently executed in **Speedster**. With the certificate-based channels, **Speedster** naturally supports interactions among multiple parties. The state information can be correctly exchanged across multiple channels of the same account.

**Evaluation.** **Speedster** is intentionally designed to be compatible with different major TEE platforms for availability and usability, such as AMD [3], Intel [77], and ARM [9]. We evaluate its cross-platform performance to show the advantage over other popular layer-2 designs. Specifically, we migrate eEVM [32], a full version of Ethereum Virtual Machine (EVM) [43] into **Speedster**, and execute unmodified Ethereum smart contracts off-chain. We develop a set of benchmark contracts to show the unique features and performance of **Speedster**. Through thorough experiments, we present the **Speedster**'s specifications, the much-improved transaction throughput, and the capability of executing different kinds of smart contracts that traditional state channels cannot support. The experiments include:

- Transaction load test: To test the transaction throughput directly between two parties without loading any smart contract;
- Instant state sharing: Participants can update and share their states instantly; this is an important performance indicator for time-sensitive applications, such as racing games and decentralized financial services;
- ERC20 contracts: To show the performance of off-chain fund exchange;
- Gomoku contract: To show the performance of the turn-based contracts;
- Paper-Scissors-Rock contract: To illustrate the fairness (for in-parallel execution) in **Speedster** channel;
- Monopoly contract: To test the multi-party state channel capability of **Speedster**, we load a Monopoly smart contract that is executed by four players alternately.

The evaluation results show that **Speedster** is efficient and takes only 0.02ms, 0.14ms, and 20.49ms to process a value-transfer transaction on Intel, AMD, and ARM platforms, respectively, which leads to much higher throughput than that of Lightning network. The source code of **Speedster** is available at https://bit.ly/3a32ju7.

## 2 BACKGROUND

### 2.1 Blockchain and Smart Contract

Blockchain is a distributed ledger that leverages cryptography to maintain a transparent, immutable, and verifiable transaction record [25, 81]. In contrast to the permissioned Blockchain [7, 89], permissionless Blockchain [98] is publicly accessible but constrained by the inefficient consensus protocols, such as the Nakamoto consensus in Bitcoin [57, 81], on top of the asynchronous network infrastructure, which leads to a series of performance bottlenecks in practice. See [48, 86, 94, 105] for detailed discussion.

Smart contracts in Blockchain complement the ledger functions by providing essential computations. In general, a smart contract is a program that is stored as a transaction on the Blockchain. Once being called, the contract will be executed by all the nodes in the network. The whole network will verify the computation result through consensus protocols, thus creating a fair and trustless environment to foster a range of novel decentralized applications [75, 107]. A well-known example is the Ethereum smart contract [25, 26], which runs inside the EVM [43]. EVM needs to be set up on every full Ethereum node to create an isolated environment from the network, file system, and I/O services for contract execution. The user transactions will be taken as input to the contract inside the EVM.
2.2 Layer-2 Channels

Layer-2 technologies are proposed to address the scalability concerns [100], short storage for historical transactions [99], etc., for the layer-1 Blockchain.

Payment channel is the first attempt to use an off-chain infrastructure to process micropayments between two parties without frequent main chain involvement. To create a channel, each party needs to send a transaction to the Blockchain to lock in a certain amount of deposit on the main chain until a transaction is issued later to close the channel. When the channel is open, transactions can be sent back and forth between participants as long as they do not surpass the committed channel capacity.

Payment channel network (PCN) is built on top of the individual payment channels to route transactions for any pair of parties who may not have direct channel connections [60, 68, 78]. Hashed Timelock Contract is exploited to guarantee balance security along the payment route, i.e., the balances of the involved nodes are changed in compliance with the prescribed agreement. PCN greatly relieves the users from costly channel creation and management, but it also brings up concerns about the privacy with intermediate routing nodes and the formation of the centrality of the network.

State channel network extends PCN by allowing for stateful activities, such as off-chain smart contract [31, 39–42]. However, recording and updating states across multiple parties are still expensive due to the sophisticated trust management and protocol design. For example, the current multi-party state channel [31, 39] is realized through recursive virtual channel establishment [40–42], which introduces non-negligible complexity and overhead.

Regardless of the technical differences of the above layer-2 technologies, they all need to involve the inefficient Blockchain for channel creation, closure, or dispute resolution. Moreover, privacy and instability [90] concerns also arise and hamper the wide adoption of those technologies.

2.3 Trusted Execution Environment

Trusted Execution Environment provides a secure, isolated environment (or enclave) in a computer system to execute programs with sensitive data. Enclave protects the data and code inside against inference and manipulation by other programs outside the trusted computing base (TCB). Intel Software Guard eXtensions (SGX) [6, 53, 77] and AMD Secure Encrypted Virtualization (SEV) [5, 59] are two popular general-purpose hardware-assisted TEEs developed for the x86 architecture. Precisely, the TCB of SGX is a set of new processor instructions and data structures that are introduced to support the execution of the enclave. The TCB of AMD SEV is the SEV-enabled virtual machine protected by an embedded 32-bit microcontroller (ARM Cortex-A5) [59]. Other prominent TEE examples include TrustZone [9] and CCA [10] on ARM, MultiZone [47] and KeyStone [64] on RISC-V, and Apple Secure Enclave in T2 chip [8]. To demonstrate the cross-platform capability of Speedster, we implement a prototype that can run on Intel, AMD, and ARM machines, and we make Speedster design general enough for other TEE platforms not limited to the tested environments.

Remote attestation [83] is used to verify the authenticity of the enclave before executing enclave programs. Specifically, to prevent attackers from simulating the enclave, a TEE-enabled processor uses a hard-coded root key to cryptographically sign the measurement of the enclave, including the initial state, code, and data. Note that even if one TEE processor sets up multiple enclaves with the same set of functions, their respective measurements will be distinctively different. As such, everyone can publicly verify the authenticity of the established enclave with help from vendors.

3 THREAT MODEL AND DESIGN GOALS

3.1 Threat Model

We assume that nodes in the system run on TEE-enabled platforms, and all parties trust the enclaves after the successful attestation. An adversary may compromise the operating system of a target node and further control the system’s software stack.

In Speedster, we use TEE as a secure abstraction to make the design and security independent of the specific platforms. We provide rigorous security proofs to show the reliability and robustness of Speedster. However, like any secure function, theoretical security could be compromised by erroneous implementations. Therefore, to be consistent with prior work [30, 36, 66], we additionally consider attacks on specific TEE platforms in our implementation for completeness, which does not indicate the insecurity of the general design of Speedster. See Section 6 for the detailed discussion for the particular platforms.

Similar to prior research [31, 39–42], this work also assumes a Blockchain abstraction to provide desired ledger functions, such as transparent and immutable storage, and verifiable computations with smart contracts. Speedster assumes that the Blockchain nodes are equipped with adequate resources for computation and storage so that we only concentrate on the off-chain related design (see Appendix A for more discussion on the TEE and Blockchain abstractions).

3.2 Design Goals

Efficient Channel System (L1, L3, L4): The current layer-2 channel system design principle derails from the promised efficiency for off-chain micropayment processing. As discussed in Section 2.2, the existing systems need expensive interactions with the Blockchain for various channel operations in terms of time and economic costs. Users are required to trust the intermediate nodes and pay extra fees for transaction forwarding and state updating.

In this work, we attempt to devise a functionally efficient off-chain network that aims to significantly reduce the channel cost for creation and closure and eliminate the dispute in light of unsynchronized communications.

Peer-to-Peer Channel Network (L2, L4): Due to the expensive channel cost, a node in layer-2 currently cannot afford to establish direct channel connections with all other nodes in the system. Multi-hopping addresses the problem but raises privacy concerns about the emergent centralized payment hubs [40, 52, 85], which is at odds with the decentralized promise of Blockchain.

In contrast, we aim to build a peer-to-peer channel network to allow users to freely set up direct channels with intended parties, thus eliminating centrality concerns. Note that none of the existing work can support this function [65, 66, 68, 78].
Efficient Multi-Party State Channel (L5): Sharing states among multiple parties is instrumental for many real-world applications, such as voting, auctioning, and gaming. However, most off-chain state channels only support pairwise state exchange [38, 40]. The involvement of more channel participants depends on intermediaries, which complicates the network setup and trust management [31, 39]. SPEEDSTER targets a more efficient multi-party state channel by streamlining the architectural design for easy setup and use. The state information of one SPEEDSTER node can be freely shared with other parties of interest without worrying about the additional cost in prior work.

Other Goals: Besides, SPEEDSTER also aims to: (1) preserve the privacy of transactions (see Section 6 for detailed security definition and analysis), (2) be abstract and general enough to not rely on any specific TEE platform.

4 SPEEDSTER DESIGN

4.1 System Architecture

![Figure 1: Framework of SPEEDSTER. A channel is opened directly between enclaves of two users. Off-chain transactions are processed by prog\text{enclave} in the enclave. The contract \text{CERTIFIED CHASSIS} is deployed on the Blockchain to record the states of the nodes. The initial state of the enclave is synchronized from the Blockchain.](image)

SPEEDSTER contains two components: the state channel core programprog\text{enclave}, executed inside the enclave and the on-chain smart contract contract\text{CERTIFIED CHASSIS} running on the Blockchain. Figure 1 shows the high-level architecture of SPEEDSTER, in which two participants are connected by a Certified Channel (see Definition 1).

\text{Prog}\text{enclave}: The program that operates inside the enclave is referred to as prog\text{enclave}. prog\text{enclave} creates and manages an enclave account for a SPEEDSTER node. It executes commands from the user to open and close channels as well as constructs and processes channel transactions. To verify enclave authenticity, it also generates measurements for remote attestation.

\text{Contract}\text{CERTIFIED CHASSIS}: contract\text{CERTIFIED CHASSIS} is a smart contract deployed on the Blockchain to manage the on-chain states of SPEEDSTER accounts. To register an account, a deposit must be sent to this contract and recorded in the Blockchain. This record will then be used to initialize the enclave state. The smart contract also handles transactions to claim funds for SPEEDSTER accounts.

4.2 Workflow

In this subsection, we outline the workflow of SPEEDSTER which includes: (1) node initialization, (2) enclave state attestation, (3) channel key establishment, (4) channel certification, and (5) multi-party state channel establishment (optional). The workflow is illustrated in Figure 2.

Node Initialization: When the program prog\text{enclave} is loaded into the enclave for the first time, an account acc\text{enclave} along with a pair of keys pk and sk are generated. The enclave keeps sk private and publishes pk as the account address that can be used to deposit acc\text{enclave} on the Blockchain. To ensure the authenticity of the opened account acc\text{enclave} for off-chain attestations, an initial deposit transaction is required to register the account on the Blockchain. After the Blockchain confirms the transaction, the user loads relevant information into the enclave as proof-of-registration to initialize the enclave state state\text{0} := (tx, aux), a tuple that contains the deposit transaction tx and auxiliary information aux, where tx can be more than one deposit and aux can be the current balance or account-related configuration information. Further deposits will update the initial state state\text{0}.

Enclave State Attestation: Step 2 is enclave attestation that needs to be carried out to authenticate the enclave environment including state\text{0}. Note that we add the initial state state\text{0} and the public key pk into the enclave measurement attic := H(SORT((pk0, pk1))) is assigned for the channel, where H is a hash function and SORT can be any function used to make sure both parties agree on the same order of pk’s, thus leading to the identical ccid. Next, both ends create a certificate certi := (pk_{i-1}, inp_{i}, \sigma_i)_{i \in \{0,1\}}

\[msg_1 = \text{tx}
\]
\[msg_2 = \text{\text{att}}
\]
\[msg_3 = \text{pk}
\]

Figure 2: Workflow of node initialization and certified channel creation. E is the environment, including the Blockchain and the channel users, who pass input to SPEEDSTER nodes.
for the other party by including the target public key pk as the identifier. With the cert, a channel user can claim the fund received from counterpart on the Blockchain when channel is closed.

**Multi-party State Channel Establishment**: This step is optional for establishing the multi-party state channel. To this end, a group channel-key is generated for securely sharing the channel states among participants. This step cannot complete until after all the necessary two-party channels have been established. Note that the group key only works for the multi-party state channel function and coexists with the keys for direct channels (see Section 4.3).

### 4.3 Key Functions

**Certified Channel**: One main challenge by incorporating TEE into the Blockchain is that current Blockchain implementation does not support remote attestation for TEE platforms. As a result, Blockchain cannot verify the authenticity of the transactional activities from layer-2. To address the problem, we propose Certified Channel defined below.

**Definition 1 (Certified Channel)**. A Speedster channel is called a Certified Channel if it is established between two attested enclave accounts and both participants have the channel certificate issued by the other party.

With the Certified Channel designation, Blockchain is agnostic to the enclave attestation and offloads this task to the layer-2 nodes. As long as a node can present a valid certificate issued by the other channel party, Blockchain will trust this enclave node and its associated transactions. In this way, balance security is guaranteed.

**Dispute-free Channels**. The main reason for the disputes existing in prior state channel networks is that Blockchain struggles to discern old states in an asynchronous network. A victim node may be intentionally blocked, for instance, in favor of an attacker’s claim when closing a channel [66, 68, 84]. With Certified Channel, Speedster relies on enclaves to correctly update its state before sending out and after receiving transactions. The node locks the channel states if it intends to send a “claim” transaction to the Blockchain. As a result, channel states are always up to date and from counterpart on the Blockchain when channel is closed. In this regard, Speedster is free from expensive on-chain dispute resolution operations.

**Peer-to-Peer Channel Network**: We anticipate that a peer-to-peer channel network (P2PCN) will significantly improve layer-2 network stability while complementing the decentralized nature of Blockchain technology. We define a peer-to-peer channel network as follows.

**Definition 2 (Peer-to-Peer Channel Network)**. A payment/state channel network in which a node can establish direct channel connections with other nodes efficiently off-chain and process transactions without relying on intermediaries.

It is economically impractical to turn current state channel networks into P2PCN because it will lock in a significant amount of collaterals into the main chain. Speedster addresses this issue by adopting an account-based channel creation structure that uses every single on-chain deposit to open multiple off-chain channels.

P2PCN also eliminates the need for transaction routing intermediaries, thus relieving users of additional fees, operational costs, and security and privacy concerns.

**Multi-Party State Channel**: As discussed in Section 3.2, achieving a multi-party state channel is inherently challenging but necessary for many off-chain smart contract use cases, such as multi-party transactions and games. Next, we detail our design.

**Multi-party channel establishment**. Before establishing a group channel, we assume that a peer-to-peer channel has already been set up between each pair of members beforehand. With n known participants in a tentative multi-party channel to be created, the channel id ccid is generated by hashing the sorted public keys of all participants as follows: ccid := H(SORT([pk_i]_{i\in[N]})). Then, a group key gk can be generated with any secure multi-party key exchange algorithm [12, 16, 20]. The group key gk is then bound with the ccid, and only transactions with a tag that matches the ccid can use the key for encryption and decryption. As a result, multi-party channel transactions only need to be encrypted once, then broadcast to other members.

![Figure 3: An example of executing a multi-party transfer contract among A, B and C, assuming SOR(pk_A) > SOR(pk_B) > SOR(pk_C). (+) and (-) in the tables represent the balance change after each respective Certified Channel transaction.](image)

**Coordinated transaction execution**. To avoid transaction execution ambiguity in a multi-party smart contract scenario, transactions from different parties need to be ordered before being processed. In a distributed network, it is difficult to locate a trusted time source for coordination. To address this issue, we order each party i send their transactions in order as determined by SOR([pk_i]_{i\in[N]}). Specifically, all nodes except for the one with the highest SOR function value are muted after the channel key is created. Moving forward, all other nodes need to wait for their turn for execution. Figure 3 shows an example of how a value-transfer multi-party contract is executed among three channel members A, B, and C. In the figure, Certified Channels are opened between any two member nodes. The nodes send transactions tx_i, tx_{i+1}, and tx_{i+2} successively through the multi-party state channel identified by a ccid.
figure also shows the balance change of A with other two channel members after each round of communication. Note that the total balance of underlying Certified Channels should not surpass the amount allocated by the nodes for the multi-party channel at any time. Moreover, channel members are also relieved from disputes concerns thanks to the unsynchronized state inherited from the underlying Certified Channels.

5 \(\Pi_{\text{Speedster}}\) Protocol

5.1 \(\Pi_{\text{Speedster}}\) Protocol \(\Pi_{\text{Speedster}}\)

We use the ideal functionalities \(F_{\text{Blockchain}}\) [Contract] and \(G_{\text{att}}\) [27, 83] (see detail in Appendix A) to formally present the protocol \(\Pi_{\text{Speedster}}\) in two parts: the program \(\text{prog}_{\text{enclave}}\), in Appendix Figure 8, that runs the enclave and the smart contract \(\text{contracts}_{\text{Speedster}}\) running on the Blockchain, shown in Appendix Figure 9. In the protocol, \(P\) denotes a user, \(R\) as the counterpart users in a channel, and \(\text{tx}\) represents an on-chain transaction. To execute an off-chain smart contract in \(\text{prog}_{\text{enclave}}\), we define the function \(\text{Contract}_{\text{cid}}(\cdot)\) as parameterized with smart contract id \(\text{cid}\). \(\text{Contract}_{\text{cid}}(\cdot)\) consumes the channel state and node balance to ensure balance consistency.

Node Initialization: To initially boot up a \(\text{Speedster}\) node, a node sends the "install" command to the enclave to load \(\text{prog}_{\text{enclave}}\). Then, the node calls the function (1) of \(\text{prog}_{\text{enclave}}\) by sending a message ("init") to create an enclave account \(\text{acc}_{\text{enclave}}\) with key pair \((\text{sk}, \text{pk})\). For attestation purposes, an enclave measurement \(\sigma_{\text{att}}\) is generated with the \(\text{prog}_{\text{enclave}}\) public key \(\text{pk}\) of \(\text{acc}_{\text{enclave}}\) and the node initial state \(\text{state}^0\).

Deposit: To deposit, a node must first sends a \(\text{tx}\) to \(\text{contracts}_{\text{Speedster}}\) on the main chain. The transaction includes the pk of the enclave account \(\text{acc}_{\text{enclave}}\) as the account address. Next, the node calls function (2) of \(\text{prog}_{\text{enclave}}\) by sending the message "deposit" and passing \(\text{tx}\) as a parameter. Finally, \(\text{prog}_{\text{enclave}}\) verifies the signature of \(\text{tx}\), and updates the local initial state \(\text{state}^0\).

Certified Channel: Each certified channel in \(\Pi_{\text{Speedster}}\) is identified by a channel ID \(\text{cid}\). A shared channel key \(\text{ck}\) is produced in this step. The certificate \(\text{cert}\) of the channel is created using the public keys of both parties. To prevent rollback attacks on \(\text{att}_{\text{enclave}}\), \(\text{prog}_{\text{enclave}}\) generates a signature \(\sigma_{\text{att}}\) by signing the tuple \((\text{state}^0, \{\text{pk}_j\}_{j=0}^1, \text{prog}_{\text{enclave}})\) for each channel after function (3) returns. The tuple is signed by the manufacture secret key \(\text{msk}\) to reflect the root trust embedded in the hardware. The cert is verified in function (5).

Multi-Party State Channel: A multi-party state channel is built upon the existing certified channels. To create a multi-party state channel, a node calls function (4) of \(\text{prog}_{\text{enclave}}\) by sending a message "openMulti" and passing a set of \(\text{cid}\) to inform the underlying certified channels with other participants of this multi-party state channel. We abstract out the process of multi-party shared key generation, which could be replaced with any secure multi-party key negotiation protocol [12, 16, 20].

Transaction: To send a channel transaction, a node calls function (6) of \(\text{prog}_{\text{enclave}}\) via the "send" command through \(G_{\text{att}}\).\text{Resume}() and passes the target \(\text{cid}\) along with other necessary parameters in input \(\text{inp}\). Then, \(\text{prog}_{\text{enclave}}\) executes \(\text{inp}\) with the associated contract by calling \(\text{Contract}_{\text{cid}}(\cdot)\) and updating the channel state accordingly. A channel transaction is constructed over the public key of \(\text{pk}\), the new channel state \(\text{state}^i\), the input \(\text{inp}\), and the output \(\text{outp}\). Then, the transaction is encrypted with an authentication scheme, such as AES-GCM, using the channel key \(\text{ck}\).

Claim: To claim the funds that \(P\) receives from the channel transactions, the node issues a "claim" call to function (7) of \(\text{prog}_{\text{enclave}}\). \(\text{prog}_{\text{enclave}}\) first freezes all two-party channels, and extract all \(\text{certs}\) from those channels. The \(\text{certs}\) and the local node state \(\text{state}\) constitute the claim transaction \(\text{tx}_{\text{prog}_{\text{enclave}}}\), then signs the \(\text{tx}\) with the private key \(\text{sk}\) of \(\text{acc}_{\text{enclave}}\) and returns the signed transaction that is further forwarded by the node to the \(\text{contracts}_{\text{Speedster}}\). In the end, \(\text{contracts}_{\text{Speedster}}\) verifies and executes the claim transaction on the Blockchain to redeem funds for the node.

6 Security and Privacy Analysis

We formalize the Universal Composability (UC) [11, 27, 63, 66] ideal functionality \(F_{\text{Speedster}}\) (shown in Figure 6 in the Appendix) to realize the security goals of \(\Pi_{\text{Speedster}}\).

The security of \(\Pi_{\text{Speedster}}\) is explained in Theorem 1.

Theorem 1 (UC-Security of \(\Pi_{\text{Speedster}}\)). If the adopted authenticated encryption \(AE\) is IND-CCA secure and digital signature scheme \(E\) is EU-CMA secure, then the protocol \(\Pi_{\text{Speedster}}\) securely UC-realizes the ideal functionality \(F_{\text{Speedster}}\) in the \((G_{\text{att}}, F_{\text{Blockchain}})\)-hybrid model for static adversaries.

Proof. (Sketch) We prove that the protocol \(\Pi_{\text{Speedster}}\) securely UC-realizes ideal functionality \(F_{\text{Speedster}}\) by simulating the behavior of a real-world adversary \(\mathcal{A}\) in an ideal world simulator \(\mathcal{S}\). Showing that \(\mathcal{S}\) could indistinguishably simulate the behavior of \(\mathcal{A}\) for all environment \(\mathcal{E}\) [27] proves the security of \(\Pi_{\text{Speedster}}\). Let \(\mathcal{E}\) be an environment and \(\mathcal{A}\) be a real-world probabilistic polynomial-time (PPT) adversary who simply relays messages between \(\mathcal{E}\) and dummy parties. To show that \(\Pi_{\text{Speedster}}\) UC-realizes \(F_{\text{Speedster}}\), we specify a simulator \(\mathcal{S}\) below such that no environment can distinguish an interaction between \(\Pi_{\text{Speedster}}\) and \(\mathcal{A}\) from an interaction with \(F_{\text{Speedster}}\) and \(\mathcal{S}\). That is, for any \(\mathcal{E}, \mathcal{S}\) satisfies

\[\forall \mathcal{E} \in \text{EXEC}, \mathcal{S} \approx \text{EXEC}, \mathcal{S} \in \text{EXEC}, \mathcal{S} \in \mathcal{S}\]

A detailed proof can be found in Appendix B.

Theorem 1 also implies stronger privacy protection compared to conventional payment/state channel networks in that: (1) All \(\text{Speedster}\) channels are created directly between participants. No intermediate node is required to relay transactions, thus alleviating the privacy concerns introduced by HTLC [40, 50, 52, 70]; (2) off-chain channels transactions are encrypted by AES-GCM, and only the enclaves of participants can decrypt it. Therefore, \(\text{Speedster}\) ensures transaction confidentiality.

Preventing Double-Spending Attacks. Each processor has a unique built-in key that is hard coded in the CPU [3, 6] to differentiate its identity during attestation. Moreover, the processor generates and assigns each enclave a unique identifier [3, 53] ensuring that even enclaves created by the same processor are distinctive. To prevent double-spending attacks, \(\text{prog}_{\text{enclave}}\) updates balance before sending transactions to peers. Once the state is updated, it
can not be rolled back. Therefore, no fund can be spent multiple times in Speedster.

Defending Against TEE Attacks. The hardware-assisted TEE serves as a way to replace complex software-based cryptographic operations. Promising as it seems, recent research shows that TEE implementations on specific platforms are vulnerable to the side-channel attacks [49, 80, 88, 95, 97], rollback attacks [21, 34, 72] and incorrect implementation and configuration [13, 23, 54, 82]. In Speedster, we use a generalized TEE abstraction that does not rely on a specific platform’s design, and its security has been proven in Theorem 1. In addition, we offer suggestions and proactively mitigate the above vulnerabilities for both hardware and software. For example, we use SEV-SE [3] to protect against specific speculative side-channel attacks and TCB rollback attacks. We also update the microcode of Intel/AMD/ARM TEE to the latest version. Besides these measures, proper implementation of the system can also help mitigate known side-channel vulnerabilities [56]. Speedster uses a side-channel-attack resistant cryptographic library [73], and requires that all nodes run on the latest version of the firmware to defend against known TEE attacks. Further, an adversary may launch a DoS attack against the node by blocking the Internet connection of the victim or abruptly shutting down the OS to force quit the enclave functions. While beyond the scope of this article, such DoS attacks can be addressed by adopting a committee enforcement design [30, 66]. The channel node state is jointly managed by a committee of TEE nodes to tolerate Byzantine fault. Despite the inevitable performance loss in light of the complexity of the committee chain, Speedster still outperforms existing works by enabling efficient multi-party state processing and management in a peer-to-peer manner (see Section 4.3 and Section 7.2.4).

7 IMPLEMENTATION AND EVALUATION

7.1 Implementation of Speedster

We build a Virtual Machine (VM) on top of the open-source C++ developed Ethereum Virtual Machine eEVM [32], which allows Speedster to run off-the-shelf Ethereum smart contracts. The cryptographic library used in _proenclave_ is mbedTLS [73], an open-source SSL library ported to TEE [33, 103]. For this work, we adopt 1) SHA256 to generate secret seeds in the enclave and the hash value of claim transactions, 2) AES-GCM [76] to authentically encrypt transactions in the state channel, and 3) ECDSA [58] to sign _cents_ and claim transactions. We also customize the OpenEnclave [33] to compile the prototypes for Intel and ARM platforms. For AMD SEV, we use VMs as the enclaves to run _proenclave_ as the host can communicate with the enclave via the socket. To highlight the advantages of Speedster, the performances of a few functions are tested, as discussed below.

Direct Transactions (Trade): This function is implemented in C++ and allows users to directly transfer _funds_ and share messages through channels without calling an off-chain smart contract. Before sending out a transaction, the sender first updates its local enclave state (e.g., the account balance), then marks the transaction as "sent". Communication between the sender and receiver enclaves is protected by AES-GCM.

Instant State Sharing: We implement an instant state sharing function in C++ to allow a user to create direct channels with other users off-chain. We also remove costly signature operations for transactions and replace it with AES-GCM, thereby significantly reducing communication overhead and enabling instant information exchange (like high-quality video/audio sharing) while preserving privacy. This is previously difficult to realize using asymmetric cryptographic functions [66, 68, 78].

Faster Fund Exchange: We implement a ERC20 contract [96] with 50 LOC in Solidity [35] to demonstrate the improved performance of Speedster in executing off-chain smart contracts. This can be attributed to the elimination of asymmetric signature operations for off-chain transactions.

Sequential Contract Execution: To highlight the performance of Speedster in executing sequential transaction contracts, we implement the popular two-party Gomoku chess smart contract with 132 LOC in Solidity. Furthermore, players cannot reuse locked _funds_ until the game ends, thus nullifying all benefits of cheating the system.

Parallel Contract Execution: Applications that require simultaneous user action, such as Rock-Paper-Scissors (RPS), are not easy to run in conventional sequentially structured state channels. Speedster supports applications running in parallel, faithfully manages multi-party states, and only reveals to players the final results. We implement a typical two-party RPS game with 64 LOC in Solidity to demonstrate this.

Multi-party Applications: To test the ability of multi-party off-chain smart contract executions, a Monopoly game smart contract with 231 LOC in Solidity is implemented. In this game, players take turns rolling two six-sided dice to determine how many steps they will move forward and how to interact with other players.

7.2 Evaluation

**SGX platform:** We test Speedster with a quad-core 3.6 GHz Intel(R) E3-1275 v5 CPU [55] with 32 GB memory. The operating system that we use is Ubuntu 18.04.3 TLS with Linux kernel version 5.0.0-32-generic. We also deploy LN nodes [67] as the baseline for comparison on another physical machine with the same configurations.

**SEV platform:** We evaluate Speedster on an SEV platform with 64 GB DRAM and an SEV-enabled AMD Epyc 7452 CPU [4], which has 32 cores and a base frequency of 2.35 GHz. The operating system installed on the AMD machine is Ubuntu 18.04.4 LTS with an AMD patched kernel of version 4.20.0-sev [3]. The version of the QEMU emulator that we use to run the virtual machine is 2.12.0-dirty. The virtual machine runs Ubuntu 18.04 LTS with the kernel version 4.15.0-101-generic and 4 CPU cores.

**TrustZone platform:** The evaluation of TrustZone is carried out in the QEMU cortex-a57 virtual machine with 1 GB memory and Linux buildroot 4.14.67-g333dc9e97-dirty as the kernel.
7.2.1 Code Size. To port eEVM into SPEEDSTER, we added extra 650 LOC to eEVM. In general, the eEVM contains 3.2k LOC in C++ and another 22.1k LOC coming from its dependencies. SPEEDSTER is evaluated on Intel, AMD, and ARM platforms with around 38.5k LOC in total, as shown in Table 1. Specifically, 25.3k LOC comes from the contract virtual machine eEVM [32] which is shared with all cases. progenclace has 3.1k LOC in C++ for SGX/TrustZone and 3.7k LOC for AMD SEV. The contract SPEEDSTER deployed on the Blockchain is implemented with 109 LOC in Solidity.

7.2.2 Time Cost for Transaction Authentication. In the SPEEDSTER prototype, we use AES-GCM to replace the ECDSA adopted in previous channel projects for transaction authentication. By trusting a secure enclave, SPEEDSTER uses efficient symmetric operations to simultaneously achieve both transaction confidentiality and authenticity. Figure 4 compares the performance of ECDSA and AES-GCM when processing 128, 256, and 1024 bytes of data, respectively. This experiment is carried out on Intel, AMD, and ARM platforms with four operations: ECDSA sign, ECDSA verify, AES-GCM encrypt, and AES-GCM decrypt. ECDSA is evaluated under secp256k1 curve. The key size of ECDSA is 256 bits while that of AES-GCM is 128 bits.

The experiment results are averaged from 10,000 trials and shown in Table 2 with the implemented smart contracts ERC20, GoMoku, and rock-paper-scissors (RPC).

**Table 1: Code size in Speedster.**

<table>
<thead>
<tr>
<th>Component</th>
<th>Code</th>
<th>LOC</th>
<th>Total(LOC)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shared</td>
<td>eEVM</td>
<td>C++</td>
<td>25.3k</td>
</tr>
<tr>
<td>SGX/TrustZone</td>
<td>progenclace</td>
<td>C++</td>
<td>3.1k</td>
</tr>
<tr>
<td></td>
<td>other</td>
<td>C++</td>
<td>2.3k</td>
</tr>
<tr>
<td>AMD SEV</td>
<td>progenclace</td>
<td>C++</td>
<td>3.7k</td>
</tr>
<tr>
<td></td>
<td>other</td>
<td>C++</td>
<td>4.1k</td>
</tr>
</tbody>
</table>

![Figure 4: Performance comparison between ECDSA and AES-GCM enabled transaction security on SGX, SEV, and TrustZone platforms. We run every experiment 10,000 times.](image)

For the direct transaction, SEV: AES-GCM takes an average of 0.1372 ms. When invoking smart contracts, the time cost varies for different applications. As shown in Table 2, RPC (0.1365 ms) and ERC20 (0.1382 ms) are faster than Gomoku (0.6667 ms) due to simpler logic and fewer steps.

**Evaluation on AMD:** As the evaluation of ARM TrustZone runs upon the QEMU emulator, the performance of ARM is the worst. Nevertheless, the evaluation results in Table 2 show that progenclace takes 20.496 ms to run direct transactions. For smart contract execution, it typically takes 30 - 90 ms to process contract transactions.

![Figure 5: The kernel of the virtual machine is 5.3.0-1034-azure, and the operating system is version 18.04.5 LTS. LN node is deployed and evaluated on](image)
the machine as a baseline to highlight the significant performance improvement of Speedster. We run every experiment 10 times and every time we run 10,000 transactions in series, Table 3 shows the evaluation result. The throughput of LN is 14tps while Speedster achieves 72, 143tps on payment operation, 5,000x more efficient than LN. Specifically, the latency to execute a Speedster transaction is around 80ms, close to the RTT between testing hosts, while the latency to run an LN payment transaction is around 500ms.

TeeChain is a TEE-supported payment channel network [66]. We tried hard to run a head-to-head comparison with it but failed to do so 2. Instead, we provide insights for a theoretical comparison. TeeChain nodes coupled with committee chains to defend against node failure. Speedster can be adapted to a similar design but inevitably sacrifices the performance 3. In this regard, the throughput of the committee-based Speedster will be comparable with that of TeeChain. However, Speedster is much more efficient in off-chain channel creation/closure (see Section 7.2.5) and supports multi-party state processing.

![Figure 5: Network setup for the evaluation.](image)

7.2.4 Channel System Comparison. To highlight the advantages of Speedster, we compare Speedster with other major channel projects. Table 4 shows these differences in terms of the following features: Direct off-chain channel open/closure, dynamic deposit (dynamically adjusting funds in an existing channel on-demand [66]), symmetric-key operations for transactions (using symmetric encryption algorithms to ensure the authenticity and privacy of off-chain transactions), off-chain smart contract execution, full decentralization (see Definition 2), multi-party state channel, dispute-free, and duplex channel (where both channel participants can send funds back and forth).

We compare the functions provided by Speedster and TeeChain. TeeChain is not a peer-to-peer channel network. Despite the dynamic deposit and bilateral termination [66], every channel opened in TeeChain has to be associated with a deposit locked on the main chain. As a result, similar to the Lightning network, creating many channels requires freezing a significant amount of collateral on the Blockchain and incurring expensive on-chain operations. Therefore, it is not realistic to build direct channels for any pair of nodes in the network. Alternatively, TeeChain still largely depends on HTLC for transaction routing in practice, which leads to privacy concerns. On the contrary, a deposit in Speedster can be shared by multiple off-chain channels. Direct channels can be efficiently established. Further, TeeChain does not support the off-chain smart contract execution and multi-party state channels. The pairwise channel structure of TeeChain confines the state within the channel. In contrast, due to balance sharing and Certified Channel, states across multiple channels can be managed and exchanged authentically in the same Speedster account.

In Perun [40], virtual channels can also be opened and closed off the Blockchain, but once the channel is created, the underlying ledger channels have to be locked. The minimum funds across the ledger channels determine the available capacity. As shown in Table 4, Speedster is the only off-chain state channel project that accomplishes all the listed functions.

7.2.5 Main Chain Cost. Similar to the previous works [24, 66], we evaluate the main chain costs: (1) the number of required on-chain transactions and (2) the number of pairs of public keys and signatures that are written to the Blockchain (defined as Blockchain cost in [24]).

We select a set of representative channel projects to evaluate and compare with Speedster. In particular, we choose LN [68] (the most popular payment channel system in reality), DMC [37] (a duplex payment channel), TeeChain [66] (a TEE-based channel project), and SFMC [24] (it also supports off-chain channel open/closure). The comparison is carried out by analyzing each project under bilateral termination [66], i.e., a channel is closed without disputes. The result is shown in Table 5. We take LN and TeeChain, for example, to demonstrate the cost efficiency of Speedster.

Before opening an LN channel, each node has to send one on-chain transaction with a Blockchain Cost (BC) of 1 to commit a deposit in the channel. Then, each LN channel has to send one on-chain transaction with a BC of 2. To close this channel, one of the channel’s participants needs to send a transaction with the latest channel state and signatures from both sides to the Blockchain.

In TeeChain [66], a group of committee nodes handles and dynamically associates deposits with channels. Thus, at least one "deposit" transaction is needed to set up the system with a BC of 1 + p/2, where p is the size of the committee. Since TeeChain can also close the channel off-chain, associated costs can be avoided. All TeeChain committee members use the same m-out-of-p multi-signature for each "deposit" transaction, so the BC is 1 + p/2 + m.

In contrast, a deposit to a Speedster account can be freely allocated to different channels. Therefore, we only need 1 "deposit" transaction to initialize the account and create c channels. There is no cost to open or close channels as Speedster can do this completely offline. To claim the remaining fund from active channels, one on-chain transaction needs to be sent. Assuming that one deposit and one claim transactions are shared by c channels on average,
Table 4: Feature comparison with other channel projects.

<table>
<thead>
<tr>
<th>Features</th>
<th>Channel Projects</th>
</tr>
</thead>
<tbody>
<tr>
<td>Direct Off-chain Channel Open</td>
<td>✓</td>
</tr>
<tr>
<td>Direct Off-chain Channel Close</td>
<td>✓</td>
</tr>
<tr>
<td>Dynamic Deposit</td>
<td>✗</td>
</tr>
<tr>
<td>Off-Chain Contract Execution</td>
<td>✗</td>
</tr>
<tr>
<td>P2PCN</td>
<td>✗</td>
</tr>
<tr>
<td>Multi-Party State Channel</td>
<td>✗</td>
</tr>
<tr>
<td>Dispute-Free</td>
<td>✗</td>
</tr>
<tr>
<td>Duplex Channel</td>
<td>✗</td>
</tr>
</tbody>
</table>

Table 5: Number of on-chain transactions and Blockchain Costs (BC) per channel.

<table>
<thead>
<tr>
<th>Payment Channel</th>
<th>Setup</th>
<th>Open</th>
<th>Close</th>
<th>Claim Total</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>No.tx</td>
<td>BC</td>
<td>No.tx</td>
<td>BC</td>
</tr>
<tr>
<td>LN [68]</td>
<td>2/1</td>
<td>1/0</td>
<td>2/2</td>
<td>0</td>
</tr>
<tr>
<td>TeeChain [66]</td>
<td>1/p</td>
<td>0/1</td>
<td>1/p+2/0</td>
<td>1/p+2+m</td>
</tr>
<tr>
<td>DMC [37]</td>
<td>0/1</td>
<td>0/0</td>
<td>2/2</td>
<td>0</td>
</tr>
<tr>
<td>SFMC [24]</td>
<td>1/p</td>
<td>0/1</td>
<td>1/p+2/0</td>
<td>1/p+2+m</td>
</tr>
<tr>
<td>Speedster</td>
<td>1/p</td>
<td>0/1</td>
<td>1/p+2/0</td>
<td>1/p+2+m</td>
</tr>
</tbody>
</table>

SPEEDSTER requires 2/c on-chain transactions with a BC of 2/c for each channel on average.

In summary, we observe that SPEEDSTER needs 80% less on-chain transactions than LN and the same number of transactions as TeeChain when c ≥ 2 and one deposit when a 2-out-of-3 multi-signature is used for each TeeChain channel. For the BC of each channel, SPEEDSTER outperforms LN by at least 66% when c ≥ 2, and 97% if c ≥ 11 [17]. Compared to TeeChain, SPEEDSTER reduces BC by over 84% when c ≥ 2.

8 DISCUSSION AND LIMITATION

Availability of TEE Hardware. SPEEDSTER leverages TEE to ensure the off-chain trust, which implies that only hardware equipped with TEE may join SPEEDSTER. However, as stated in Section 2.3, all major CPUs vendors of various architectures have incorporated TEE into their chip design. We have shown that SPEEDSTER can be deployed to run over multiple kinds of platforms, including Intel SGX, AMD SEV, and ARM TrustZone. We plan to further implement SPEEDSTER on more CPU architectures, such as RISC-V.

Security of TEE. Although TEE implies strong security assumptions to provide a secure and isolated execution environment, different platforms may have varying implementations that may contain a variety of known and unknown faults that could jeopardize the protection. Indeed, it is impossible to eradicate all TEE attacks, we explored defensive strategies for TEE vulnerabilities in Section 6, such as single node failure and rollback attacks.

Privacy Concerns with Remote Attestation. Remote attestation must be performed on the central server of the chip manufacturers, which is a centralized approach that raise privacy concerns [51]. However, the message used for remote attestation contains no runtime information about _provenance_, therefore, the privacy of transactions in SPEEDSTER channel is preserved. Additionally, there are ongoing efforts to address the privacy concerns associated with remote attestation [29, 87], which SPEEDSTER could adopt in the future without breaching any commitment.

9 RELATED WORK

HTLC Privacy and Security: HTLC is one of the fundamental building blocks in the current layer-2 channel design to facilitate transactions between parties without direct channel connections [68, 84]. HTLC comes with privacy issues [40, 50, 52, 70], however, and is vulnerable to various types of attacks [60, 71, 90, 93]. MAP-PCN [92], MHTLC [70], AMHL [71], and CHTLC [101] tried to address the privacy issues introduced by HTLC by adding additional countermeasures. MAD-HTLC [93] presented the mutual assured destruction HTLC that could mediate the bribery attack. Nevertheless, they introduce extra overhead and still require HTLC. Perun [40] enabled a user to create a virtual payment channel to avoid HTLC, but it can only span two ledger channels. In contrast, SPEEDSTER allows all nodes to connect directly without relying on HTLC and expensive on-chain operations.

Efficient Channel Network: Multi-hop transactions in existing channel networks [68, 84] incur non-negligible overhead and come with capacity and scalability issues. Current channel design addresses these problems with distinct focuses. MicroCash [2], for example, introduced the escrow setup that supports concurrent micropayments. Sprites [78] is built on LN and reduced LN latency in multi-hop transactions. Celer [38] leveraged a provably optimal value transfer routing algorithm to improve HTLC routing performance. Pisa [74] enabled parties to delegate a third party manager in case routing goes off-line. REVIVE [60] rebalances channel funds to increase the scalability of its payment channel network. Liquidity Network [45, 61] used hubs to connect users, which raises privacy and centralization concerns. SPEEDSTER, in contrast, is an account-based peer-to-peer channel network, and outperforms existing channel networks in various ways.
Multi-Party Channel Network: Several related works offer multi-party payment/state channel solutions. Based on Perun, Dziembowski et al. proposed the first multi-party state channel [39] that operates recursively among participants. Burchert et al. [24] presented a multi-party channel with timelocks by adding a new layer between the Blockchain and the payment channel. Hydra [28] introduced an isomorphic multi-party state channel by directly adopting the layer-1 smart contract system. Speedster establishes multi-party channels directly between participants without intermediaries, thus reducing costs and enhancing security.

Blockchain projects based on trusted hardware: Using trusted hardware provides promising solutions to Blockchain issues. For instance, Town Crier [104] used SGX to implement an authenticated data feed for smart contracts. Ekiden [50], PrivacyGuard [106], and FastKitten [36] proposed Blockchain projects that aims to elevate the confidentiality of smart contracts. In Tesseract [14], credits could be exchanged across multiple chains. Obscuro [91] built a privacy-preserving Bitcoin mixer. For layer-2 compatibility, TeeChan [65] was built on top of the Lightning network and instantly created new off-chain channels. However, it still requires synchronization with Blockchain and cannot create multiple channels with a single deposit. Based on TeeChan, TeeChain [66] was proposed to set up a committee for each node and dynamically allocate deposits to channels, but it is a payment channel system and still requires HTLC for multi-hop transactions. In contrast, Speedster provides better privacy protection via peer-to-peer decentralized.

10 CONCLUSION

Speedster is the first account-based state channel system, where off-chain channels can be freely opened/closed using the existing account balance without involving Blockchain. Speedster introduces Certified Channel to eliminate the expensive operations for transaction processing and dispute resolution. To the best of our knowledge, Speedster is the first channel system that achieves P2PCN, thus eliminating the risks and overhead introduced by HTLC once for all. With the Certified Channel and P2PCN, Speedster is capable of executing multi-party state channel efficiently. The practicality of Speedster is validated on different TEE platforms (i.e., Intel SGX, AMD SEV, and ARM TrustZone). The experimental results show much-improved performance compared to LN and other layer-2 channel networks.

ACKNOWLEDGMENTS

This work was supported by the Technology and Innovation Commission of Shenzhen Municipality under Grant SGIX20201103095408029, National Natural Science Foundation of China under Grant 62002151, and US National Science Foundation under grant number CNS-1928331. In addition, we thank our shepherd Cris Staicu, and the anonymous reviewers for useful feedback.

REFERENCES


A IDEAL FUNCTIONALITY

In \( \Pi_{\text{speedster}} \), two ideal functionalities are assumed: a Blockchain abstraction \( F_{\text{Blockchain}}[\text{Contract}] \) and a TEE abstraction \( G_{\text{att}} \) formally defined in [83]. As a result, the design and security of \( \Pi_{\text{speedster}} \) are independent of the specific Blockchain and TEE implementations as long as they can provide the required functions. Specifically, we define \( F_{\text{Blockchain}}[\text{Contract}] \) as an ideal functionality that models the behavior of Blockchain. \( F_{\text{Blockchain}} \) defines a smart-contract enabled append-only ledger. The parameter \( \text{Contract} \) is the smart contract function of the Blockchain. \( F_{\text{Blockchain}} \) has an internal \( \text{Storage} \) that contains the Blockchain data associated with transaction IDs. To append a transaction to the Blockchain, a user sends a transaction to \( F_{\text{Blockchain}} \), which will subsequently trigger the function “append” to execute the transaction (see Figure 7 for details).

\( G_{\text{att}} \) [83] provides an abstraction for the general-purpose TEE-enabled secure processor. During initialization, \( G_{\text{att}} \) creates a key pair as the manufacture key (msk, mpk), while msk is preserved in the processor and the mpk could be accessed through \( \text{getmpk} \) command. In such an ideal functionality, user first creates an enclave, and loads \( \text{pro} \) into enclave by sending an “install” command. To call the functions in \( \text{pro} \)-enclave, user sends “resume” command to \( G_{\text{att}} \) along with the parameters. All operations through the “resume” command of \( G_{\text{att}} \) is signed with msk by default to ensure the authenticity, whereas \( \text{Certified Channel} \) leverages symmetric-key authenticated encryption instead of digital signatures. Therefore, we add a switch to “resume” command to be able to turn off the signature and only when the switch is set, execution output through “resume” is signed. (see Figure 10 in the Appendix for detail).

A.1 Ideal Functionality \( F_{\text{Speedster}} \)

The ideal functionality in Figure 6 defines the security goal of \( \Pi_{\text{speedster}} \) in the ideal functionality \( F_{\text{Speedster}} \). Participants of \( F_{\text{Speedster}} \) are denoted as \( P \). The internal communication among participants is protected through authenticated encryption scheme. Following [27] [30], we parameterize \( F_{\text{Speedster}} \) with a leakage function \( \ell(\cdot) : \{0, 1\}^* \to \{0, 1\}^* \) to demonstrate the amount of privacy leaked from the message that is encrypted by the authenticated encryption scheme.

B SECURITY PROOF FOR THEOREM. 1

As defined in Theorem. 1, we now formally present the proof that the protocol \( \Pi_{\text{Speedster}} \) securely UC-realizes ideal functionality \( F_{\text{Speedster}} \) by simulating the behavior of a real-world adversary \( A \) in an ideal world simulator \( S \). And the security of \( \Pi_{\text{Speedster}} \) is proved by showing that \( S \) could indistinguishably simulate the behavior of \( A \) for all environment \( E \) [27].

Proof. Let \( E \) be an environment and \( A \) be a real-world PPT adversary [27] who simply relays messages between \( E \) and dummy parties. To show that \( \Pi_{\text{Speedster}} \) UC-realizes \( F_{\text{Speedster}} \), we specify below a simulator \( S \) such that no environment can distinguish an interaction between \( \Pi_{\text{Speedster}} \) and \( A \) from an interaction with \( F_{\text{Speedster}} \) and \( S \). That is, for any \( E \), \( S \) should satisfy

\[ \forall E. \text{EXEC}_{E(\cdot, \text{Speedster})} \approx \text{EXEC}_{E(\cdot, \text{Speedster})} \]

B.1 Construction of \( S \)

\( S \) simulates \( A, F_{\text{Speedster}} \) internally. \( S \) forwards any input \( e \) from \( E \) to \( A \) and records the traffic going to and from \( A \).

(1) **Deposit**: If \( \rho \) is honest, \( S \) obtains message (“deposit”, \( tx, aux \)), and emulates a call of “deposit” to \( G_{\text{att}} \) through “resume” interface. Otherwise, \( S \) reads \( tx \) and \( aux \) from \( E \), then emulates message (“deposit”, \( tx, aux \)) to \( F_{\text{Speedster}} \) with the identity of \( \rho \), and sends the “deposit” call to \( G_{\text{att}} \).

(2) **Open Channel**: When \( \rho \) is honest, \( S \) emulates a call of “open” to \( G_{\text{att}} \) on receiving (“open”, \( cid, \rho \), \( inp \)) from \( F_{\text{Speedster}} \).
When $\varphi_i$ is corrupted:

- $S$ obtains a public key $pk$, and a smart contract id $cid$ from $e_i$, then generates a random string as $inp$. $S$ sends the message ("open", $cid$, $pk$, $inp$) to $\mathcal{F}_{\text{Speedster}}$ and collects the output with the identity of $\varphi_i$. Then $S$ emulates a "resume" call to $\mathcal{G}_{\text{att}}$ with the same messages ("open", $cid$, $pk$, $inp$) on behalf of $\varphi_i$ and collects the output from $\mathcal{G}_{\text{att}}$.
- Upon receiving ("open", $cid$, $\varphi_j$) from $\mathcal{F}_{\text{Speedster}}$, $S$ obtains $inp$ from $e_i$ and emulates a "resume" call to $\mathcal{G}_{\text{att}}$ sending message ("open", $cid$, $\varphi_j$) on behalf of $\varphi_i$ and record the output from $\mathcal{G}_{\text{att}}$.

(3) Channel Authentication: Upon receiving message ("authenticate", $ccid$, $\varphi_i$, $cert$) of an honest node $\varphi_i$, $S$ records $cert$. $S$ emulates a "resume" call to $\mathcal{G}_{\text{att}}$ sending message ("authenticate", $ccid$, $\varphi_i$, $cert$).

Then, $S$ sends an "OK" command to $\mathcal{F}_{\text{Speedster}}$.

If $\varphi_i$ is corrupted, $S$ obtains a public key $pk$, a channel id $ccid$ from $e_i$, a sk from a signature challenger $Ch$, then generates a random string as $m$. $S$ computes signature $\sigma := \Sigma.$Sig$(sk, m)$, then sends the message ("authenticate", $pk$, $ccid$, $pk[|m|\sigma]$) to $\mathcal{F}_{\text{Speedster}}$ and collects the output with the identity of $\varphi_i$. Then $S$ emulates a "resume" call to $\mathcal{G}_{\text{att}}$ with the same messages on behalf of $\varphi_i$ and collects the output from $\mathcal{G}_{\text{att}}$.

(4) Multi-party Channel: Upon receiving message ("openMulti", $cid$, $\{ccid\}$) of an honest $\varphi_i$, $S$ emulates a "resume" call to $\mathcal{G}_{\text{att}}$ sending message ("openMulti", $cid$, $\{ccid\}$). Then relay the output to $\varphi_i$.

While dealing with a corrupted party $\varphi_i$:

- $S$ queries a set of channel id $\{ccid\}$ and a smart contract id $cid$ from $e_i$. Then, $S$ sends the message ("openMulti", $cid$, $\{ccid\}$) to $\mathcal{F}_{\text{Speedster}}$ and collects the output with $\varphi_i$'s identity. Then $S$ emulates a "resume" call to $\mathcal{G}_{\text{att}}$ with the same messages on behalf of $\varphi_i$ and collects the output from $\mathcal{G}_{\text{att}}$.
- Upon receiving message ("openMulti", $cid$, $\{ccid\}$), $S$ emulates a "resume" call to $\mathcal{G}_{\text{att}}$ sending message ("openMulti", $cid$, $\{ccid\}$). Then relay the output to $\varphi_i$.

(5) Channel Transaction: Upon receiving the message ("send", $ccid$, $(\ell(msg))$ from $\mathcal{F}_{\text{Speedster}}$ of $\varphi_i$, $S$ requests a key from a challenger $Ch$ who generates $AE$ keys. $S$ generates a random string $r$, and computes $m := AE.$Enc$(key, r)$, of which $|m| = |\ell(msg)|$. $S$ emulates a "resume" call to $\mathcal{G}_{\text{att}}$ sending message ("receive", $ccid$, $m$) on behalf of $\varphi_i$. Then relay the output to $\varphi_i$.

While dealing with a corrupted party $\varphi_i$:

- $S$ queries a channel id $ccid$ and a random string $inp := \{0, 1\}^*$ from $e_i$. Then, $S$ sends the message ("send", $ccid$, $\{ccid\}$) to $\mathcal{F}_{\text{Speedster}}$ on $\varphi_i$'s behalf, and collects the output. Then $S$ emulates a "resume" call to $\mathcal{G}_{\text{att}}$ with the same messages on behalf of $\varphi_i$ and collects the output from $\mathcal{G}_{\text{att}}$.
- Upon receiving message ("send", $ccid$, $(\ell(msg))$) from $\mathcal{F}_{\text{Speedster}}$, $S$ requests a key from $Ch$. $S$ computes $m := AE.$Enc$(key, 0)$, of which $|m| = |\ell(msg)|$. $S$ emulates a "resume" call to $\mathcal{G}_{\text{att}}$ sending message ("receive", $ccid$, $m$) on behalf of $\varphi_i$. Then relay the output to $\varphi_i$.

(6) Claim: Upon receiving message ("claim", $tx$) of $\varphi_i$ from $\mathcal{F}_{\text{Speedster}}$, $S$ emulates a "resume" call to $\mathcal{G}_{\text{att}}$ sending message ("claim", $tx$) on behalf of $\varphi_i$. Then, and send "OK" to $\mathcal{F}_{\text{Speedster}}$, and relay the output to the Blockchain.

---

**Figure 6:** Ideal functionality of $\mathcal{F}_{\text{Speedster}}$.

While $\varphi_i$ is corrupted, $S$ sends message ("claim") to $\mathcal{F}_{\text{Speedster}}$ on behalf of $\varphi_i$ and collects the output. Then $S$ emulates a "resume" call to $\mathcal{G}_{\text{att}}$ with the same message on behalf of $\varphi_i$ and collects the output from $\mathcal{G}_{\text{att}}$, then relay the output to the Blockchain.

**B.2 Indistinguishability**

We show that the execution of the real-world and ideal-world is indistinguishable for all $\mathcal{E}$ from the view of a probabilistic polynomial-time adversary $\mathcal{A}$ by a series of hybrid steps that reduce the real-world execution to the ideal-world execution.

- Hybrid $H_0$ is the real-world execution of $\mathcal{E}$.
- Hybrid $H_1$ behaves the same as $H_0$ except that $S$ generates key pair $(sk, pk)$ for digital signature scheme $\Sigma$ for each dummy party $\varphi$ and publishes the public key $pk$. Whenever $\mathcal{A}$ wants to call $\mathcal{G}_{\text{att}}$, $S$ faithfully simulates the behavior of $\mathcal{G}_{\text{att}}$, and relay output to $\varphi_i$. Since $S$ perfectly simulates the protocol, $\mathcal{E}$ could not distinguish $H_0$ from $H_1$.
- Hybrid $H_2$ is similar to $H_1$ except that $S$ also simulates $\mathcal{F}_{\text{Blockchain}}$.

Whenever $\mathcal{A}$ wants to communicate with $\mathcal{F}_{\text{Blockchain}}$, $S$ emulates the behavior of $\mathcal{F}_{\text{Blockchain}}$ internally. $\mathcal{E}$ cannot distinguish between $H_2$ and $H_1$ as $S$ perfectly emulates the interaction between $\mathcal{A}$ and $\mathcal{F}_{\text{Blockchain}}$. 
Protocol $\Pi_{\text{Speedster}}\{P_0, P_1, P_2, ..., P_N\}$

**Protocol $\Pi_{\text{Speedster}}\{P_0, P_1, P_2, ..., P_N\}$**

![Diagram of on-chain smart contract contract $\Pi_{\text{Speedster}}$.](image)

**Figure 9:** On-chain smart contract contract $\Pi_{\text{Speedster}}$ of $\Pi_{\text{Speedster}}$.

**Figure 8:** Program $\text{progeclave}$ program of $\Pi_{\text{Speedster}}$

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{progeclave}$</td>
<td>Program to generate a secure channel key</td>
</tr>
<tr>
<td>$\text{Contract}$</td>
<td>On-chain contract for secure channels</td>
</tr>
<tr>
<td>$\text{Storage}$</td>
<td>Off-chain storage for secure channels</td>
</tr>
</tbody>
</table>

**Figure 10:** Global functionality of TEE secure processor.

- Hybrid $H_3$ behaves the same as $H_2$ except that: If $A$ invokes $G_{\text{att}}$ with a correct install message with program $\text{progeclave}$, then for every correct "resume" message, $S$ records the tuple $(\text{outp}, \text{sigma})$ from $G_{\text{att}}$.
- $\Pi_{\text{Speedster}}$ and $A$ with $S$ and $T$.

On receive $\text{getpk}()$ from some $P$: send mpk to $P$