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

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

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

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:
Jinghui Liao, Fengwei Zhang, Wenhai Sun, and Weisong Shi. 2022. SPEEDSTER: An Efficient Multi-party State Channel via Enclaves. In *Proceedings of the 2022 ACM Asia Conference on Computer and Communications Security (ASIA CCS '22), May 30–June 3, 2022, Nagasaki, Japan.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3488932.3523259

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.

*Fengwei Zhang is the corresponding author

ASIA CCS '22, May 30–June 3, 2022, Nagasaki, Japan.
© 2022 Association for Computing Machinery.
ACM ISBN 978-1-4503-9140-5/22/05…$15.00
https://doi.org/10.1145/3488932.3523259

(**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, allieviating 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 decentralization 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

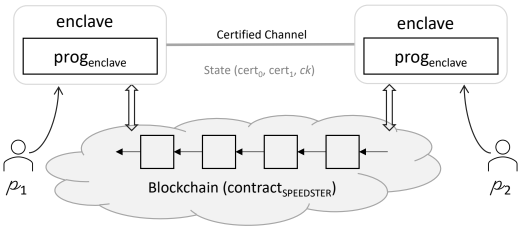

Figure 1: Framework of Speedster. A channel is opened directly between enclaves of two users. Off-chain transactions are processed by prog$_{enclave}$ in the enclave. The contract$_{\text{SPEEDSTER}}$ is deployed on the Blockchain to record the states of the nodes. The initial state of the enclave is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

Speedster contains two components: the state channel core program prog$_{enclave}$ executed inside the enclave and the on-chain smart contract contract$_{\text{SPEEDSTER}}$ 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).

Prog$_{enclave}$. The program that operates inside the enclave is referred to as prog$_{enclave}$. prog$_{enclave}$ 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.

Contract$_{\text{SPEEDSTER}}$. contract$_{\text{SPEEDSTER}}$ 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.

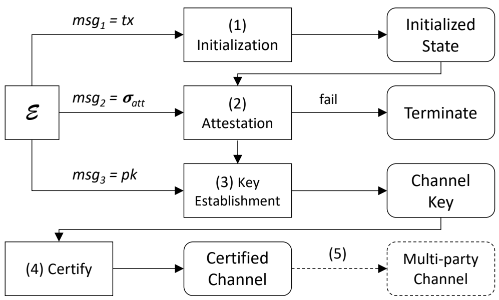

Figure 2: Workflow of node initialization and certified channel creation. $\mathcal{E}$ is the environment, including the Blockchain and the channel users, who pass input to Speedster nodes.

Node Initialization: When the program prog$_{enclave}$ is loaded into the enclave for the first time, an account acc$_{enclave}$ 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$_{enclave}$ on the Blockchain. To ensure the authenticity of the opened account acc$_{enclave}$ 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^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^0.

Enclave State Attestation: Step 2 is enclave attestation that needs to be carried out to authenticate the enclave environment including state^0. Note that we add the initial state state^0 and the public key pk into the enclave measurement $\sigma_{att} = \Sigma.\mathsf{Sig}(\mathsf{msk}, (\mathsf{prog}_{enclave}, \mathsf{pk}, \mathsf{state}^0))$ [1] where msk is the manufacturer-generated secret key for the processor [83]. The initial state reflects the starting point of acc$_{enclave}$, which should match the recorded state on the Blockchain. If a node passes the attestation, it means that the acc$_{enclave}$ is set up with the correct on-chain deposit and should be trusted for the subsequent off-chain transactions.

Channel Key Establishment: Once the enclave account is verified, the channel participants start to generate the shared channel key by leveraging any secure two-party key agreement protocols [15, 22].

Channel Certification: In this step, an identifier denoted as ccid := $\mathsf{H}(SORT(\{\mathsf{pk}_0, \mathsf{pk}_1\}))$ 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 $\mathsf{cert}_i := (\mathsf{pk}_{1-i} \| \mathsf{inp} \| \sigma_i)_{i \in \{0,1\}}$

[1] Specific implementation may vary depending on the underlying platform.

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 the channel can be unilaterally and securely closed without fear of unstable network connections. 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 := $\mathsf{H}(SORT(\{\mathsf{pk}_i\}^{i\in[N]}))$. 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.

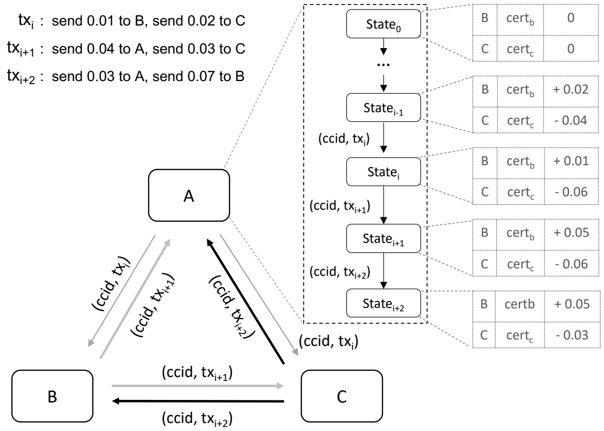

Figure 3: An example of executing a multi-party transfer contract among A, B and C, assuming SORT(pk_A)>SORT(pk_B)>SORT(pk_C). (+) and (-) in the tables represent the balance change after each respective *Certified Channel* transaction.

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 let each party i send their transactions in order as determined by SORT($\{\mathsf{pk}_i\}^{i\in[N]}$). Specifically, all nodes except for the one with the highest SORT 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 $\mathtt{tx}_i$, $\mathtt{tx}_{i+1}$, and $\mathtt{tx}_{i+2}$ successively through the multi-party state channel identified by a ccid. The 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 Channel*s 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 Channel*s.

5 $\Pi_{\text{Speedster}}$ PROTOCOL

5.1 Speedster Protocol $\Pi_{\text{Speedster}}$

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

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

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

Certified Channel: Each certified channel in $\Pi_{\text{Speedster}}$ is identified by a channel ID ccid. A shared channel key ck is produced in this step. The certificate cert of the channel is created using the public keys of both parties. To prevent rollback attacks on σ_{att}, $\text{prog}_{enclave}$ generates a signature σ_{att} by signing the tuple (state^0, $\{\text{pk}_i\}^{i\in\{0,1\}}$, $\text{prog}_{enclave}$) for each channel after function *(3)* returns. The tuple is signed by the manufacture secret key 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 $\text{prog}_{enclave}$ by sending the message "openMulti" and passing a set of ccid 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 $\text{prog}_{enclave}$ via the "send" command through $\mathcal{G}_{att}.\text{resume}(\cdot)$ and passes the target ccid along with other necessary parameters in input inp. Then, $\text{prog}_{enclave}$ executes 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 pk, the new channel state state′, the input inp, and the output outp. Then, the transaction is encrypted with an authentication scheme,such as AES-GCM, using the channel key ck.

Claim: To claim the funds that $\mathcal{P}$ receives from the channel transactions, the node issues a "claim" call to function *(7)* of $\text{prog}_{enclave}$. $\text{prog}_{enclave}$ first freezes all two-party channels, and extracts all certs from those channels. The certs and the local node state state constitute the claim transaction tx. $\text{prog}_{enclave}$ then signs the tx with the private key sk of $\text{acc}_{enclave}$ and returns the signed transaction that is further forwarded by the node to the $\text{contract}_{\text{Speedster}}$. In the end, $\text{contract}_{\text{Speedster}}$ 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 $\mathcal{F}_{\text{Speedster}}$ (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 $\mathcal{AE}$ is* IND-CCA *secure and digital signature scheme Σ is EU-CMA secure, then the protocol $\Pi_{\text{Speedster}}$ securely UC-realizes the ideal functionality $\mathcal{F}_{\text{Speedster}}$ in the $(\mathcal{G}_{att}, \mathcal{F}_{blockchain})$-hybrid model for static adversaries.*

Proof. (Sketch) We prove that the protocol $\Pi_{\text{Speedster}}$ securely UC-realizes ideal functionality $\mathcal{F}_{\text{Speedster}}$ 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 $\mathcal{F}_{\text{Speedster}}$, 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 $\mathcal{F}_{\text{Speedster}}$ and $\mathcal{S}$. That is, for any $\mathcal{E}$, $\mathcal{S}$ satisfies

$$\forall \mathcal{E}.\text{EXEC}^{\mathcal{E}}_{\Pi_{\text{Speedster}},\, \mathcal{A}} \approx \text{EXEC}^{\mathcal{E}}_{\mathcal{F}_{\text{Speedster}},\, \mathcal{S}}$$

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 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, 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, $\text{prog}_{enclave}$ 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 $\text{prog}_{enclave}$ 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 certs 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 $\text{prog}_{enclave}$, 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.

Table 1: Code size in Speedster.

	Component	Code	LOC	Total(#)
Shared	eEVM [32]	C++	25.3k	25.3k
SGX/TrustZone	$\text{prog}_{enclave}$	C++	3.1k	5.4k
	other	C++	2.3k	
AMD SEV	$\text{prog}_{enclave}$	C++	3.7k	7.8k
	other	C++	4.1k	

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. $\text{prog}_{enclave}$ has $3.1k$ LOC in C++ for SGX/TrustZone and $3.7k$ LOC for AMD SEV. The $\text{contract}_{\text{Speedster}}$ 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.

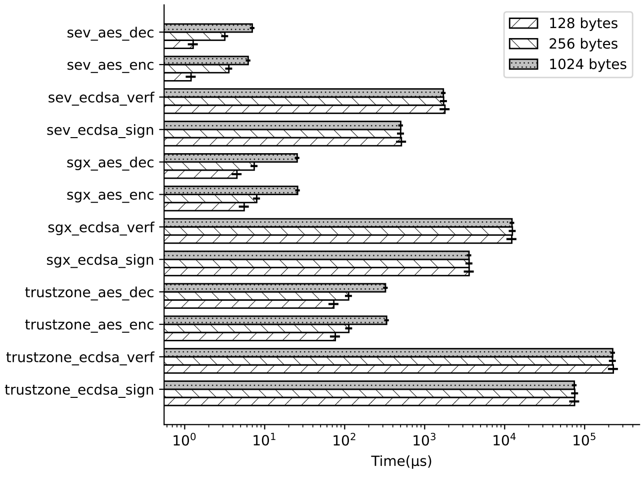

Figure 4: Performance comparison between ECDSA and AES-GCM enabled transaction security on SGX, SEV, and TrustZone platforms. We run every experiment $10,000$ times.

Figure 4 is plotted on a log scale. We can see that regardless of the tested platform, AES-GCM is $3-4$ orders of magnitude faster. Additionally, AES-GCM performs better with small-sized messages. With increased data size, the time cost of ECDSA remains constant while that of AES-GCM grows. This is because ECDSA always signs a constant hash digest rather than the actual data. In practice, the average transaction size on the Ethereum is 405 bytes [44]. Therefore, using symmetric-key operations will significantly boost transaction-related performance.

7.2.3 Transaction Performance. We evaluate Speedster on time costs for transactions in a direct channel on Intel, AMD, and ARM platforms under the test cases in Section 7.1. In this experiment, we use the popular layer-2 network, the LN, as a baseline. We measure the time cost for transactions over a direct channel, which may include the time cost for transaction generation and confirmation, corresponding contract execution, transmission in the local network, and other related activities in a life cycle of an off-chain transaction. We test Speedster in AES-GCM mode to reflect our intended symmetric-key design. Additionally, we also test the batching transaction performance to compare with that of TeeChain [66].

Table 2: Local time cost for end-to-end transaction (ms).

	Payment	ERC20	Gomuku	RPC
LN	192.630	N/A	N/A	N/A
SEV:AES-GCM	0.1372	0.1382	0.6667	0.1365
SGX:AES-GCM	0.0205	0.3500	0.4500	0.1930
TZ:AES-GCM	20.496	40.148	95.092	37.215

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

Evaluation on SGX: Evaluation of Speedster on the SGX platform is carried out by running two Speedster instances on the same SGX machine. Direct transaction without contract execution takes $0.0205ms$ with AES-GCM, which is four orders of magnitude faster compared to LN. It takes $0.1930ms - 0.4500ms$ to process a contract-calling transaction.

Evaluation on AMD: As no AMD cloud virtual machine supports SEV, we only evaluate Speedster on the AMD platform by running the $\text{prog}_{enclave}$ in two Ubuntu guest virtual machines as the enclaves. To protect the code and data of $\text{prog}_{enclave}$ that runs in the enclave, we only allow users to access $\text{prog}_{enclave}$ by calling the related interface through the socket.

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

Evaluation on ARM: 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 $\text{prog}_{enclave}$ takes $20.496ms$ to run direct transactions. For smart contract execution, it typically takes $30-90ms$ to process contract transactions.

Real-world Evaluation: To evaluate the performance of Speedster in the real world, we deploy Speedster on two Azure Standard DC1s_v2 (1 vCPUs, 4 GB memory) virtual machines, which are backed by the 3.7GHz Intel XEON E-2288G processor, one in East US, and the other in West Europe, as shown in 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

Table 3: Channel performance.

	LN (lnd)	**Speedster**			
	Payment		ERC20	RPC	Gomoku
Throughput (tps)	14 ±9 %	72,143 ±4 %	30,920 ±10 %	53,355 ±7 %	2,549 ±15 %
Latency (ms)	548.183 ±7 %	80.483 ±1 %	82.490 ±1 %	80.743 ±1 %	82.866 ±1 %

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, 000× 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.

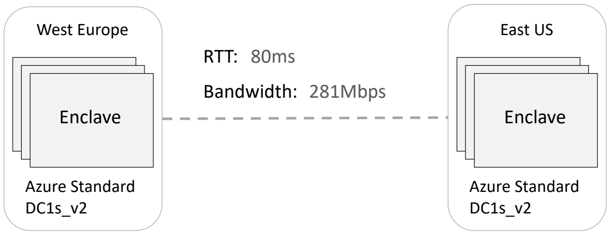

Figure 5: Network setup for the evaluation.

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,

[2] Though TeeChain is open source, we were not able to successfully run the project even after we contacted the author of TeeChain.

[3] Each fund spending needs to be approved by the committee using a multi-signature.

Table 4: Feature comparison with other channel projects.

Features	Channel Projects							
	LN [68]	TeeChan [65]	TeeChain [66]	DMC [37]	SFMC [24]	Perun [40]	Celer [38]	Speedster
Direct Off-chain Channel Open	✗	✓	✓	✗	✓	✓	✗	✓
Direct Off-chain Channel Close	✗	✗	✓	✗	✓	✓	✗	✓
Dynamic Deposit	✗	✗	✓	✗	✓	✗	✗	✓
Off-Chain Contract Execution	✗	✗	✗	✗	✗	✗	✓	✓
P2PCN	✗	✗	✗	✗	✗	✗	✗	✓
Multi-Party State Channel	✗	✗	✗	✗	✓	✗	✗	✓
Dispute-Free	✗	✗	✓	✗	✗	✗	✗	✓
Duplex Channel	✗	✗	✓	✓	✓	✗	✗	✓

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

Payment Channel	Setup| Open| Close| Claim		Total	
	No.tx	BC	No.tx	BC
LN [68]	2|1|1|0	2|2|2|0	4	6
TeeChain [66]	1|0|0|1	1+p/2|0|0|1+p/2+m	2	2+p+m
DMC [37]	0|1|1|0	0|2|2|0	2	4
SFMC [24]	1/c|0|0|1/c	p/c|0|0|p/c	2/c	2p/c
Speedster	1/c|0|0|1/c	1/c|0|0|1/c	2/c	2/c

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 $c \geq 2$ 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 $c \geq 2$, and 97% if $c \geq 11$ [17]. Compared to TeeChain, Speedster reduces BC by over 84% when $c \geq 2$.

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 $\text{prog}_{enclave}$, 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 [30], 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 HTCL for multi-hop transactions. In contrast, Speedster provides better privacy protection via peer-to-peer decentralization.

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 SGDX20201103095408029, 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

[1] Ian Allison. 2016. Ethereum's Vitalik Buterin explains how state channels address privacy and scalability.

[2] Ghada Almashaqbeh, Allison Bishop, and Justin Cappos. 2019. Microcash: Practical concurrent processing of micropayments. *arXiv preprint arXiv:1911.08520* (2019).

[3] AMD. 2018. AMD ESE/AMD SEV. https://github.com/AMDESE/AMDSEV.

[4] AMD. 2020. AMD EPYC™ 7452. https://www.amd.com/en/products/cpu/amd-epyc-7452.

[5] AMD. 2020. Secure Encrypted Virtualization (SEV). https://developer.amd.com/sev/.

[6] Ittai Anati, Shay Gueron, Simon Johnson, and Vincent Scarlata. 2013. Innovative technology for CPU based attestation and sealing. In *Proceedings of the 2nd international workshop on hardware and architectural support for security and privacy*, Vol. 13.

[7] Elli Androulaki, Artem Barger, Vita Bortnikov, Christian Cachin, Konstantinos Christidis, Angelo De Caro, David Enyeart, Christopher Ferris, Gennady Laventman, Yacov Manevich, et al. 2018. Hyperledger fabric: a distributed operating system for permissioned blockchains. In *Proceedings of the Thirteenth EuroSys Conference*. 1–15.

[8] Apple. 2019. *Apple T2 Secure Chip.* https://support.apple.com/guide/security/secure-enclave-overview-sec59b0b31ff/1/web/1

[9] ARM. 2019-12-13. Arm TrustZone Technology. https://developer.arm.com/ip-products/security-ip/trustzone.

[10] ARM. 2021. Arm Confidential Compute Architecture. https://developer.arm.com/architectures/architecture-security-features. accessed: 2021-03-31.

[11] Christian Badertscher, Ueli Maurer, Daniel Tschudi, and Vassilis Zikas. 2017. Bitcoin as a transaction ledger: A composable treatment. In *Annual International Cryptology Conference*. Springer, 324–356.

[12] Rana Barua, Ratna Dutta, and Palash Sarkar. 2003. Extending Joux's protocol to multi party key agreement. In *International Conference on Cryptology in India*. Springer, 205–217.

[13] Gal Beniamini. 2017. Trust Issues: Exploiting TrustZone TEEs. *Google Project Zero Blog* (2017).

[14] Iddo Bentov, Yan Ji, Fan Zhang, Lorenz Breidenbach, Philip Daian, and Ari Juels. 2019. Tesseract: Real-time cryptocurrency exchange using trusted hardware. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 1521–1538.

[15] Daniel J Bernstein. 2006. Curve25519: new Diffie-Hellman speed records. In *International Workshop on Public Key Cryptography*. Springer, 207–228.

[16] GP Biswas. 2008. Diffie–Hellman technique: extended to multiple two-party keys and one multi-party key. *IET Information Security* 2, 1 (2008), 12–18.

[17] bitcoinvisuals.com. 2021. Average Channels per Node. https://bitcoinvisuals.com/lightning.

[18] blockchain.com. 2021. Average Block Size. https://www.blockchain.com/charts/avg-block-size.

[19] blockchain.com. 2021. Bitcoin Transaction Rate. https://www.blockchain.com/en/charts/transactions-per-second?timespan=all.

[20] Dan Boneh and Mark Zhandry. 2017. Multiparty key exchange, efficient traitor tracing, and more from indistinguishability obfuscation. *Algorithmica* 79, 4 (2017), 1233–1285.

[21] Marcus Brandenburger, Christian Cachin, Matthias Lorenz, and Rüdiger Kapitza. 2017. Rollback and forking detection for trusted execution environments using lightweight collective memory. In *2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 157–168.

[22] Emmanuel Bresson, Olivier Chevassut, and David Pointcheval. 2007. Provably secure authenticated group Diffie-Hellman key exchange. *ACM Transactions on Information and System Security (TISSEC)* 10, 3 (2007), 10–es.

[23] Robert Buhren, Christian Werling, and Jean-Pierre Seifert. 2019. Insecure Until Proven Updated: Analyzing AMD SEV's Remote Attestation. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. 1087–1099.

[24] Conrad Burchert, Christian Decker, and Roger Wattenhofer. 2018. Scalable funding of Bitcoin micropayment channel networks. *Royal Society open science* 5, 8 (2018), 180089.

[25] Vitalik Buterin et al. 2014. Ethereum: A next-generation smart contract and decentralized application platform. *URL https://github. com/ethereum/wiki/wiki/% 5BEnglish% 5D-White-Paper* (2014).

[26] Vitalik Buterin et al. 2014. A next-generation smart contract and decentralized application platform. *white paper* (2014).

[27] Ran Canetti. 2001. Universally composable security: A new paradigm for cryptographic protocols. In *Proceedings 42nd IEEE Symposium on Foundations of Computer Science*. IEEE, 136–145.

[28] Manuel MT Chakravarty, Sandro Coretti, Matthias Fitzi, Peter Gazi, Philipp Kant, Aggelos Kiayias, and Alexander Russell. 2020. Hydra: Fast Isomorphic State Channels. *IACR Cryptol. ePrint Arch.* 2020 (2020), 299.

[29] Guoxing Chen, Yinqian Zhang, and Ten-Hwang Lai. 2019. Opera: Open remote attestation for intel's secure enclaves. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. 2317–2331.

[30] Raymond Cheng, Fan Zhang, Jernej Kos, Warren He, Nicholas Hynes, Noah Johnson, Ari Juels, Andrew Miller, and Dawn Song. 2019. Ekiden: A platform for confidentiality-preserving, trustworthy, and performant smart contracts. In *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 185–200.

[31] Tom Close. 2019. Nitro Protocol. *IACR Cryptology ePrint Archive* 2019 (2019), 219.

[32] Microsoft Corporation. 2019. EVM. https://github.com/microsoft/eEVM.

[33] Microsoft Corporation. 2019. openenclave. https://github.com/microsoft/openenclave.

[34] Victor Costan and Srinivas Devadas. 2016. Intel SGX Explained. *IACR Cryptology ePrint Archive* 2016, 086 (2016), 1–118.

[35] Chris Dannen. 2017. *Introducing Ethereum and solidity*. Vol. 318. Springer.

[36] Poulami Das, Lisa Eckey, Tommaso Frassetto, David Gens, Kristina Hostáková, Patrick Jauernig, Sebastian Faust, and Ahmad-Reza Sadeghi. 2019. FastKitten: practical smart contracts on bitcoin. In *28th USENIX Security Symposium (USENIX Security 19)*. 801–818.

[37] Christian Decker and Roger Wattenhofer. 2015. A fast and scalable payment network with bitcoin duplex micropayment channels. In *Symposium on Self-Stabilizing Systems*. Springer, 3–18.

[38] Mo Dong, Qingkai Liang, Xiaozhou Li, and Junda Liu. 2018. Celer Network: Bring Internet Scale to Every Blockchain. *arXiv preprint arXiv:1810.00037* (2018).

[39] Stefan Dziembowski, Lisa Eckey, Sebastian Faust, Julia Hesse, and Kristina Hostáková. 2019. Multi-party virtual state channels. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 625–656.

[40] Stefan Dziembowski, Lisa Eckey, Sebastian Faust, and Daniel Malinowski. 2019. Perun: Virtual payment hubs over cryptocurrencies. In *2019 IEEE Symposium on Security and Privacy (SP)*. 327–344.

[41] Stefan Dziembowski, Sebastian Faust, and Kristina Hostakova. 2018. Foundations of State Channel Networks. *IACR Cryptology ePrint Archive* 2018 (2018), 320.

[42] Stefan Dziembowski, Sebastian Faust, and Kristina Hostáková. 2018. General state channel networks. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 949–966.

[43] ethereum. 2020-05-02. Ethereum Virtual Machine (EVM) Awesome List. https://github.com/ethereum/wiki/wiki/Ethereum-Virtual-Machine-(EVM)-Awesome-List.

[44] etherscan.io. 2021. Ethereum Blockchain Size. https://etherscan.io/chartsync/chaindefault.

[45] Guillaume Felley, Arthur Gervais, and Roger Wattenhofer. 2018. Towards Usable Off-Chain Payments. (2018).

[46] William Foxley. 2019. As Bitcoin Cash Hard Forks, Unknown Mining Pool Continues Old Chain. *https://shorturl.at/svATX* (2019).

[47] Cesare Garlati. 2019. Multi Zone Trusted Execution Environment Free And Open API. In *RISC-V Workshop*.

[48] Arthur Gervais, Ghassan O Karame, Karl Wüst, Vasileios Glykantzis, Hubert Ritzdorf, and Srdjan Capkun. 2016. On the security and performance of proof of work blockchains. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*. 3–16.

[49] Johannes Götzfried, Moritz Eckert, Sebastian Schinzel, and Tilo Müller. 2017. Cache attacks on Intel SGX. In *Proceedings of the 10th European Workshop on Systems Security*. ACM, 2.

[50] Matthew Green and Ian Miers. 2017. Bolt: Anonymous payment channels for decentralized currencies. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 473–489.

[51] Lewis Gudgeon, Pedro Moreno-Sanchez, Stefanie Roos, Patrick McCorry, and Arthur Gervais. 2020. Sok: Layer-two blockchain protocols. In *International Conference on Financial Cryptography and Data Security*. Springer, 201–226.

[52] Jordi Herrera-Joancomarti, Guillermo Navarro-Arribas, Alejandro Ranchal Pedrosa, Perez-Sola Cristina, and Joaquin Garcia-Alfaro. 2019. *On the difficulty of hiding the balance of lightning network channels*. Ph. D. Dissertation. Dépt. Réseaux et Service de Télécom (Institut Mines-Télécom-Télécom SudParis ....

[53] Matthew Hoekstra, Reshma Lal, Pradeep Pappachan, Vinay Phegade, and Juan Del Cuvillo. 2013. Using innovative instructions to create trustworthy software solutions.. In *HASP@ ISCA*. 11.

[54] Intel. 2019. *Intel® Processors Voltage Settings Modification Advisory*. https://www.intel.com/content/www/us/en/security-center/advisory/intel-sa-00289.html

[55] Intel. 2019-12-3. Intel® Xeon® Processor E3 v5 Family. https://ark.intel.com/content/www/us/en/ark/products/88177/intel-xeon-processor-e3-1275-v5-8m-cache-3-60-ghz.html.

[56] Intel. 2020. Security Best Practices for Side Channel Resistance. https://software.intel.com/security-software-guidance/insights/security-best-practices-side-channel-resistance. accessed: 2020-08-18.

[57] Markus Jakobsson and Ari Juels. 1999. Proofs of work and bread pudding protocols. In *Secure Information Networks*. Springer, 258–272.

[58] Don Johnson, Alfred Menezes, and Scott Vanstone. 2001. The elliptic curve digital signature algorithm (ECDSA). *International journal of information security* 1, 1 (2001), 36–63.

[59] David Kaplan, Jeremy Powell, and Tom Woller. 2016. AMD memory encryption. *White paper* (2016).

[60] Rami Khalil and Arthur Gervais. 2017. Revive: Rebalancing off-blockchain payment networks. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 439–453.

[61] Rami Khalil, Arthur Gervais, and G Felley. 2018. NOCUST-A Non-Custodial 2nd-Layer Financial Intermediary. *IACR Cryptol. ePrint Arch.* 2018 (2018), 642.

[62] Christina Kim. 2019. Ethereum's Istanbul Upgrade Arrives Early, Causes Testnet Split. *https://shorturl.at/bEQ29* (2019).

[63] Ahmed Kosba, Andrew Miller, Elaine Shi, Zikai Wen, and Charalampos Papamanthou. 2016. Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In *2016 IEEE symposium on security and privacy (SP)*. IEEE, 839–858.

[64] Dayeol Lee, David Kohlbrenner, Shweta Shinde, Dawn Song, and Krste Asanović. 2019. Keystone: A framework for architecting tees. *arXiv preprint arXiv:1907.10119* (2019).

[65] Joshua Lind, Ittay Eyal, Peter Pietzuch, and Emin Gün Sirer. 2016. Teechan: Payment channels using trusted execution environments. *arXiv preprint arXiv:1612.07766* (2016).

[66] Joshua Lind, Oded Naor, Ittay Eyal, Florian Kelbert, Emin Gün Sirer, and Peter Pietzuch. 2019. Teechain: a secure payment network with asynchronous blockchain access. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*. 63–79.

[67] lnd. 2019. Lightning Network Daemon. https://github.com/lightningnetwork/lnd.

[68] loomx.io. 2017. Loom: A new architecture for a high performance blockchain. https://loomx.io/. Accessed: 2019-054-18.

[69] Loi Luu, Viswesh Narayanan, Chaodong Zheng, Kunal Baweja, Seth Gilbert, and Prateek Saxena. 2016. A secure sharding protocol for open blockchains. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 17–30.

[70] Giulio Malavolta, Pedro Moreno-Sanchez, Aniket Kate, Matteo Maffei, and Srivatsan Ravi. 2017. Concurrency and privacy with payment-channel networks. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 455–471.

[71] Giulio Malavolta, Pedro Moreno-Sanchez, Clara Schneidewind, Aniket Kate, and Matteo Maffei. 2019. Anonymous multi-hop locks for blockchain scalability and interoperability. In *Network and Distributed System Security Symposium (NDSS)*.

[72] Sinisa Matetic, Mansoor Ahmed, Kari Kostiainen, Aritra Dhar, David Sommer, Arthur Gervais, Ari Juels, and Srdjan Capkun. 2017. ROTE: Rollback protection for trusted execution. In *26th USENIX Security Symposium (USENIX Security'17)*. 1289–1306.

[73] mbed.org. 2017. mbedtls:An open source, portable, easy to use, readable and flexible SSL library. https://tls.mbed.org/. Accessed: 2019-12-3.

[74] Patrick McCorry, Surya Bakshi, Iddo Bentov, Sarah Meiklejohn, and Andrew Miller. 2019. Pisa: Arbitration outsourcing for state channels. In *Proceedings of the 1st ACM Conference on Advances in Financial Technologies*. ACM, 16–30.

[75] Patrick McCorry, Siamak F Shahandashti, and Feng Hao. 2017. A smart contract for boardroom voting with maximum voter privacy. In *International Conference on Financial Cryptography and Data Security*. Springer, 357–375.

[76] David McGrew and John Viega. 2004. The Galois/counter mode of operation (GCM). *Submission to NIST Modes of Operation Process* 20 (2004).

[77] Frank McKeen, Ilya Alexandrovich, Alex Berenzon, Carlos V Rozas, Hisham Shafi, Vedvyas Shanbhogue, and Uday R Savagaonkar. 2013. Innovative instructions and software model for isolated execution.. In *HASP@ISCA*. 10.

[78] Andrew Miller, Iddo Bentov, Surya Bakshi, Ranjit Kumaresan, and Patrick McCorry. 2019. Sprites and state channels: Payment networks that go faster than lightning. In *International Conference on Financial Cryptography and Data Security*. Springer, 508–526.

[79] Du Mingxiao, Ma Xiaofeng, Zhang Zhe, Wang Xiangwei, and Chen Qijun. 2017. A review on consensus algorithm of blockchain. In *2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. IEEE, 2567–2572.

[80] Kit Murdock, David Oswald, Flavio D Garcia, Jo Van Bulck, Daniel Gruss, and Frank Piessens. 2020. Plundervolt: Software-based Fault Injection Attacks against Intel SGX. In *2020 IEEE Symposium on Security and Privacy (SP)*.

[81] Satoshi Nakamoto. 2016. Bitcoin: A peer-to-peer electronic cash system. http://bitcoin.org/bitcoin.pdf.

[82] Zhenyu Ning and Fengwei Zhang. 2019. Understanding the security of arm debugging features. In *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, 602–619.

[83] Rafael Pass, Elaine Shi, and Florian Tramer. 2017. Formal abstractions for attested execution secure processors. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 260–289.

[84] Raiden. 2017. The Raiden Network. https://raiden.network/.

[85] Elias Rohrer, Julian Malliaris, and Florian Tschorsch. 2019. Discharged Payment Channels: Quantifying the Lightning Network's Resilience to Topology-Based Attacks. *arXiv preprint arXiv:1904.10253* (2019).

[86] Fahad Saleh. 2020. Blockchain without waste: Proof-of-stake. *Available at SSRN 3183935* (2020).

[87] Vinnie Scarlata, Simon Johnson, James Beaney, and Piotr Zmijewski. 2018. Supporting third party attestation for Intel SGX with Intel data center attestation

primitives. *White paper* (2018).

[88] Michael Schwarz, Samuel Weiser, Daniel Gruss, Clémentine Maurice, and Stefan Mangard. 2017. Malware guard extension: Using SGX to conceal cache attacks. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment.* Springer, 3–24.

[89] Tim Swanson. 2015. Consensus-as-a-service: a brief report on the emergence of permissioned, distributed ledger systems. *Report, available online* (2015).

[90] Saar Tochner, Stefan Schmid, and Aviv Zohar. 2019. Hijacking Routes in Payment Channel Networks: A Predictability Tradeoff. *arXiv preprint arXiv:1909.06890* (2019).

[91] Muoi Tran, Loi Luu, Min Suk Kang, Iddo Bentov, and Prateek Saxena. 2018. Obscuro: A bitcoin mixer using trusted execution environments. In *Proceedings of the 34th Annual Computer Security Applications Conference.* 692–701.

[92] Somanath Tripathy and Susil Kumar Mohanty. 2020. Mappcn: Multi-hop anonymous and privacy-preserving payment channel network. In *International Conference on Financial Cryptography and Data Security.* Springer, 481–495.

[93] Itay Tsabary, Matan Yechieli, and Ittay Eyal. 2020. MAD-HTLC: because HTLC is crazy-cheap to attack. *arXiv preprint arXiv:2006.12031* (2020).

[94] Andrew Urquhart. 2016. The inefficiency of Bitcoin. *Economics Letters* 148 (2016), 80–82.

[95] Jo Van Bulck, Marina Minkin, Ofir Weisse, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Thomas F Wenisch, Yuval Yarom, and Raoul Strackx. 2018. Foreshadow: Extracting the Keys to the Intel SGX Kingdom with Transient Out-of-Order Execution. In *27th USENIX Security Symposium (USENIX Security'18).* 991–1008.

[96] Fabian Vogelsteller and Vitalik Buterin. 2015. Erc-20 token standard. *Ethereum Foundation (Stiftung Ethereum), Zug, Switzerland* (2015).

[97] Nico Weichbrodt, Anil Kurmus, Peter Pietzuch, and Rüdiger Kapitza. 2016. AsyncShock: Exploiting synchronisation bugs in Intel SGX enclaves. In *European Symposium on Research in Computer Security.* Springer, 440–457.

[98] Karl Wüst and Arthur Gervais. 2018. Do you need a blockchain?. In *2018 Crypto Valley Conference on Blockchain Technology (CVCBT).* IEEE, 45–54.

[99] www.blockchain.com. 2020. Blockchain Size. https://www.blockchain.com/charts/blocks-size.

[100] www.blockchain.com. 2021. Average Confirmation Time. https://www.blockchain.com/charts/avg-confirmation-time?timespan=all&daysAverageString=7.

[101] Bin Yu, Shabnam Kasra Kermanshahi, Amin Sakzad, and Surya Nepal. 2019. Chameleon hash time-lock contract for privacy preserving payment channel networks. In *International Conference on Provable Security.* Springer, 303–318.

[102] Mahdi Zamani, Mahnush Movahedi, and Mariana Raykova. 2018. Rapidchain: Scaling blockchain via full sharding. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security.* 931–948.

[103] Fan Zhang. 2017. *mbedtls-SGX: a SGX-friendly TLS stack (ported from mbedtls).* https://github.com/bl4ck5un/mbedtls-SGX

[104] Fan Zhang, Ethan Cecchetti, Kyle Croman, Ari Juels, and Elaine Shi. 2016. Town crier: An authenticated data feed for smart contracts. In *Proceedings of the 2016 aCM sIGSAC conference on computer and communications security.* ACM, 270–282.

[105] Fan Zhang, Ittay Eyal, Robert Escriva, Ari Juels, and Robbert Van Renesse. 2017. REM: Resource-Efficient Mining for Blockchains. *IACR Cryptology ePrint Archive* 2017 (2017), 179.

[106] Ning Zhang, Jin Li, Wenjing Lou, and Y Thomas Hou. 2018. PrivacyGuard: Enforcing private data usage with blockchain and attested execution. In *Data Privacy Management, Cryptocurrencies and Blockchain Technology.* Springer, 345–353.

[107] Yuanyu Zhang, Shoji Kasahara, Yulong Shen, Xiaohong Jiang, and Jianxiong Wan. 2018. Smart contract-based access control for the internet of things. *IEEE Internet of Things Journal* 6, 2 (2018), 1594–1605.

A IDEAL FUNCTIONALITY

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

$\mathcal{G}_{att}$ [83] provides an abstraction for the general-purpose TEE-enabled secure processor. During initialization, $\mathcal{G}_{att}$ creates a key pair as the manufacture key (msk, mpk), while msk is preserved in the processor and the mpk could be accessed through "getpk" command. In such an ideal functionality, user first creates an enclave, and loads $\mathsf{prog}_{enclave}$ into enclave by sending an "install" command. To call the functions in $\mathsf{prog}_{enclave}$, user sends "resume" command to $\mathcal{G}_{att}$ along with the parameters. All operations through the "resume" command of $\mathcal{G}_{att}$ is signed with msk by default to ensure the authenticity, whereas *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 $\mathcal{F}_{\text{SPEEDSTER}}$

The ideal functionality in Figure 6 defines the security goal of $\Pi_{\text{SPEEDSTER}}$ in the ideal functionality $\mathcal{F}_{\text{SPEEDSTER}}$. Participants of $\mathcal{F}_{\text{SPEEDSTER}}$ are denoted as $\mathcal{P}$. The internal communication among participants is protected through authenticated encryption scheme. Following [27] [30], we parameterize $\mathcal{F}_{\text{SPEEDSTER}}$ 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 $\mathcal{F}_{\text{SPEEDSTER}}$ by simulating the behavior of a real-world adversary $\mathcal{A}$ in an ideal world simulator $\mathcal{S}$. And the security of $\Pi_{\text{SPEEDSTER}}$ is proved by showing that $\mathcal{S}$ could indistinguishably simulate the behavior of $\mathcal{A}$ for all environment $\mathcal{E}$ [27].

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

$$\forall \mathcal{E}.\mathrm{EXEC}^{\mathcal{E}}_{\Pi_{\text{SPEEDSTER}},\,\mathcal{A}} \approx \mathrm{EXEC}^{\mathcal{E}}_{\mathcal{F}_{\text{SPEEDSTER}},\,\mathcal{S}}$$

B.1 Construction of $\mathcal{S}$

$\mathcal{S}$ simulates $\mathcal{A}$, $\mathcal{F}_{\text{SPEEDSTER}}$ internally. $\mathcal{S}$ forwards any input e from $\mathcal{E}$ to $\mathcal{A}$ and records the traffic going to and from $\mathcal{A}$.

(1) *Deposit:* If $\mathcal{P}_i$ is honest, $\mathcal{S}$ obtains message ("deposit", tx, aux), and emulates a call of "deposit" to $\mathcal{G}_{att}$ through "resume" interface. Otherwise, $\mathcal{S}$ reads tx and aux from $\mathcal{E}$, then emulates message ("deposit", tx, aux) to $\mathcal{F}_{\text{SPEEDSTER}}$ with the identity of $\mathcal{P}_i$ and sends the "deposit" call to $\mathcal{G}_{att}$.

(2) *Open Channel:* When $\mathcal{P}_i$ is honest, $\mathcal{S}$ emulates a call of "open" to $\mathcal{G}_{att}$ on receiving ("open", cid, $\mathcal{P}_j$, inp) from $\mathcal{F}_{\text{SPEEDSTER}}$.

When $\mathcal{P}_i$ is corrupted:

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

(3) *Channel Authentication:* Upon receiving message ("authenticate", ccid, $\mathcal{P}_j$, cert) of an honest node $\mathcal{P}_i$, $\mathcal{S}$ records cert. $\mathcal{S}$ emulates a "resume" call to $\mathcal{G}_{att}$ sending message ("authenticate", ccid, $\mathcal{P}_j$, cert). Then, $\mathcal{S}$ sends an "OK" command to $\mathcal{F}_{\text{Speedster}}$.

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

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

While dealing with a corrupted party $\mathcal{P}_i$:

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

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

While dealing with a corrupted party $\mathcal{P}_i$:

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

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

```
$\mathcal{F}_{\text{Speedster}}[\ell, \mathcal{P}_0, \mathcal{P}_1, \mathcal{P}_2, ..., \mathcal{P}_N]$
Initially:
bals := $\emptyset$, certs := $\emptyset$, channels:= $\emptyset$, states$^0$ := $\emptyset$
For each $\mathcal{P}_i$: $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow\$ \Sigma.\mathsf{KGen}(1^n)$
(1) On receive ("deposit", tx, aux) from $\mathcal{P}_i$ where $i \in [N]$:
        parse tx as (pk', $val, _, $\sigma$) // _ means unused value
        Verify signature of tx, abort if false
        bals[$\mathcal{P}_i$] += $val
        append (tx, aux) to states$^0[\mathcal{P}_i]$
        leak ("deposit", tx) to $\mathcal{A}$
(2) On receive ("open", cid, $\mathcal{P}_j$, inp) from $\mathcal{P}_i$ where $i, j \in [N]$ and $\mathcal{P}_i \neq \mathcal{P}_j$:
        ccid $\leftarrow\$ \{0,1\}^*$
        $\mathsf{state}_{\mathsf{ccid}} := \mathsf{Contract}_{\mathsf{cid}}(\mathsf{pk}_i, \overrightarrow{0}, \perp)$
        append (ccid, ( cid, $\mathsf{state}_{\mathsf{ccid}}$, $\{\mathcal{P}_j, \mathcal{P}_i\}$)) to channels
        leak ("open", ccid, cid, $\mathcal{P}_i$, $\mathcal{P}_j$, inp) to $\mathcal{A}$
(3) On receive ("authenticate", ccid, $\mathcal{P}_j$, cert) from $\mathcal{P}_i$ where $i, j \in [N]$ and
$i \neq j$:
        assert certs[ccid][$\mathcal{P}_i$] = $\perp$
        certs[ccid][$\mathcal{P}_i$] := cert
        leak ($\mathcal{P}_i$, $\mathcal{P}_j$, "authenticate", cert) to $\mathcal{A}$;
        await "OK" from $\mathcal{A}$
        send("authenticate", cert) to $\mathcal{P}_j$
(4) On receive ("openMulti", cid, {ccid}$^*$) from $\mathcal{P}_i$ where $i \in [N]$:
        ccid $\leftarrow\$ \{0,1\}^*$
        state := $\mathsf{Contract}_{\mathsf{cid}}(\mathcal{P}_i, \overrightarrow{0}, \perp)$
        collect dummy parties $\{\mathcal{P}\}^*$ in channels $\{\mathsf{ccid}\}^*$
        append (ccid, (cid, state, $\{\mathcal{P}\}^*$)) to certs
        leak ("openMulti", ccid, cid, {ccid}$^*$) to $\mathcal{A}$
(5) On receive ("send", ccid, inp) from $\mathcal{P}_i$ where $i \in [N]$:
        (cid, state, $\{\mathcal{P}\}^*$) = certs[ccid] abort if $\perp$
        (state', outp) := $\mathsf{Contract}_{cid}(\mathcal{P}_i, \mathsf{state}, inp)$
        msg := $(\mathcal{P}_i \| r \| \mathsf{inp} \| \mathsf{state}' \| \mathsf{outp})$
        leak ("send", ccid, $\ell(\mathsf{msg})$) to $\mathcal{A}$; await "OK" from $\mathcal{A}$
        send(msg) to each member of $\{\mathcal{P}\}^*$ except $\mathcal{P}_i$
(6) On receive ("claim") from $\mathcal{P}_i$ where i ∈ [N]:
        construct an on-chain claim transaction tx
        leak("claim", tx) to $\mathcal{A}$; await "OK" from $\mathcal{A}$
        append(tx) to Blockchain
```

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

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

• Hybrid H_1 behaves the same as H_0 except that $\mathcal{S}$ generates key pair (sk, pk) for digital signature scheme Σ for each dummy party $\mathcal{P}$ and publishes the public key pk. Whenever $\mathcal{A}$ wants to call $\mathcal{G}_{att}$, $\mathcal{S}$ faithfully simulates the behavior of $\mathcal{G}_{att}$, and relay output to $\mathcal{P}_i$. Since $\mathcal{S}$ perfectly simulates the protocol, $\mathcal{E}$ could not distinguish H_1 from H_0.

• Hybrid H_2 is similar to H_1 except that $\mathcal{S}$ also simulates $\mathcal{F}_{blockchain}$. Whenever $\mathcal{A}$ wants to communicate with $\mathcal{F}_{blockchain}$, $\mathcal{S}$ emulates the behavior of $\mathcal{F}_{blockchain}$ internally. $\mathcal{E}$ cannot distinguish between H_2 and H_1 as $\mathcal{S}$ perfectly emulates the interaction between $\mathcal{A}$ and $\mathcal{F}_{blockchain}$,

Protocol $\Pi_{\textsc{Speedster}}(\mathcal{P}_0, \mathcal{P}_1, \mathcal{P}_2, ..., \mathcal{P}_N]$

```
Program prog_{enclave}
Initially:
bal := ∅, certs := ∅, channels := ∅, $state^0$ := ⊥
(1) On receive("init")
    (pk, sk) ←$ $\Sigma$.KGen($1^n$) // generate $acc_{enclave}$
    mpk := $\mathcal{G}_{att}$.getpk()
    return (pk, mpk)
(2) On receive ("deposit", tx, aux)
    parse tx as (_, pk', $\$val$, $\sigma$) // _ represents unused value
    assert $val ≥ 0; assert $\Sigma$.Vf(pk, tx)
    bal += $val; add (tx, aux) to $state^0$
(3) On receive ("open", cid, $\mathcal{P}$, inp)
    ccid := H($SORT$\{$pk_{\mathcal{P}}$, pk\})
    abort if channels[ccid] ≠⊥
    ck ←$ $\{0,1\}^*$ // channel key
    cp := \{pk, $pk_{\mathcal{P}}$\}
    (state', outp) := $Contract_{cid}$(sk, bal, $\overrightarrow{0}$, cp)
    append (ccid, (ck, cid, state', cp)) to channels
    $\sigma$ = $\Sigma$.Sig(sk, $pk_{\mathcal{P}}$||inp||$state^0$); cert = ($pk_{\mathcal{P}}$||inp||$\sigma$)
    return (cert, $state^0$, outp)
(4) On receive ("openMulti", cid, {ccid}*)
    for each ccid'∈ \{ccid\}*:
        assert channels[ccid'] ≠⊥
        extract pk' from channels[ccid']
    cp := \{\{pk'\}* ∪ pk\}; ccid := H($SORT$(cp))
    assert channels[ccid] = ⊥, gk ←$ $\{0,1\}^*$ // Group key
    (state', outp) := $Contract_{cid}$($sk_i$, state, $\overrightarrow{0}$, cp)
    append (ccid, (gk, cid, state', cp)) to channels
    ct := Enc (gk, outp)
    return (ct)
(5) On receive ("authenticate", ccid, $\mathcal{P}$, cert)
    abort if certs[ccid][$pk_{\mathcal{P}}$] ≠⊥
    parse cert as (msg, $\sigma$), $\Sigma$.Vf($pk_{\mathcal{P}}$, msg, $\sigma$)
    extract $state^0_{\mathcal{P}}$ from msg, check $state^0_R$ on Blockchain
    certs[ccid][$pk_{\mathcal{P}}$] := cert
(6) On receive ("send", ccid, inp):
    assert certs[ccid] ≠⊥
    (ck, cid, state, cp) := channles[ccid]
    (st´, outp) := $Contract_{cid}$(sk, state, state, inp)
    update channels[ccid] to (ck, cid, st´, cp)
    msg := (pk||inp||state'||outp); ct := Enc (ck, msg)
    return (ct)
(7) On receive ("claim")
    freeze send function
    tx := {cert}*||state; $\sigma$ := $\Sigma$.Sig(sk,tx)
    return (tx||$\sigma$)
```

Figure 8: $prog_{enclave}$ program of $\Pi_{\textsc{Speedster}}$

Protocol $\Pi_{\textsc{Speedster}}(\mathcal{P}_0, \mathcal{P}_1, \mathcal{P}_2, ..., \mathcal{P}_N]$

```
contract_{Speedster}
Parameters:
$Ledger$ : Append only public ledger of $\mathcal{F}_{blockchain}$
$Coin$ : Blockchain function that convert value into coins.

On receive ("deposit", tx) from $\mathcal{P}$:
  assert tx ∉ $Ledger$
  execute tx on the $Blockchain$
  append tx to $Ledger$
On receive ("claim", tx) from $\mathcal{P}$:
  parse tx as ({cert}*, state) // state contains channel data
  For each cert in {cert}*:
      parse cert to (to´, from´, $\sigma$ )
      abort if Verify($\sigma$, cert) fails // verify the cert
      extract $\$val$ from state[from]
      assert $\$val \neq 0$ and to´ = $\mathcal{P}$
      send(from, $\mathcal{P}$, Coin($\$val$)) if $\$val > 0$
      send($\mathcal{P}$, from, Coin($-\$val$)) otherwise
  append(tx) to $Ledger$
On receive ("read", tx) from $\mathcal{P}$:
  output $Ledger$[tx]
```

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

```
$\mathcal{G}_{att}[\Sigma, reg]$
On initialize: (mpk, msk) := $\Sigma$.KGen($1^n$); $T = \emptyset$
On receive* getpk() from some $\mathcal{P}$: send mpk to $\mathcal{P}$
------------------------------------------------------------
Enclave operations
On receive* install($idx$, prog) from some $\mathcal{P} \in reg$:
  if $\mathcal{P}$ is honest, assert $idx = sid$ generate nonce $eid \in \{0,1\}^\lambda$,
  store $T[eid, \mathcal{P}]$ := ($idx$, prog, 0), send $eid$ to $\mathcal{P}$
On receive* resume($eid$, $inp$, $switch$ := $on$) from some $\mathcal{P} \in reg$:
  let ($idx$, prog, $mem$) := $T[eid, \mathcal{P}]$, abort if not found
  let ($outp$, $mem$) := prog($inp$, $mem$),
  update $T[eid, \mathcal{P}]$ := ($idx$, prog, $mem$)
  if $switch$ is set to $on$
      let $\sigma$ := $\Sigma.\mathrm{Sig}_{\mathrm{msk}}$($idx$, $eid$, prog, $outp$); send ($outp$, $\sigma$) to $\mathcal{P}$
  otherwise: send ($outp$, ⊥) to $\mathcal{P}$
```

Figure 10: Global functionality of TEE secure processor.

- Hybrid H_3 behaves the same as H_2 except that: If $\mathcal{A}$ invokes $\mathcal{G}_{att}$ with a correct install message with program $prog_{enclave}$, then for every correct "resume" message, $\mathcal{S}$ records the tuple (outp, σ) from $\mathcal{G}_{att}$, where outp is the output of running $prog_{enclave}$ in $\mathcal{G}_{att}$, and σ is the signature generated inside the $\mathcal{G}_{att}$, using the sk generated in H_1. Let Ω denote all such possible tuples. If (outp, σ) $\notin \Omega$ then $\mathcal{S}$ aborts, otherwise, $\mathcal{S}$ delivers the message to counterpart. H_3 is indistinguishable from H_2 by reducing the problem to the EUF-CMA of the digital signature scheme. If $\mathcal{A}$ does not send one of the correct tuples to the counterpart, it will fail on attestation. Otherwise, $\mathcal{E}$ and $\mathcal{A}$ can be leveraged to construct an adversary that succeeds in a signature forgery.
- Hybrid H_4 behaves the same as H_3 except that $\mathcal{S}$ generates a channel key ck for each channel. When $\mathcal{A}$ communicates with $\mathcal{G}_{att}$ on sending transaction through channel, $\mathcal{S}$ records ct from $\mathcal{G}_{att}$, where ct is the ciphertext of encrypted transaction, using the ck of that channel. Let Ω denote all such possible strings. If ct $\notin \Omega$ then $\mathcal{S}$ aborts, otherwise, $\mathcal{S}$ delivers the message to counterpart. H_4 is indistinguishable from H_3 by reducing the problem to the IND-CCA of the authenticated encryption scheme. As $\mathcal{A}$ does not hold control of ck, it can not distinguish the encryption of a random string and Ω.
- Hybrid H_5 is the execution in the ideal-world. H_5 is similar to H_4 except that $\mathcal{S}$ emulates all real-world operations. As we discussed above, $\mathcal{S}$ could faithfully map the real-world operations into ideal-world execution from the view of $\mathcal{A}$. Therefore, no $\mathcal{E}$ could distinguish the execution from the real-world protocol $\Pi_{\textsc{Speedster}}$ and $\mathcal{A}$ with $\mathcal{S}$ and $\mathcal{F}_{\textsc{Speedster}}$.

```
$\mathcal{F}_{blockchain}[Contract]$
On initialize: $Storage$ := ∅
On receive* read($id$) from $\mathcal{P}$: output $Storage[id]$, or ⊥ if not found
On receive* append(tx) from $\mathcal{P}$: abort if $Storage$[tx.$id$] ≠⊥
  if $Contract$(tx) = $true$ : $Storage$[tx.$id$] := tx; output ("success")
```

Figure 7: Ideal functionality of append-only ledger.