SPEEDSTER: A TEE-assisted State Channel System

***Abstract*—State channel network is the most studied second-layer solution to scalability problem, high transaction fees, and low transaction throughput for public Blockchain networks. However, the existing work has many practical concerns, such as expensive channel creation and close, strict synchronization between the main chain and off-chain channels, frozen deposits, and multi-party participation.**

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 certified channels that can be operated efficiently off the Blockchain without dispute concerns. SPEEDSTER is fully decentralized and enjoys better privacy protection. The system supports fast native multi-party contract execution. In comparison, the prototype systemjinghui:~~has no off-chain fee cost and~~ shows about $10,000\times$ faster transaction speed than Lighting Network and jinghui:~~two orders faster than other channel projects that also use TEE~~$50\times$ faster than TeeChain under batching mode. SPEEDSTER generates 97% less on-chain data in a scale comparable with Lightning Network.

I. INTRODUCTION

Blockchain has been deemed a disruptive technology to construct decentralized trust and foster a great number of novel applications in both the public and private sectors. However, scalability has become a great concern with the ever-growing adoption of the decentralized infrastructure. For example, Bitcoin network [74] can only handle approximately $3,500$ transactions in every new block due to the block size limitation [15] and process 7 transactions per second (tps) on average [16], [74]. The scalability issue has also haunted other major Blockchain networks based on a similar design principle, such as Ethereum [21]. Modifying the on-chain protocols may help alleviate the problem, for instance, using alternative consensus algorithms [72], improving the information propagation [61], [90]. However, any changes at layer one will inevitably affect the existing participants with an undesired cost [38], [54]. Shifting to layer two, payment channels [20], [60], [71], [77] can carry out micropayment transactions off the Blockchain to avoid unpleasant on-chain overhead. State channels [1], [35], [71], further advancing this off-chain innovation, enable stateful transactions, and off-chain smart contract execution. While state channels have significantly improved the performance of Blockchain, they suffer from the following limitations as well:

(1) To create a new channel or close an existing channel, one transaction that records the deposits of each participant has to be committed to the main chain. The user is charged with a transaction fee and needs to wait for the confirmation of the transaction by the Blockchain [33], [57], [71]. Therefore, it is expensive to open/close channels under the current off-chain network design.

(2) Current dispute resolving protocol in state channel uses the Blockchain as an arbiter [60] and allows for unilateral channel closing but leaves a time window for possible dispute filing. However, this mechanism is vulnerable to the denial-of-service (DoS) attack where one channel participant can maliciously sends an outdated channel state to the Blockchain while DoS-ing its counterpart to prevent the victim from submitting the dispute transaction that contains the lasted channel state to the Blockchain.

(3) Albeit the Hashed Timelock Contract (HTLC) enables multi-hop transactions and significantly reduces the architectural complexity, it also raises privacy concerns [33], [42], [43], [62] and leads to a multitude of attacks, such as wormhole attacks [63] and DoS attacks [53], [83].

(4) Despite the ambition of processing off-chain transactions instantly [60], the actual latency of state channels is very high and the throughput is only tens of tps [33], [58], [71]. The complex routing process and state updating mechanism gives rise to non-negligible overhead, thus considerably degrading promised performance.

(5) The exchange of the state information is confined to a pairwise channel, which fundamentally hinders the creation and execution of multi-party smart contracts. Though a multi-party state channel can be recursively established using the virtual channel techniques [25], [32], [33], it is still expensive to set up and run the protocol.

We present SPEEDSTER to address the above limitations. SPEEDSTER innovatively certifies the transactional activities in a channel and protects the channel in the enclave, an instance of a trusted execution environment (TEE). In general, SPEEDSTER is a state channel system, where a channel can be opened and closed freely among multiple participants off the main chain by bridging the on-chain trust – the Blockchain the off-chain trust – secure enclave. The main idea of SPEEDSTER is that every user creates an account in the enclave and makes an initial deposit through Blockchain. Afterward, channels can be freely opened between different enclave accounts. Relying on enclave trust also enables lightweight protocol design for channel confidentiality, authenticity, finalization, and dispute resolution.

TECHNICAL CONTRIBUTIONS. SPEEDSTER has a clear and efficient system design that significantly outperforms the conventional state channel networks in terms of security, performance, and functionality. SPEEDSTER effectively shifts the costly on-chain trust to the hardware-assisted TEEs in the off-chain network; layer-2 transactions and other chain functions can be promptly processed with minimum interaction with the main chain. In SPEEDSTER, channel members certify the

channel environment before taking any transactional activity. As such, expensive operations in prior work for channel authenticity can be replaced by lightweight cryptographic functions, thus increasing the channel efficiency. Unlike other state channels, SPEEDSTER is an account-based channel system, it supports multi-party state channel by nature, and nodes could create direct channel connections freely without touching the Blockchain. The potential settlement dispute is completely avoided, as well.

In SPEEDSTER, it is unnecessary to submit a transaction to the Blockchain every time to open/close a channel. Instead, only one transaction is needed to initialize a TEE-enabled account for each off-chain participant. Later, a participating node can create/close an arbitrary number of channels with any other nodes by direct communicationjinghui:with out freezing any deposit. Therefore, SPEEDSTER enables instant channel creation and close completely off the main chain.

SPEEDSTER adopts a novel certificate-based off-chain transaction processing mechanism where the channel state is retained in the secure enclave. SPEEDSTER modifies the state of the sender before issuing the transaction to make sure that the presented state to the Blockchain is always up to date. Therefore, an attacker cannot roll back to the previous states and fool the Blockchain into a biased decision.

We design SPEEDSTER to be a fully decentralized state channel network, i.e., any node can directly transact with any other nodes without intermediaries. As a result, vulnerabilities introduced by HTLC-based multi-hopping and routing are avoided [53], [63], [83]. Confidentiality and authenticity of transactions are ensured.

SPEEDSTER adopts efficient symmetric-key operations to replace the costly public-key algorithms for transaction generation and verification. Experimental results show that SPEEDSTER achieves four orders of magnitude better performance than the Lightning Network, the most widely adopted layer-2 network in practicejinghui:, and $50\times$ faster than TeeChain with batching.

With certificate-based channels, SPEEDSTER naturally supports interaction among multiple parties. The state information can be freely exchanged across channels of the same account. Therefore, off-chain multi-party smart contracts can be efficiently executed in SPEEDSTER.

EVALUATION. To evaluate the performance of SPEEDSTER, we implement a prototype system that can run on AMD [4], Intel [70], and ARM [9] platforms. We migrate eEVM [26], a full version of Ethereum Virtual Machine (EVM) [36] into this project, and execute unmodified Ethereum smart contracts off-chain. We also develop a set of contracts to show the unique features and performance of SPEEDSTER. Through the experiments, we present the specification for running SPEEDSTER, the outstanding transaction throughput, and the capability of executing various kinds of smart contracts that traditional state channels cannot support. The experiments include:

- Transaction load test: to test the transaction throughput directly occurred 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 Finance (DIFI) 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 (in-parallel game participation) in SPEEDSTER channel;
- Monopoly contract: To test the multi-party state channel capability of SPEEDSTER, we load a Monopoly smart contract into the enclave and execute it by four players alternately.

We implement SPEEDSTER on prototypes with multiple TEE types on Intel, AMD, and ARM platforms. The evaluation on all the platforms shows SPEEDSTER performs fast and takes $0.02ms$, $0.14ms$, and $20.49ms$ to process a contract-free transaction on Intel, AMD, and ARM platforms, respectively.

In comparison with other channel projects, the prototype system shows about $10,000\times$ faster transaction speed than the Lighting network and two orders faster than public-key based channel projects that also use TEE to process transactions, with no off-chain fee cost. jinghui:Under the batching mode [58], the prototype achieves a peek throughput of 6.9 million tps while TeeChain is 0.15 million tps [58].

We also compare the main chain cost of SPEEDSTER with the projects of Lightning network (LN) [60], DMC [30], TeeChain [58], and SFMC [20] to demonstrate the cost efficiency of SPEEDSTER. Based on our evaluation results, creating/closing channels off the chain reduces 97% on-chain data in a similar scale of LN.

The source code of SPEEDSTER is publicly available at https://bit.ly/3a32ju7.

II. BACKGROUND

In this section, we provide the background of the technologies that we use in our system.

A. Blockchain and Smart Contract

Blockchain is a distributed ledger that leverages cryptography to maintain a transparent, immutable, and verifiable transaction record [21], [74]. In contrast to the permissioned Blockchain [7], [82], permissionless Blockchain [82] is publicly accessible but constrained by the inefficient consensus protocols, such as PoW [50], [74], on top of the asynchronous global network infrastructure, which causes a series of performance bottlenecks in practice. See [40], [79], [84], [93] 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

also verify the computation result through consensus protocols, thus creating a fair and trustless environment to foster a range of novel applications [68], [94]. A well-known example is the Ethereum smart contract [21], [22], which runs inside the EVM [36]. EVM is isolated from the network and the file system or any other io services. EVM exists in every full Ethereum node and is used to execute the Ethereum smart contract with user transactions as input.

B. Layer-2 Channel Technologies

Layer-2 technologies are proposed to respond to scalability concerns [88], short storage for historical transactions [89], etc., for the main chain.

Payment channel is the first attempt to use an off-chain infrastructure to process micropayments between two parties without constant main chain involvement. To create an off-chain channel, each party of the channel needs to send a transaction to the Blockchain to commit a certain deposit amount. Once committed, the deposit is locked in the main chain until another transaction is issued to close the channel. Transactions in a payment channel can be sent back and forth and only use the locked-in deposit in the channel. Thus, this approach can avoid expensive on-chain operations and enhance transactional privacy.

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 [53], [60], [71]. 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 enabling state recording and updating. Hence the state channel network supports off-chain smart contract execution [25], [32]–[35]. However, conducting stateful activities across multiple parties are still costly due to the complex trust management. Currently, multi-party state channel [25], [32] are realized through the virtual state channel techniques [33]–[35].

Regardless of the differences, the current layer-2 technologies all need the assistance of the main chain for channel creation, remove, or dispute resolution on the basis of each channel, which incurs a huge overhead [60]. Moreover, privacy and instability [83] concerns also arise and hamper the wide adoption of the technology.

C. Trusted Execution Environment

Trusted Execution Environment (TEE) provides a secure, isolated environment (or enclave) in the 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], [44], [70] and AMD Secure Encrypted Virtualization (SEV) [5], [52] 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 enclave; and the TCB of AMD SEV is the SEV-enabled virtual machine that is protected by an embedded 32-bit microcontroller (Arm Cortex-A5) [52]. Other prominent TEE examples include Arm TrustZone [9], MultiZone and KeyStone [56] on RISC-V [39], 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 platforms.

Remote attestation [76] is an important function of TEE, which is used to verify the authenticity of the processor before the enclave program execution. Specifically, to prevent an attacker from simulating an enclave environment, a TEE-enabled processor uses a hard-coded root key to cryptographically sign the measurement of the enclave environment, including all the initial states, code, and data. As such, everyone can publicly verify the authenticity of the established enclave environment with help from vendors.

III. Threat Model and Design Goals

In this section, we present the threat model and design goals of SPEEDSTER.

A. Threat Model

SPEEDSTER is a layer-2 channel system that supports efficient channel transactions and multi-party state channels. We assume that nodes in the system run in the TEE-enabled platform (e.g., Intel, ARM, or AMD)jinghui:~~, and TEE is secure~~and all party trust enclave after successful attestation. An adversary could compromise the operating system of a target node through exploiting zero-day software vulnerabilities, and the adversary may further control the whole system software stack in the node. jinghui:make it simple, using same assumption as other project, refer to other chapter

jinghui:we have the same assumption with previous works, refer to later section for detail

jinghui:Upon attacks against enclave, we assume that adversary may want to seal the private key in the enclave through side-channel attack and forge fake balance or state to defraud other honest nodes; Adversary may also want to start rollback attack to reset the node to a previous state to gain benefit; DoS attack against enclave is also a concern to the node where adversary may disconnect the enclave from the internet making the victim node unable to join the transaction.

Consistent with previous research [25], [32]–[35], we jinghui:~~also~~ assume an abstract Blockchain instantiation as a trusted party to provide desired ledger functions, such as transparency, immutable storage, and verifiable computations with smart contracts. SPEEDSTER also assumes that the Blockchain nodes are empowered with adequate computing resources and reasonable storage space. Therefore, SPEEDSTER only concentrates on the design of the off-chain channel system.

B. Design Goals

SPEEDSTER is proposed to allow the network participants to work jointly and efficiently to process off-chain transactions and smart contracts. In this subsection, we discuss the main design goals to be accomplished in this work.

1) Efficient Channel System: As discussed in Section II-B, the current layer-2 channel system is not efficient in terms of time and economic costs. Specifically, expensive interactions with the main chain need to be carried out for each channel, and extra fees and trust need to be given to intermediate nodes for transaction forwarding and state updating. The implication of such a design contradicts the promised efficiency for off-chain micropayment processing.

Accordingly, we attempt to devise a functionally efficient off-chain network that should be able to attain the objectives: Channel creation and close are independent of the Blockchain. Channels are directly established between parties to avoid additional routing fees. Channels are dispute free.

2) Fully Decentralized Channel Network: To transact with a node in the off-chain network, the sender must create a direct channel connection with or find a valid routing path to the receiver. However, expensive channel creation makes it impossible for a channel user to have direct channels with all other nodes in the system. Multi-hopping addresses the problem and raises privacy concerns about the emergent centralized payment hubs [33], [43], [78], which is at odds with the decentralized blockchain promise.

In this work, we aim to design a fully decentralized channel network, where a user can freely set up a direct channel with any other user in the system. Thus, there are no centrality concerns. Note that none of the existing channel projects can support such distributed channel topology and management [57], [58], [60], [71].

3) Efficient Multi-Party State Channel: State sharing among multiple parties is instrumental for many real-world applications, such as voting, auction, and games. However, the current off-chain state channels are only supporting efficient state exchange between two parties [31], [33]. The involvement of more channel participants depends on intermediaries for complicated setup and management [25], [32].

In contrast, SPEEDSTER will enable an efficient design for multi-party state channel. The state information of one SPEEDSTER node will not be confined in any particular channel. Rather, it can be freely shared with other parties of interest. We aim to streamline the architectural design for easy setup and use.

4) Other Goals: Aside from the above design goals, we also aim to design SPEEDSTER to be able to: (1) preserve the privacy of transactions to make sure that all transactions are securely protected and only the result of transaction execution is unveiled (see Section V-C for detailed security definition and analysis); (2) SPEEDSTER system design should be abstract and general enough to not rely on any specific platform.

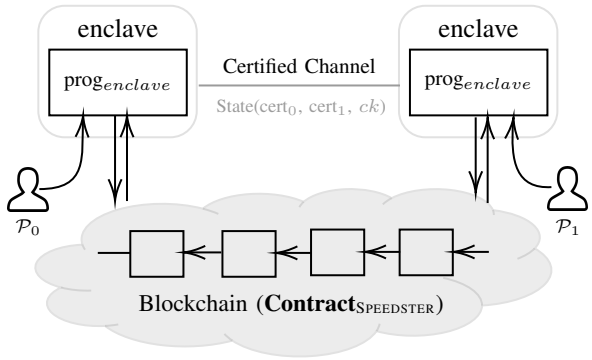

Figure 1. High-level architecture of SPEEDSTER. Channel is opened directly between enclaves of two users. Off-chain transaction is jinghui:~~relayed by mux and~~ processed in enclave. **Contract**$_{\text{SPEEDSTER}}$ is deployed on the Blockchain to record the state of nodes. The initial state of enclave is synchronized from the Blockchain.

IV. SPEEDSTER SYSTEM OVERVIEW

We first introduce the architecture of SPEEDSTER. Then we present the detailed design in this section.

A. System Architecture

SPEEDSTER contains two components: Enclave to execute the state channel core program $\text{prog}_{enclave}$ and Blockchain that runs the on-chain smart contract $\text{contract}_{\text{SPEEDSTER}}$. Figure 1 shows the high-level architecture of SPEEDSTER, in which two participants are connected with a *Certified Channel* (see Definition 1).

Prog$_{enclave}$. It runs in the enclave. The enclave is the root of off-chain trust that ensures the correct execution of the off-chain smart contract in the state channel. The program also handles the core functions related to the enclave account $\text{acc}_{enclave}$ and channel management.

Contract$_{\text{SPEEDSTER}}$. Blockchain is trusted to host and execute the smart contract $\text{contract}_{\text{SPEEDSTER}}$ to process transactions related to deposit and claim associated with $\text{acc}_{enclave}$. The claim transaction of SPEEDSTER contains multiple signatures and hence requires on-chain assistance of the smart contract.

B. Workflow

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

1) Node Initialization: In Step 1, when the $\text{prog}_{enclave}$ is loaded into the enclave for the first time, an account $\text{acc}_{enclave}$ is created in the enclave along with a pair of keys pk and sk. The enclave keeps sk securely, and pk can be used as the address by the node owner to deposit into the $\text{acc}_{enclave}$ on the Blockchain. After the Blockchain confirms the initial transaction, the user also loads this transaction into the enclave to initialize its state. The initial state of the enclave is defined as $\text{state}^0 := (\text{tx}_0, \text{amt}_0, \text{msg}_0)$, a tuple that contain the deposit transaction tx_0, the associated amount amt_0, and the additional

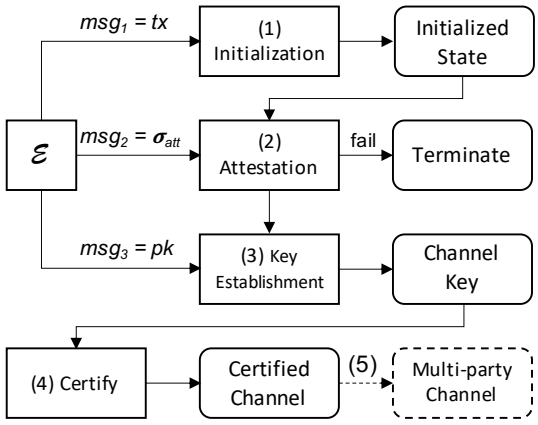

Figure 2. Workflow of node initialization and certified channel creation. $\mathcal{E}$ is the environment, including such as the Blockchain and the channel users, who passes input to SPEEDSTER node.

message msg_0. Further deposit is allowed to update the initial state of $\mathsf{acc}_{enclave}$.

2) Enclave State Attestation: Step 2 is enclave attestation that needs to be carried out to attest the enclave environment including the initial states of the parties with whom the node wants to establish channels. Note that in our design 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 Σ is a digital signature scheme and msk is the processor manufacturer-assigned secret key [76]. The initial state reflects the starting point of $\mathsf{acc}_{enclave}$, which should match the recorded state on the Blockchain. If a node passes the attestation, it means that the $\mathsf{acc}_{enclave}$ is set up with the correct on-chain deposit and should be trusted for the subsequent off-chain transactions.

3) Channel Key Establishment: Step 3 is activated once both ends of the channel have passed the remote attestation. It leverages a key agreement protocol that takes the public key as input to securely generate a shared channel-key [13], [17]. The messages transferred is signed with the respective private key sk of $\mathsf{acc}_{enclave}$.

4) Channel Certification: Step 4 assigns a new identifier $\mathsf{ccid} := \mathsf{H}(SORT(\{\mathsf{pk}_0, \mathsf{pk}_1\}))$ for the channel, where SORT can be any function used to make sure both ends of the channel agree on the same order of pk's, thus leading to the identical ccid. Next, one party creates a certificate $\mathsf{cert} := \Sigma.\mathsf{Sig}(\mathsf{sk}_i, \mathsf{pk}_{1-i})_{i\in\{0,1\}}$ for the other with the target pk. With a cert from the counterpart, the player can claim the fund received from the other party on the Blockchain. A *Certified Channel* has certs from both of its participants.

5) Multi-party State Channel Establishment: Step 5 will be optionally taken only if there is a need to establish a multi-party state channel. In this step, a group channel-key is generated for securely sharing the channel state among parties. This step cannot complete until after all the necessary two-party channels are founded. Note that the group key only works for the multi-party state channel function and can coexist with the keys for direct channels. See Section IV-C3 for details.

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

C. Key Functions

In this subsection, we elaborate on the key functions of SPEEDSTER: (1) certified channel, (2) fully decentralized channel network, and (3) efficient multi-party state channel.

1) Certified Channel: One of the issues that arise by introducing TEE into the Blockchain system is that existing Blockchain projects cannot verify the authenticity of TEE platforms. Though remote attestation provides authenticity and integrity, it is not straightforward for the current Blockchain implementation. SPEEDSTER proposes *Certified Channel*, as defined below.

Definition 1 (*Certified Channel*)**.** *A channel with both of its participants running on TEE-supported processors and having certificates from the other party in the channel is a Certified Channel.*

For example, in a *Certified Channel* between *Alice* and *Bob*, *Alice* has to verify the authenticity and integrity of both the hardware and software environment of *Bob* through enclave attestation and then sends a certificate to the channel before sending any transaction to *Bob*, vice versa.

With the *Certified Channel*, Blockchain is agnostic to the verification implementation for off-chain nodes by offloading the task to the enclaves of the nodes. As long as the node presents a valid certificate issued by the other channel counterpart, Blockchain will trust this enclave node and the associated transactions as authenticated. In this way, the balance security is guaranteed, i.e., no party can counterfeit transactions or claim more than he/she deserves because all off-chain transactions are believed to be securely processed by enclave functions.

Dispute-free Channels. The reasons for the dispute in prior state channel design are two folds: (1) allowing users to commit old channel state to the Blockchain; (2) the unresponsive counterpart. By realizing *Certified Channel*, SPEEDSTER can avoid expensive on-chain dispute resolution and create a dispute-free channel network. jinghui: ~~This is because the enclave can only manage the channel state, and it will be updated before sending any transactions. So each channel will always keep the lasted channel state. Once a "claim" transaction is issued, the channel will be frozen to have the state in the channel match that in the transaction.~~To achieve this, enclave updates the state before sending any transaction and locks the channel if a "claim" transaction is issued.

2) Fully Decentralized Channel Network: We define Fully Decentralized Channel Network (FDCN) as a network where all nodes can establish direct channels economically, and therefore it is not necessary to rely on an intermediary for routing. In such a network, the privacy and security risks associated with the central routing nodes can be mitigated. jinghui: ~~Further, only a minimum fee will be charged for off-chain payments. The stability of the layer-2 network will~~

~~be significantly improved, which is also aligned with the decentralized design principle of Blockchain.~~FDCN significantly improves the stability of the layer-2 network and also aligns with the decentralized design principle of Blockchain.

In reality, current state channels cannot support this idea as a substantial amount of collaterals have to be locked in the main chain jinghui:~~to open multiple channels~~and it is costly to open channels. SPEEDSTER addresses this issue by adopting an account-based channel creation jinghui:~~. In other words, ~~, where the same on-chain deposit can be reused to open multiple channels. As a result, a SPEEDSTER node may create as many channels as it prefers with any other nodes in the network economically.

Definition 2 (Fully Decentralized Channel Network)**.** *A payment/state channel network where nodes can establish direct channel connections with other nodes efficiently off the chain and process channel transactions without relying on intermediaries is a Fully Decentralized Channel Network.*

3) Efficient Multi-Party State Channel: As discussed in Section III-B3, designing a multi-party state channel is a fundamental challenge and necessary for many off-chain smart contract use cases, such as a multi-party game. Next, we detail our design.

Multi-party channel key establishment. Before establishing the group channel-key, gk, we assume that all channel members have passed the attestation and state verification such that all participants are trustworthy in the channel key exchange process. We also assume that a peer-to-peer channel key has already been set up between each pair of members beforehand.

With n known participants of a channel to be created, we first generate the channel id ccid by hashing the sorted public keys of all participants as follows $\mathsf{ccid} := \mathsf{H}(SORT(\{\mathsf{pk}_i\}^{i\in[N]}))$. Consequently, all participanting nodes get the same ccid. Next, each node i sends a locally-generated random number r_i to all other channel members through the peer-to-peer channels. After collecting all the random numbers from other members, every node generates a shared multi-party channel key locally by applying another hash function, $\mathsf{gk} := \mathsf{H}(SORT(\{r_i\}^{i\in[N]}))$.

In this way, the key is bind with ccid, and only transactions with a tag of the matched ccid can use the key for encryption and decryption. Using this group key, a transaction in a multi-party channel only needs to be encrypted once and then broadcast to all the members.

Coordinated transaction execution. To avoid ambiguity of the transaction execution in a multi-party smart contract scenario, transactions from different parties need to be ordered before processed. In a distributed network, a trusted time source is hard to get for coordination. To address this issue, we let each party i takes turn to send its transactions by the order derived from $\mathrm{SORT}(\{\mathsf{pk}_i\}^{i\in[N]})$. Specifically, all the nodes are mute after the channel key is created except for the one with the highest order by the SORT function. Moving forward, all other nodes need to wait for their turn to issue transactions. Figure 3 shows an example of the communication order among three-channel members. Moreover, other nodes would negotiate to skip it for this turn if a node does not respond on its turn. We discuss more multi-party state synchronization in detail in the Appendix D.

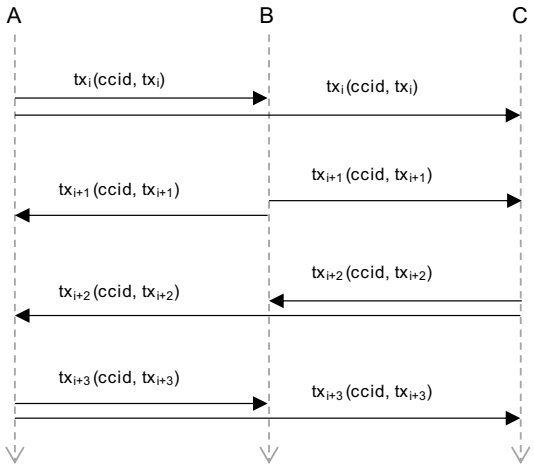

Figure 3. An example of ordered communication among A, B and C, assuming $\mathrm{SORT}(\mathsf{pk}_A)>\mathrm{SORT}(\mathsf{pk}_B)>\mathrm{SORT}(\mathsf{pk}_C)$.

Note that there will be no dispute in the created multi-party state channel. This is inherited from the underlying *Certified Channel*s that can effectively eliminate the disputes related to channel states.

V. PROTOCOL AND SECURITY ANALYSIS

In this section, we formally introduce the SPEEDSTER protocol $\Pi_{\text{SPEEDSTER}}$ and analyze its security.

A. Attested Execution

$\mathcal{G}_{att}$ 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 [76]. 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 mpk by default to ensure the authenticity, whereas *Certified Channel* leverages authenticated encryption to ensure the authentication and integrity instead of signature. Therefore, we add a switch to "resume" command to turn off the signature, when the "switch" is set, execution output through "resume" is signed under the mpk, otherwise, "resume" does not sign it (see Figure 8 in Appendix for detail).

B. SPEEDSTER Protocol $\Pi_{\text{SPEEDSTER}}$

We formally present $\Pi_{\text{SPEEDSTER}}$ into three parts: the program $\mathsf{prog}_{enclave}$ that is loaded into the enclave in Figure 9 in Appendix, and the smart contract $\mathsf{contract}_{\text{SPEEDSTER}}$ running on the Blockchain in Figure 7 in Appendix. In the protocol, we let $\mathcal{P}$ denote the user and tx represents the on-chain transaction. To

execute the off-chain smart contract in $\mathsf{prog}_{enclave}$, we define a function $\mathsf{Contract}_{\mathsf{cid}}(\cdot)$ which is parameterized with a smart contract id cid. $\mathsf{Contract}_{\mathsf{cid}}(\cdot)$ consumes the channel state and the node balance to ensure the node balance consistency across channels. $\mathsf{Contract}_{\mathsf{cid}}(\cdot)$ generates output outp based on the input and updates the channel state state'.

1) Node Initialization.: For the first time to boot up a SPEEDSTER node, user sends the "install" command to the enclave to load $\mathsf{prog}_{enclave}$. Then, user calls function *(1)* of $\mathsf{prog}_{enclave}$ by sending message ("init"), and an enclave account $\mathsf{acc}_{enclave}$ (sk, pk) is created by the digital signature scheme Σ. For attestation purpose, a measurement σ_{att} of the enclave is generated with the $\mathsf{prog}_{enclave}$, the public key pk of $\mathsf{acc}_{enclave}$, the currently available node balance bal, and the node initial state state^0.

2) Deposit.: Depositing into the $\mathsf{acc}_{enclave}$ issues the on-chain state of the $\mathsf{acc}_{enclave}$, we use pk to represent the address for clarity. For generality, tx is committed to the smart contract $\mathsf{contract}_{\text{SPEEDSTER}}$. The "deposit" phase works as follows. First a user commits tx to the on-chain smart contract $\mathsf{contract}_{\text{SPEEDSTER}}$. Next, user calls the the function *(2)* of $\mathsf{prog}_{enclave}$ by sending message "deposit" and pass tx as a parameter to $\mathsf{prog}_{enclave}$. Finally, $\mathsf{prog}_{enclave}$ verifies the signature of tx, and updates the local initial state state^0.

3) Certified Channel.: Each certified channel in $\Pi_{\text{SPEEDSTER}}$ is associated with an ID $\mathsf{ccid} := \mathsf{H}(SORT\{\mathsf{pk}_{\mathcal{R}}, \mathsf{pk}\})$. See function *(3)* in Figure 9 in Appendix for details. A shared channel key ck is produced in this step. The certificate cert of the channel is created using public keys of both parties. To prevent rollback attacks on σ_{att}, $\mathsf{prog}_{enclave}$ generates a new signature σ_{att} by signing the tuple (state^0, bal, $\{\mathsf{pk}_i\}^{i\in\{0,1\}}$, $\mathsf{prog}_{enclave}$) for each channel after function *(3)* returns. state^0 is the initial state of $\mathsf{acc}_{enclave}$, and bal is the currently available balance of $\mathsf{acc}_{enclave}$. The tuple is signed under the manufacture secret key msk to reflect the root trust of the hardware. The cert is verified in function *(5)*.

4) Multi-Party State Channel.: A multi-party state channel is built over the existing certified channels but does not block the function of the existing certified channel. To create a multi-party state channel, user calls the function *(4)* of $\mathsf{prog}_{enclave}$ by sending "openMulti" message and passes the set of ccid that connects current user with other participants. We abstract out the process of multi-party shared key generation for the same reason mentioned above.

5) Transaction.: To send a channel transaction, user calls function *(6)* of $\mathsf{prog}_{enclave}$ by sending "send" command to $\mathsf{prog}_{enclave}$ through $\mathcal{G}_{att}$.resume(·) and pass the target ccid along with the necessary parameters in inp. Then, $\mathsf{prog}_{enclave}$ executes inp over the associated contract by calling $\mathsf{Contract}_{\mathsf{cid}}(\cdot)$ and update the channel state. 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 channel transaction is encrypted with the channel key ck through authenticated encryption scheme $\mathcal{AE}$.

6) Claim.: To claim the credits that $\mathcal{P}$ receives from channel transactions, user could issue a "claim" call to $\mathsf{prog}_{enclave}$. To answer "claim" call, $\mathsf{prog}_{enclave}$ freezes all local channels, and extracts the cert that $\mathcal{P}$ receives in each channel from counterparts. The certs and the local node state state construct the claim transaction tx. Then, $\mathsf{prog}_{enclave}$ signs tx with the private key sk of $\mathsf{acc}_{enclave}$ and returns the signed transaction to the user. Then, user relays tx to the $\mathsf{contract}_{\text{SPEEDSTER}}$. Finally, $\mathsf{contract}_{\text{SPEEDSTER}}$ would execute the claim transaction on the Blockchain following given smart contract logic.

C. Security and Privacy of $\Pi_{\text{SPEEDSTER}}$

1) Security.: We formalize an Universal Composability (UC) [10], [23], [55], [58] ideal functionality $\mathcal{F}_{\text{SPEEDSTER}}$of SPEEDSTER in Figure 6 of Appendix A. $\Pi_{\text{SPEEDSTER}}$ is modeled to UC-realize $\mathcal{F}_{\text{SPEEDSTER}}$. The security of $\Pi_{\text{SPEEDSTER}}$ is given 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. jinghui:~~See Appendix A~~Due to the page limitation, we leave our full version protocol and proof in link:xxx sun: Please refer to the full version of this paper []

□

2) Privacy.: Theorem 1 also implies the stronger privacy protection compared to conventional payment/state channel networks in that: a. All channels in SPEEDSTER are created directly between participants. No intermediate node is required to relay transactions, thus alleviating the privacy concerns introduced by HTLC [33], [42], [43], [62]; b. Except for the channel participants, transaction confidentiality is ensured by SPEEDSTER.

3) defend against TEE Attacks: jinghui:hardware, software jinghui:worst case, realiblity after break down jinghui:what we did to prevent attack Though the hardware-assisted TEE serves as a promising way to replace the complex software-based cryptographic operations, recent research shows that TEE implementations on certain platforms are vulnerable to the side-channel attacks [41], [73], [80], [85], [87], roll-back attacks [18], [28], [64] and incorrect implementation and configurations [11], [19], [48], [75].

SPEEDSTER does not assume dependency on any particular platform and supports on AMD, Intel, and ARM platforms; instead, it adopts TEE as an abstraction. Additionally, researchers are working diligently to address these vulnerabilities from both software and hardware levels. For example, AMD provides SEV-SE [4] and SEV-SNP [81] to protect against certain speculative side-channel attacks and TCB roll-back attacks. SGX provides hardware monotonic counters [47] to address the rollback attack, and Intel keeps updating its microcode [45] to mitigate various side-channel attacks. Apart from the hardware level updating and patching, proper implementation of the system can also help mitigate side-channel

vulnerabilities [46]. Thus, we argue that those vulnerabilities should not stop us from researching and developing systems upon a secure TEE.

DoS Attacks. SPEEDSTER can create and close channels off the main chain. The channel state is stored in and updated only by TEE. We note that an adversary may launch a denial-of-service (DoS) attack against the node by cutting off the Internet connection of the victim, or abruptly shut down the OS, to force stop the enclave functions. These DoS attacks against the enclave remain as orthogonal issues. Additionally, for the node failure issue, e.g., caused by DoS attack and physical damage, SPEEDSTER can use a similar technique employed by Teechain [58] and Ekiden [24] to address the concern.

VI. IMPLEMENTATION OF SPEEDSTER

In this section, we overview the implementation of SPEEDSTER on various platforms, i.e. Intel, AMD, and ARM. We build the virtual machine (VM) on top of the open-source eEVM [26]. Thus SPEEDSTER runs Ethereum smart contracts off-the-shelf. The cryptographic library that we use in $\mathsf{prog}_{enclave}$ is `mbedTLS` [66], an open-source SSL library ported to TEE [27], [91]. In particular, we adopt 1) `SHA256` to generate the secret seed in enclave to create $\mathsf{acc}_{enclave}$ and to get the hash value of a claim transaction, 2) `AES-GCM` [69] to authentically encrypt transactions in the state channel, and 3) `ECDSA` [51] to sign the cert and the claim transaction. We also customized the OpenEnclave [27] to compile the prototype for Intel and ARM platforms. For AMD SEV, we use VMs as the enclaves to run $\mathsf{prog}_{enclave}$. The host can communicate with the enclave via the socket. We test the performance of the following functions to highlight the advantages of SPEEDSTER.

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

Instant State Sharing: We implement instant state sharing function in C++ to allow users to create direct channels among each other off-chain and remove costly signature operations for transactions. As a result, SPEEDSTER significantly reduces the communication overhead and enables instant information exchange, such as high-quality video/audio sharing while preserving privacy, which is difficult to realize with prior efforts [].

Faster Fund Exchange: We implement a `ERC20` contract [86] with 50 LOC in Solidity to demonstrate the improved performance of SPEEDSTER in executing off-chain smart contracts. This is thanks to the elimination of asymmetric signature operations for off-chain transactions in SPEEDSTER.

*Sequential Contract Execution:*jinghui:~~To compare with conventional state channels [] and highlight the performance gain~~ To highlight the performance of SPEEDSTER executing sequential transaction contract, we implement the popular two-party Gomoku chess smart contract with 132 LOC in Solidity. With this turn-based application, we also show that SPEEDSTER can resolve the dispute off the main chain. Players cannot reuse the locked fund in the beginning until the game ends. Therefore, a cheating player jinghui:~~will gain~~gains no benefit but playing the game honestly.

Parallel Contract Execution: Applications that need users to act simultaneously, such as Rock-Paper-Scissors (RPS), are not easy to achieve in conventional sequentially structured state channels. SPEEDSTER supports applications running in parallel, faithfully manages the states, and only reveals the final results to players. We implement a typical two-party RPS game with 64 LOC in Solidity.

Multi-party Applications: To test the ability of multi-party off-chain smart contract execution, we implement Monopoly game smart contact with 231 LOC in solidity. In Monopoly, players roll two six-sided dices to decide how many steps to move forward in turn and transact with other players.

VII. EVALUATION

We first elaborate on the configurations of the used platforms and then present the evaluation results of SPEEDSTER.

SGX platform: We test SPEEDSTER with a quad-core 3.6 GHz Intel(R) E3-1275 v5 CPU [49] 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 node [59] as the baseline for comparison on another physical machine with the same configurations.

SEV platform: We evaluate SPEEDSTER on a SEV platform with 64 GB DRAM and a SEV-enabled AMD Epyc 7452 CPU [3], 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 [4]. The QEMU emulator version 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.

A. Code Size

The eEVM contains $3.2k$ LOC in C++ and $22.1k$ LOC coming from its reliance. We added extra 650 LOC to eEVM to port it into SPEEDSTER.

SPEEDSTER is evaluated on Intel, AMD, and ARM platforms with around $38.5k$ LOC in total, as shown in Table I. Specifically, $25.3k$ LOC comes from the contract virtual machine eEVM [26] which is shared with all cases. $\mathsf{prog}_{enclave}$ has $3.1k$ LOC in C++ for SGX/TrustZone and $3.7k$ LOC for AMD SEV. The $\mathsf{contract}_{\text{SPEEDSTER}}$ deployed on the Blockchain is implemented with 109 LOC in Solidity.

Table I
LINE OF CODE IN SPEEDSTER.

	Component	Code	LOC	Total(#)
Shared	eEVM [26]	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	

B. Time Cost for Transaction Authentication

In the SPEEDSTER prototype, we use AES-GCM to replace ECDSA that is adopted in previous channel projects for transaction authentication. By trusting the secure enclave, SPEEDSTER uses efficient symmetric operations to realize both confidentiality and authenticity of transactions at the same time. Figure 4 shows the comparison of the performance between ECDSA and AES-GCM when processing data of the size 128 bytes, 256 bytes, and 1024 bytes, 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.

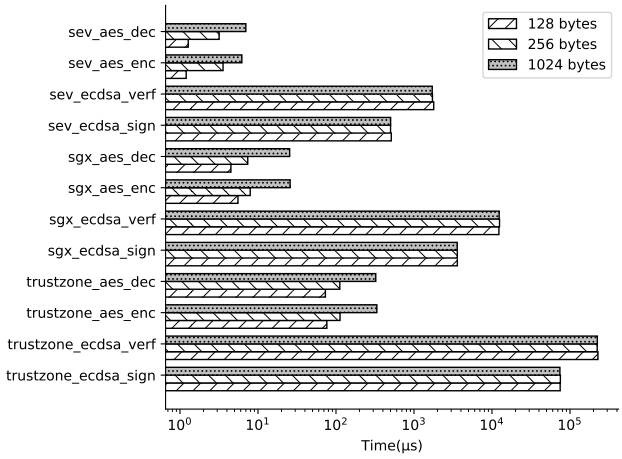

Figure 4. Performance comparison between ECDSA and AES-GCM enabled transaction security on SGX, SEV, and TrustZone platforms.

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. Besides, 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 [37]. Therefore, using symmetric-key operations will significantly boost the transaction-related performance.

C. Transaction Performance

We evaluate SPEEDSTER on the time cost for transactions in a direct channel under the test cases in Section VI on Intel, AMD, and ARM platforms. In this experiment, we use the popular layer-2 network, Lightning network, as the baseline. jinghui:~~In LN, an end-to-end transaction needs to wait for two round-trips to be confirmed, one to discover the routing path and the other to process the transaction. SPEEDSTER can set up a direct channel between two nodes. So there will be no routing cost, and we~~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.jinghui:Additionally, we also test the batching transaction performance as a comparison with TeeChain [58].

Table II
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 result is an average of 10,000 trials and is shown in Table II. ERC20, Gomoku, and rock-paper-scissor (RPC) are the smart contracts that we implemented.

1) Evaluation on SGX: The 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. When calling smart contracts, it takes $0.1930ms - 0.4500ms$ to process a contract-calling transaction.

2) Evaluation on AMD: As there is no available AMD cloud virtual machine that supports SEV, we only evaluate SPEEDSTER on the AMD platform by running the $\text{prog}_{enclave}$ in two Ubuntu guest virtual machines. To protect the code and data of $\text{prog}_{enclave}$ that runs in the enclave, we only allow users to access $\text{prog}_{enclave}$ through mux by calling the related interface. 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 II, RPC ($0.1365ms$) and ERC20 ($0.1382ms$) are faster than Gomoku ($0.6667ms$) due to their simple logic and fewer steps to take.

3) Evaluation on ARM: As the evaluation of ARM TrustZone runs upon the QEMU emulator, the performance of ARM is the worst. Nevertheless, the evaluation result in Table II shows that $\text{prog}_{enclave}$ takes $20.496ms$ to run direct transactions, which is around $9\times$ better than LN. For smart contract execution, it takes $30 - 90ms$ to process contract transactions.

Table III
CHANNEL PERFORMANCE.

	Throughput (tps)			Latency (ms)	
	LN(lnd)	Speedster	Change (×)	LN(lnd)	Speedster
Payment	14	72143	5153×	548.183	80.483
ERC20	N/A	30920	N/A	N/A	82.490
RPC	N/A	53355	N/A	N/A	80.743
Gomoku	N/A	2549	N/A	N/A	82.866

4) 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 GiB memory) virtual machines, which are backed by the 3.7GHz Intel XEON E-2288G processor, one at the East US, and another 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 the machine as a baseline to highlight the significant performance improvement of SPEEDSTER. Table III shows the evaluation result. The throughput of LN is $14tps$ while SPEEDSTER achieves $72143tps$ on payment operation, $5000\times$ more efficient than LN. Specifically, the latency to execute a SPEEDSTER transaction is around $80ms$, close to the RTT time between testing hosts, while the latency to run an LN payment transaction is around $500ms$.jinghui:To test the batching performance of SPEEDSTER, we batch the transaction for $100ms$ [58] and get a peek throughput of 6.9 million tps while TeeChain is 0.15 million tps on the paper. Noting here that though TeeChain is open source, we could not find the part related to the batching and no more information was given even after we contacted the TeeChain author.

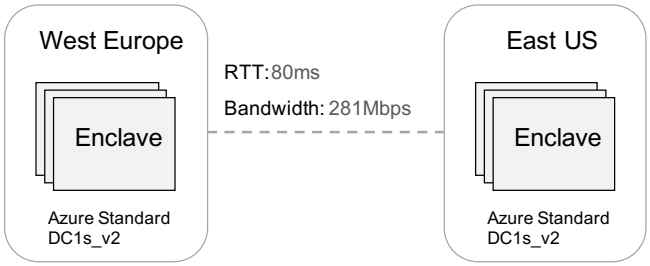

Figure 5. Network setup for the evaluation.

jinghui:~~Batching is a test feature of the Lightning network to enable aggregated transactions processing to boost the network throughput. TeeChain [58] uses batching to increase its transaction speed. Batching cannot reduce the execution time for a single channel transaction. SPEEDSTER can also use this technique to improve its performance. However, the implementation is platform-dependent. Lightning has not published the specification. We are not able to access the source code of TeeChain after contacting the authors. On the other hand, the evaluated smart contract types do not support trivial transaction batching. Therefore, we assess the system performance without considering this technique.~~

D. Channel System Comparison

SPEEDSTER supports efficient multi-party state channel creation and close. To highlight the advantages of SPEEDSTER, we compare SPEEDSTER with other major existing channel projects. Table IV shows the difference in terms of the following features: Direct off-chain channel open/close, dynamic deposit (dynamically adjusting fund in an existing channel on demand [58]), symmetric-key operations for transactions, off-chain smart contract execution, full decentralization, multi-party state channel, dispute-free, and duplex channel.

TeeChain [58] also adopts TEE for the off-chain channel network. However, TeeChain relies on the Blockchain to deposit in advance, which will be used for channel creation. The fund of each channel is adjustable on demand. However, a TeeChain node is not likely to open many channels simultaneously since it requires much deposit and may lock in a large amount of fund in channels. jinghui:As transaction in TeeChain bounds to the deposit from the Blockchain, it needs to resolve dispute online, while transaction in SPEEDSTER is related to the cert in the off-chain channel, as long as a party has a valid cert, it can claim fund online, and the transaction procedure of SPEEDSTER ensures the balance security.

In Perun [33], 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 determine the available capacity in those sub-channels. As shown in Table IV SPEEDSTER is the only off-chain state channel project that accomplishes all the listed functions.

E. Main Chain Cost

Similar to previous work [20], [58], we evaluate the main chain cost: (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 [20]).

We select a set of representative channel projects to evaluate and be compared with SPEEDSTER. In particular, we choose LN [60] (the most popular payment channel system in reality), DMC [30] (a duplex payment channel), TeeChain [58] (a TEE-based channel project), and SFMC [20] (it also supports off-chain channel open/close). The comparison is carried out by analyzing each project under bilateral termination [58], i.e., each channel could be closed without any dispute. The result is shown in Table V. 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 for close with the latest channel state and signatures from both sides to the Blockchain.

In TeeChain [58], deposits are handled by a group of committee nodes and dynamically associated 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

Table IV
FEATURE COMPARISON WITH OTHER CHANNEL PROJECTS

Features	Channel Projects							
	LN [60]	TeeChan [57]	TeeChain [58]	DMC [30]	SFMC [20]	Perun [33]	Celer [31]	Speedster
Direct Off-chain Channel Open	✗	✓	✓	✗	✓	✓	✗	✓
Direct Off-chain Channel Close	✗	✗	✓	✗	✓	✓	✗	✓
Dynamic Deposit	✗	✗	✓	✗	✓	✗	✗	✓
Off-Chain Contract Execution	✗	✗	✗	✗	✗	✗	✓	✓
Fully Distributed	✗	✗	✗	✗	✗	✗	✗	✓
Multi-Party State Channel	✗	✗	✗	✗	✓	✗	✗	✓
Dispute-Free	✗	✗	✗	✗	✗	✗	✗	✓
Duplex Channel	✗	✗	✓	✓	✓	✗	✗	✓

Table V
NUMBER OF ON-CHAIN TRANSACTIONS AND BLOCKCHAIN COSTS (BC) FOR EACH CHANNEL.

Payment Channel	Setup		Open Channel		Close Channel		Claim		Total	
	No.tx	BC	No.tx	BC	No.tx	BC	No.tx	BC	No.tx	BC
LN [59]	2	2	1	2	1	2	N/A	N/A	4	6
DMC [30]	N/A	N/A	1	2	1	2	N/A	N/A	2	4
TeeChain [58]	1	1+p/2	off chain	off chain	off chain	off chain	1	1+p/2+m	2	2+p+m
SFMC [20]	1/c	p/c	off chain	off chain	off chain	off chain	1/c	p/c	2/c	2p/c
Speedster	1/c	1/c	off chain	off chain	off chain	off chain	1/c	1/c	2/c	2/c

the committee. Since TeeChain can also close the channel off the chain, there is no associated cost for that. All committee members in TeeChain use the same m-out-of-p multi-signature for each "deposit" transaction, so the BC is $1 + p/2 + m$.

In contrast, the deposit to the account of a SPEEDSTER node can be freely allocated to different channels created by the same node. Therefore, we only need 1 "deposit" transaction to initialize the account and create c channels. There is no cost for channel opening and closing as SPEEDSTER can do it completely offline. To claim the remaining fund from active channels, one on-chain transaction needs to be sent. Assuming that each deposit and claim transaction is shared by c channels on average, SPEEDSTER requires $2/c$ on-chain transactions with a BC of $2/c$ for each channel on average.

In general, 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 and 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 $c \geq 11$ [14]. Compared to TeeChain, SPEEDSTER reduces over 84% BC when $c \geq 2$.

VIII. DISCUSSION

In this section, we discuss other aspects of SPEEDSTER, i.e., partial claim from $\mathsf{acc}_{enclave}$, unavailability of TEE, and cross-chain state channel.

Partial Claim. In SPEEDSTER, one "claim" transaction can be used to freeze all the channels that belong to the same user and to withdraw all the related fund from these channels. We can also extend the current function to support the partial claim. In other words, the user can decide how much funds can be claimed from the total in a channel. As a result, the partially-claimed channel can still operate as normal.

Cross-chain State Channel. Running inside the enclave, $\mathsf{prog}_{enclave}$ does not rely on any particular form of Blockchain to operate. Therefore, it is possible to generate multiple enclave accounts and deposit from multiple Blockchain projects to these accounts. Ideally, this would allow SPEEDSTER to create a cross-chain state channel that could significantly reduce related costs. We will leave this as our future work.

IX. RELATED WORK

There is extensive research on constructing efficient layer-2 channel networks. MicroCash [2] that supports concurrent micropayments. Perun [33] created a virtual payment channel over two or more existing ledger channels without the involvement of intermediate nodes. Based on Perun, Dziembowski *et al.* proposed the first multi-party state channel [32] that operates recursively among participants. Sprites [71] was built on LN and reduced the latency of LN in multi-hop transactions. Pisa [67] enabled the party to delegate itself to a third party in case it goes off-line. REVIVE [53] rebalances the fund in channels to increase the scalability of the payment channel network.

Table VI
BLOCKCHAIN RELATED PROJECTS THAT USE TEE

	Projects	Description
Blockchain	BITE [65]	Bitcoin Lightweight Client
	FastKitten [29]	Smart Contracts on Bitcoin
	Tesseract [12]	Real-Time Cryptocurrency Exchange
	Town Crier [92]	An authenticated Data Feed
	Ekiden [24]	Smart Contract Execution Platform
Layer-2 Channel	TeeChan [57]	Secure Payment Network
	TeeChain [58]	Secure Payment Network

As listed in Table VI, there are many TEE-enabled projects on Blockchain and state channels. Town Crier [92] used SGX to implement authenticated data feed for smart contracts. Ekiden [24] and FastKitten [29] proposed Blockchain projects that can preserve the confidentiality of the smart contract. In Tesseract [12], credits can be exchanged across multiple chains. TeeChan [57] was built on top of LN and created new channels instantly off-the-chain but it still required synchronization with Blockchain and could only create one channel with one deposit. Based on TeeChan, TeeChain [58] was proposed to set up a committee for each TEE node and dynamically allocate deposit to a channel.

X. CONCLUSION

SPEEDSTER is the first account-based off-chain channel system, where 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. The practicality of SPEEDSTER is validated on different TEE platforms. The experimental results show much-improved performance compared to LN and other layer-2 channel networks.

REFERENCES

[1] I. Allison, "Ethereum's vitalik buterin explains how state channels address privacy and scalability.(july 2016)," 2016.

[2] G. Almashaqbeh, A. Bishop, and J. Cappos, "Microcash: Practical concurrent processing of micropayments," *arXiv preprint arXiv:1911.08520*, 2019.

[3] AMD, "Amd epyc™ 7452," https://www.amd.com/en/products/cpu/amd-epyc-7452, accessed: 2020-04-27.

[4] ——, "AMD ESE/AMD SEV," https://github.com/AMDESE/AMDSEV, accessed: 2020-04-27.

[5] ——, "Secure encrypted virtualization (sev)," https://developer.amd.com/sev/.

[6] I. Anati, S. Gueron, S. Johnson, and V. Scarlata, "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, 2013.

[7] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. De Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich *et al.*, "Hyperledger fabric: a distributed operating system for permissioned blockchains," in *Proceedings of the Thirteenth EuroSys Conference*, 2018, pp. 1–15.

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

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

[10] C. Badertscher, U. Maurer, D. Tschudi, and V. Zikas, "Bitcoin as a transaction ledger: A composable treatment," in *Annual International Cryptology Conference*. Springer, 2017, pp. 324–356.

[11] G. Beniamini, "Trust issues: Exploiting trustzone tees," *Google Project Zero Blog*, 2017.

[12] I. Bentov, Y. Ji, F. Zhang, L. Breidenbach, P. Daian, and A. Juels, "Tesseract: Real-time cryptocurrency exchange using trusted hardware," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2019, pp. 1521–1538.

[13] D. J. Bernstein, "Curve25519: new diffie-hellman speed records," in *International Workshop on Public Key Cryptography*. Springer, 2006, pp. 207–228.

[14] bitcoinvisuals.com, "Average channels per node," https://bitcoinvisuals.com/lightning, 2020.

[15] blockchain.com, "Average block size," https://www.blockchain.com/charts/avg-block-size, 2019.

[16] ——, "Bitcoin transaction rate," https://www.blockchain.com/en/charts/transactions-per-second?timespan=all, 2019.

[17] D. Boneh and M. Zhandry, "Multiparty key exchange, efficient traitor tracing, and more from indistinguishability obfuscation," *Algorithmica*, vol. 79, no. 4, pp. 1233–1285, 2017.

[18] M. Brandenburger, C. Cachin, M. Lorenz, and R. Kapitza, "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, 2017, pp. 157–168.

[19] R. Buhren, C. Werling, and J.-P. Seifert, "Insecure until proven updated: Analyzing amd sev's remote attestation," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 1087–1099.

[20] C. Burchert, C. Decker, and R. Wattenhofer, "Scalable funding of bitcoin micropayment channel networks," *Royal Society open science*, vol. 5, no. 8, p. 180089, 2018.

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

[22] ——, "A next-generation smart contract and decentralized application platform," *white paper*, 2014.

[23] R. Canetti, "Universally composable security: A new paradigm for cryptographic protocols," in *Proceedings 42nd IEEE Symposium on Foundations of Computer Science*. IEEE, 2001, pp. 136–145.

[24] R. Cheng, F. Zhang, J. Kos, W. He, N. Hynes, N. Johnson, A. Juels, A. Miller, and D. Song, "Ekiden: A platform for confidentiality-preserving, trustworthy, and performant smart contracts," in *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2019, pp. 185–200.

[25] T. Close, "Nitro protocol." *IACR Cryptology ePrint Archive*, vol. 2019, p. 219, 2019.

[26] M. Corporation, "Evm," https://github.com/microsoft/eEVM, 2019.

[27] ——, "openenclave," https://github.com/microsoft/openenclave, 2019.

[28] V. Costan and S. Devadas, "Intel sgx explained." *IACR Cryptology ePrint Archive*, vol. 2016, no. 086, pp. 1–118, 2016.

[29] P. Das, L. Eckey, T. Frassetto, D. Gens, K. Hostáková, P. Jauernig, S. Faust, and A.-R. Sadeghi, "Fastkitten: practical smart contracts on bitcoin," in *28th USENIX Security Symposium (USENIX Security 19)*, 2019, pp. 801–818.

[30] C. Decker and R. Wattenhofer, "A fast and scalable payment network with bitcoin duplex micropayment channels," in *Symposium on Self-Stabilizing Systems*. Springer, 2015, pp. 3–18.

[31] M. Dong, Q. Liang, X. Li, and J. Liu, "Celer network: Bring internet scale to every blockchain," *arXiv preprint arXiv:1810.00037*, 2018.

[32] S. Dziembowski, L. Eckey, S. Faust, J. Hesse, and K. Hostáková, "Multi-party virtual state channels," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2019, pp. 625–656.

[33] S. Dziembowski, L. Eckey, S. Faust, and D. Malinowski, "Perun: Virtual payment hubs over cryptocurrencies," in *2019 IEEE Symposium on Security and Privacy (SP)*, 2019, pp. 327–344.

[34] S. Dziembowski, S. Faust, and K. Hostakova, "Foundations of state channel networks." *IACR Cryptology ePrint Archive*, vol. 2018, p. 320, 2018.

[35] S. Dziembowski, S. Faust, and K. Hostáková, "General state channel networks," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2018, pp. 949–966.

[36] ethereum, "Ethereum virtual machine (evm) awesome list," https://github.com/ethereum/wiki/wiki/Ethereum-Virtual-Machine-(EVM)-Awesome-List, 2020-05-02.

[37] etherscan.io, "Ethereum blockchain size," https://etherscan.io/chartsync/chaindefault, 2020.

[38] W. Foxley, "As bitcoin cash hard forks, unknown mining pool continues old chain," *https://shorturl.at/svATX*, 2019.

[39] C. Garlati, "Multi zone trusted execution environment free and open api," in *RISC-V Workshop*, 2019.

[40] A. Gervais, G. O. Karame, K. Wüst, V. Glykantzis, H. Ritzdorf, and S. Capkun, "On the security and performance of proof of work blockchains," in *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, 2016, pp. 3–16.

[41] J. Götzfried, M. Eckert, S. Schinzel, and T. Müller, "Cache attacks on intel sgx," in *Proceedings of the 10th European Workshop on Systems Security*. ACM, 2017, p. 2.

[42] M. Green and I. Miers, "Bolt: Anonymous payment channels for decentralized currencies," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2017, pp. 473–489.

[43] J. Herrera-Joancomarti, G. Navarro-Arribas, A. R. Pedrosa, P.-S. Cristina, and J. Garcia-Alfaro, "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 ..., 2019.

[44] M. Hoekstra, R. Lal, P. Pappachan, V. Phegade, and J. Del Cuvillo, "Using innovative instructions to create trustworthy software solutions." in *HASP@ ISCA*, 2013, p. 11.

[45] Intel, "The latest security information on intel® products." https://www.intel.com/content/www/us/en/security-center/advisory/intel-sa-00329.html, accessed: 2020-08-18.

[46] ——, "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.

[47] ——, "Intel® dynamic application loader (intel® dal) developer guide," https://software.intel.com/en-us/dal-developer-guide-features-monotonic-counters, 2018.

[48] ——, "Intel® processors voltage settings modification advisory," 2019. [Online]. Available: https://www.intel.com/content/www/us/en/security-center/advisory/intel-sa-00289.html

[49] ——, "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, 2019-12-3.

[50] M. Jakobsson and A. Juels, "Proofs of work and bread pudding protocols," in *Secure Information Networks*. Springer, 1999, pp. 258–272.

[51] D. Johnson, A. Menezes, and S. Vanstone, "The elliptic curve digital signature algorithm (ecdsa)," *International journal of information security*, vol. 1, no. 1, pp. 36–63, 2001.

[52] D. Kaplan, J. Powell, and T. Woller, "Amd memory encryption," *White paper*, 2016.

[53] R. Khalil and A. Gervais, "Revive: Rebalancing off-blockchain payment networks," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2017, pp. 439–453.

[54] C. Kim, "Ethereum's istanbul upgrade arrives early, causes testnet split," *https://shorturl.at/bEQ29*, 2019.

[55] A. Kosba, A. Miller, E. Shi, Z. Wen, and C. Papamanthou, "Hawk: The blockchain model of cryptography and privacy-preserving smart contracts," in *2016 IEEE symposium on security and privacy (SP)*. IEEE, 2016, pp. 839–858.

[56] D. Lee, D. Kohlbrenner, S. Shinde, D. Song, and K. Asanović, "Keystone: A framework for architecting tees," *arXiv preprint arXiv:1907.10119*, 2019.

[57] J. Lind, I. Eyal, P. Pietzuch, and E. G. Sirer, "Teechan: Payment channels using trusted execution environments," *arXiv preprint arXiv:1612.07766*, 2016.

[58] J. Lind, O. Naor, I. Eyal, F. Kelbert, E. G. Sirer, and P. Pietzuch, "Teechain: a secure payment network with asynchronous blockchain access," in *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, 2019, pp. 63–79.

[59] lnd, "Lightning network daemon," https://github.com/lightningnetwork/lnd, 2019.

[60] loomx.io, "Loom: A new architecture for a high performance blockchain," https://loomx.io/, accessed: 2019-054-18.

[61] L. Luu, V. Narayanan, C. Zheng, K. Baweja, S. Gilbert, and P. Saxena, "A secure sharding protocol for open blockchains," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2016, pp. 17–30.

[62] G. Malavolta, P. Moreno-Sanchez, A. Kate, M. Maffei, and S. Ravi, "Concurrency and privacy with payment-channel networks," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2017, pp. 455–471.

[63] G. Malavolta, P. Moreno-Sanchez, C. Schneidewind, A. Kate, and M. Maffei, "Anonymous multi-hop locks for blockchain scalability and interoperability," in *Network and Distributed System Security Symposium (NDSS)*, 2019.

[64] S. Matetic, M. Ahmed, K. Kostiainen, A. Dhar, D. Sommer, A. Gervais, A. Juels, and S. Capkun, "ROTE: Rollback protection for trusted execution," in *26th USENIX Security Symposium (USENIX Security'17)*, 2017, pp. 1289–1306.

[65] S. Matetic, K. Wüst, M. Schneider, K. Kostiainen, G. Karame, and S. Capkun, "BITE: Bitcoin lightweight client privacy using trusted execution," in *28th USENIX Security Symposium (USENIX Security 19)*, 2019, pp. 783–800.

[66] mbed.org, "mbedtls:an open source, portable, easy to use, readable and flexible ssl library," https://tls.mbed.org/, accessed: 2019-12-3.

[67] P. McCorry, S. Bakshi, I. Bentov, S. Meiklejohn, and A. Miller, "Pisa: Arbitration outsourcing for state channels," in *Proceedings of the 1st ACM Conference on Advances in Financial Technologies*. ACM, 2019, pp. 16–30.

[68] P. McCorry, S. F. Shahandashti, and F. Hao, "A smart contract for board-room voting with maximum voter privacy," in *International Conference on Financial Cryptography and Data Security*. Springer, 2017, pp. 357–375.

[69] D. McGrew and J. Viega, "The galois/counter mode of operation (gcm)," *Submission to NIST Modes of Operation Process*, vol. 20, 2004.

[70] F. McKeen, I. Alexandrovich, A. Berenzon, C. V. Rozas, H. Shafi, V. Shanbhogue, and U. R. Savagaonkar, "Innovative instructions and software model for isolated execution." in *HASP@ISCA*, 2013, p. 10.

[71] A. Miller, I. Bentov, S. Bakshi, R. Kumaresan, and P. McCorry, "Sprites and state channels: Payment networks that go faster than lightning," in *International Conference on Financial Cryptography and Data Security*. Springer, 2019, pp. 508–526.

[72] D. Mingxiao, M. Xiaofeng, Z. Zhe, W. Xiangwei, and C. Qijun, "A review on consensus algorithm of blockchain," in *2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. IEEE, 2017, pp. 2567–2572.

[73] K. Murdock, D. Oswald, F. D. Garcia, J. Van Bulck, D. Gruss, and F. Piessens, "Plundervolt: Software-based fault injection attacks against intel sgx," in *2020 IEEE Symposium on Security and Privacy (SP)*, 2020.

[74] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," http://bitcoin.org/bitcoin.pdf, 2016.

[75] Z. Ning and F. Zhang, "Understanding the security of arm debugging features," in *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2019, pp. 602–619.

[76] R. Pass, E. Shi, and F. Tramer, "Formal abstractions for attested execution secure processors," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2017, pp. 260–289.

[77] Raiden, "The raiden network," https://raiden.network/, 2017.

[78] E. Rohrer, J. Malliaris, and F. Tschorsch, "Discharged payment channels: Quantifying the lightning network's resilience to topology-based attacks," *arXiv preprint arXiv:1904.10253*, 2019.

[79] F. Saleh, "Blockchain without waste: Proof-of-stake," *Available at SSRN 3183935*, 2020.

[80] M. Schwarz, S. Weiser, D. Gruss, C. Maurice, and S. Mangard, "Malware guard extension: Using sgx to conceal cache attacks," in *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 2017, pp. 3–24.

[81] A. SEV-SNP, "Strengthening vm isolation with integrity protection and more," *White Paper, January*, 2020.

[82] T. Swanson, "Consensus-as-a-service: a brief report on the emergence of permissioned, distributed ledger systems," *Report, available online*, 2015.

[83] S. Tochner, S. Schmid, and A. Zohar, "Hijacking routes in payment channel networks: A predictability tradeoff," *arXiv preprint arXiv:1909.06890*, 2019.

[84] A. Urquhart, "The inefficiency of bitcoin," *Economics Letters*, vol. 148, pp. 80–82, 2016.
[85] J. Van Bulck, M. Minkin, O. Weisse, D. Genkin, B. Kasikci, F. Piessens, M. Silberstein, T. F. Wenisch, Y. Yarom, and R. Strackx, "Foreshadow: extracting the keys to the intel SGX kingdom with transient out-of-order execution," in *27th USENIX Security Symposium (USENIX Security'18)*, 2018, pp. 991–1008.
[86] F. Vogelsteller and V. Buterin, "Erc-20 token standard," *Ethereum Foundation (Stiftung Ethereum), Zug, Switzerland*, 2015.
[87] N. Weichbrodt, A. Kurmus, P. Pietzuch, and R. Kapitza, "Asyncshock: Exploiting synchronisation bugs in intel sgx enclaves," in *European Symposium on Research in Computer Security*. Springer, 2016, pp. 440–457.
[88] www.blockchain.com, "Average confirmation time," https://www.blockchain.com/charts/avg-confirmation-time?timespan=all&daysAverageString=7, 2020.
[89] ——, "Blockchain size," https://www.blockchain.com/charts/blocks-size, 2020.
[90] M. Zamani, M. Movahedi, and M. Raykova, "Rapidchain: Scaling blockchain via full sharding," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018, pp. 931–948.
[91] F. Zhang, *mbedtls-SGX: a SGX-friendly TLS stack (ported from mbedtls)*, 2017. [Online]. Available: https://github.com/bl4ck5un/mbedtls-SGX
[92] F. Zhang, E. Cecchetti, K. Croman, A. Juels, and E. Shi, "Town crier: An authenticated data feed for smart contracts," in *Proceedings of the 2016 aCM sIGSAC conference on computer and communications security*. ACM, 2016, pp. 270–282.
[93] F. Zhang, I. Eyal, R. Escriva, A. Juels, and R. Van Renesse, "Rem: Resource-efficient mining for blockchains." *IACR Cryptology ePrint Archive*, vol. 2017, p. 179, 2017.
[94] Y. Zhang, S. Kasahara, Y. Shen, X. Jiang, and J. Wan, "Smart contract-based access control for the internet of things," *IEEE Internet of Things Journal*, vol. 6, no. 2, pp. 1594–1605, 2018.

APPENDIX

The ideal functionality: 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 [23] [24], 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.

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 $\mathcal{E}$ [23].

Proof. Let $\mathcal{E}$ be an environment and $\mathcal{A}$ be a real-world PPT adversary [23] 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

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

□

A. 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:**

- With honest $\mathcal{P}_i$, $\mathcal{S}$ obtains ("deposit", tx) from $\mathcal{F}_{\text{SPEEDSTER}}$, and emulates a call of "deposit" to $\mathcal{G}_{att}$ through "resume" interface.
- With corrupted $\mathcal{P}_i$, $\mathcal{S}$ reads tx from $\mathcal{E}$ and emulates message ("deposit", tx) to $\mathcal{F}_{\text{SPEEDSTER}}$ with the identity of $\mathcal{P}_i$. Then send 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",ccid, cid, $\mathcal{P}_i$, $\mathcal{P}_j$) 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, $\mathcal{R}$, 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, $\mathcal{R}$, inp) on behalf of $\mathcal{P}_i$ and collect the output from $\mathcal{G}_{att}$.
- Upon receiving ("open", ccid, cid, $\mathcal{P}_i, \mathcal{P}_j$) 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, $\mathcal{P}_j$, inp) on behalf of $\mathcal{P}_i$ and record the output from $\mathcal{G}_{att}$

(3) **Authenticate:**

Upon receiving message ($\mathcal{P}_i$, $\mathcal{P}_j$, "authenticate", cert) from $\mathcal{F}_{\text{SPEEDSTER}}$ of an honest $\mathcal{P}_i$, $\mathcal{S}$ 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 idccid from $\mathcal{E}$, a sk from a signature challenger SCh, then generates a random string as m. $\mathcal{S}$ computes $\sigma := \Sigma.\text{Sig}(\text{sk}, \text{pk}\|m)$, then sends the message ("authenticate", pk, ccid, $(\text{pk}_i\|\text{pk}\|m\|\sigma)$) 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 on behalf of $\mathcal{P}_i$ and collect the output from $\mathcal{G}_{att}$.

(4) **Multi-party State Channel:**

Upon receiving message ("openMulti", ccid, cid, $\{\text{ccid}\}^*$) from $\mathcal{F}_{\text{SPEEDSTER}}$ of an honest $\mathcal{P}_i$, $\mathcal{S}$ emulates a "resume" call to $\mathcal{G}_{att}$ sending message ("openMulti", cid, $\{\text{ccid}\}^*$). 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 $\{\text{ccid}\}^*$ and a smart contract id from $\mathcal{E}$. Then, $\mathcal{S}$ sends the message ("openMulti", cid, $\{\text{ccid}\}^*$) 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", ccid, cid, $\{\text{ccid}\}^*$) from $\mathcal{F}_{\text{SPEEDSTER}}$. $\mathcal{S}$ emulates a "resume" call to $\mathcal{G}_{att}$ sending message ("openMulti", cid, $\{\text{ccid}\}^*$). Then relay the output to $\mathcal{P}_i$.

(5) **Channel Transaction:**

Upon receiving message ("send", ccid, ℓ(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 $m := \mathcal{AE}.\text{Enc}(\text{key}, r)$, of which $|m| = |\ell(\text{msg})|$. $\mathcal{S}$ 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, $\{\text{ccid}\}^*$) 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, ℓ(msg)) from $\mathcal{F}_{\text{SPEEDSTER}}$. $\mathcal{S}$ requests a key from Ch. $\mathcal{S}$ computes $m := \mathcal{AE}.\text{Enc}(\text{key}, \overrightarrow{0})$, of which $|m| = |\ell(\text{msg})|$. $\mathcal{S}$ 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.

While $\mathcal{P}_i$ is corrupted. $\mathcal{S}$ sends the message ("claim") to $\mathcal{F}_{\text{SPEEDSTER}}$ on behalf of $\mathcal{P}_i$ and collect 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 $\mathcal{G}_{att}$, then 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 $\mathcal{P}_i$: (pk$_i$, sk$_i$) $\leftarrow_\$$ KGen(1^n)

(1) **On receive** ("deposit", tx) from $\mathcal{P}_i$ where $i \in [N]$:
parse tx as (pk', \$val, _, σ)
Verify signature of tx, abort if false
bals[$\mathcal{P}_i$] += \$val
append (tx, \$val) to states^0[$\mathcal{P}_i$]
leak ("deposit", tx) to $\mathcal{A}$

(2) **On receive** ("open", cid, $\mathcal{P}_j$) from $\mathcal{P}_i$ where $i, j \in [N]$ and $\mathcal{P}_i \neq \mathcal{P}_j$:
ccid $\leftarrow_\$ \{0,1\}^*$
state$_{\text{ccid}}$:= Contract$_{\text{cid}}$(pk$_i$, $\overrightarrow{0}$, $\perp$)
append (ccid, (cid, state$_{\text{ccid}}$, $\{\mathcal{R}, \mathcal{P}_i\}$)) to channels
leak ("open", ccid, cid, $\mathcal{P}_i$, $\mathcal{P}_j$) 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 $i \in [N]$ and $\mathcal{R} \neq \mathcal{P}_i$:
ccid $\leftarrow_\$ \{0,1\}^*$
state := Contract$_{\text{cid}}$($\mathcal{P}_i$, $\overrightarrow{0}$, $\perp$)
collect dummy parties $\{\mathcal{P}\}^*$ in channels $\{$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) := Contract$_{cid}$($\mathcal{P}_i$, state, inp)
msg := ($\mathcal{P}_i \| r \|$inp$\|$state'$\|$outp)
leak ("send", ccid, ℓ(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 $\in$ [N]:
construct an on-chain claim transaction tx
leak("claim", tx) to $\mathcal{A}$; await "OK" from $\mathcal{A}$
send(tx) to Blockchain

Figure 6. Ideal functionality of $\mathcal{F}_{\text{SPEEDSTER}}$. Internal communications are assumed to be encrypted with authenticated encryption.

B. Indistinguishability

We show that the execution of the real-world and ideal-world is indistinguishable for all $\mathcal{E}$ from the view of $\mathcal{A}$ 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 $\mathcal{F}_{blockchain}$,

• **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 in Appendix. A, $\mathcal{S}$ 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 $\Pi_{\text{SPEEDSTER}}$ and $\mathcal{A}$ with $\mathcal{S}$ and $\mathcal{F}_{\text{SPEEDSTER}}$.

C. Single Deposit for Opening Channel:

Certified Channel enables SPEEDSTER to open a new channel without locking any fund in the channel; therefore, one-time fund deposit into acc$_{enclave}$ on the main chain is enough to create many channels; With fund in the acc$_{enclave}$ securely managed, even the owner cannot access it unless through a properly defined interface. Thereafter, no more on-chain transactions are needed. With n participants in the system, at most $\tau * n$ amount of fee is needed to start up the system where each participant makes a deposit.

$\mathcal{G}_{att}[\Sigma, reg]$

// initialization:
On initialize: $(\mathsf{mpk}, \mathsf{msk}) := \Sigma.\mathsf{KGen}(1^n), T = \emptyset$

// public query interface:
On receive* getpk() from some $\mathcal{P}$: send mpk to $\mathcal{P}$

Enclave operations

//local interface | install an enclave:
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 $T[eid, \mathcal{P}] := (idx, \mathsf{prog}, 0)$, send eid to $\mathcal{P}$

// local interface | resume an enclave:
On receive* resume($eid, inp, switch := on$) from some $\mathcal{P} \in reg$:
let $(idx, \mathsf{prog}, mem) := T[eid, \mathcal{P}]$, abort if not found
let $(outp, mem) := \mathsf{prog}(inp, mem)$,
update $T[eid, \mathcal{P}] := (idx, \mathsf{prog}, mem)$
if $switch$ is set to on
let $\sigma := \Sigma.\mathsf{Sig}_{\mathsf{msk}}(idx, eid, \mathsf{prog}, outp)$
send $(outp, \sigma)$ to $\mathcal{P}$
otherwise:
send $(outp, \perp)$ to $\mathcal{P}$

Figure 8. A global functionality modeling an SGX-like secure processor. Compared to [76], a switch is added to the "resume" command to allow users to disable the signature. The default value of $switch$ is set to "on".

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

$\mathsf{contract}_{\text{SPEEDSTER}}$

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 $\notin Ledger$
execute tx on the $Blockchain$
append tx to $Ledger$

On receive ("claim", tx) from $\mathcal{P}$:
parse tx as $\{\mathsf{cert}, \mathsf{state}\}^*$
For each (cert, state) in $\{\mathsf{cert}, \mathsf{state}\}^*$:
parse cert_i to (to′, from′, $val′, σ)
abort if $\mathsf{Verify}(\sigma, \mathsf{cert}_i)$ fails
assert from = from′ && to = to′ = $\mathcal{P}$ && $val $\leq$ $val′
send(from, $\mathcal{P}$, Coin($val))
append(tx) to $Ledger$

On receive ("read", tx) from $\mathcal{P}$:
output $Ledger$[tx]

Figure 7. On-chain smart contract $\mathsf{contract}_{\text{SPEEDSTER}}$ of $\Pi_{\text{SPEEDSTER}}$

D. Single Claim for Closing Channel:

In SPEEDSTER, we expect the participant who has more funds in the channel is better motivated to carry out the channel settlement, so one transaction is sufficient in this scenario. For instance, a *Certified Channel* with *Alice* and *Bob* as two participants. If *Bob* sends more funds to *Alice*, then *Alice* is the one who sends the claim transaction to close the channel, vice versa; If there is no balance change in the channel, anyone can submit the claim transaction.[2] Formally, denote x as the fund *Alice* sends to *Bob* and y as the fund *Bob* sends to *Alice*, the party to send the claim transaction is decided by:

$$claimer = \begin{cases} Alice & \text{if } x < y \\ \perp & \text{if } x = y \\ Bob & \text{if } x > y \end{cases} \quad (1)$$

One single transaction can be used to claim all the fund from multiple channels associated with the claimer. With a n node network, the worst scenario is that every node needs to issue a claim transaction; henceforth, the fee to claim all fund in the system is $\leq \tau * n$. Table **??** shows the comparison between *Certified Channel* and state channel on transaction fees for opening/closing a channel.

We briefly discuss the cases where there might be message delay/loss in a multi-party transaction process.

[2]In practice, we can designate the party who sent the last transaction in the channel to start the claim process.

E. Late/lost messages from a rightful node

In Figure 10.b, when a transaction tx_{i-1} from A is late on reaching the next rightful node B, B will query A for tx_{i-1} in case of transaction lost.

F. Waiting time expires

Assume a corrected timer is set up on each node. As showing in Figure 10.c, if the timer for the transaction tx_i from node B expires, A and C will broadcast skipping messages to indicate that they agree on skipping B for this turn. Once C receives the skip message from A, and it also agrees to skip B, then C will initiate the generation of transaction tx_i.

G. A message from the next rightful node is received

Figure 10.d demonstrates that node C receives tx_i when it is still waiting for tx_{i-1} from A. This means that the transaction tx_{i-1} sent to C is lost. In this case, C can request tx_{i-1} from B.

Protocol $\Pi_{\text{SPEEDSTER}}[\mathcal{P}_0, \mathcal{P}_1, \mathcal{P}_2, ..., \mathcal{P}_N]$

Program $\text{prog}_{enclave}$

Initially:
bal := $\emptyset$, certs := $\emptyset$, channels := $\emptyset$, state^0 := $\perp$

(1) **On receive**("init")
(pk, sk) $\leftarrow_{\$}$ KGen(1^n)
mpk := $\mathcal{G}_{att}$.getpk()
return (pk, mpk)

(2) **On receive** ("deposit", tx)
parse tx as (_, pk', $val, σ)
assert state^0 $\neq\perp$ and $val ≥ 0
assert Verify(pk, tx, σ) is true
bal += $val; assign (tx, $val) to state^0

(3) **On receive** ("open", cid, $\mathcal{R}$, inp)
ccid := H($SORT${pk$_{\mathcal{R}}$, pk})
abort if channels[ccid] $\neq\perp$
ck $\leftarrow_{\$}$ $\{0,1\}^*$ // channel key
cp := {pk, pk$_{\mathcal{R}}$}
(st, outp) := Contract$_{\text{cid}}$(sk, bal, $\overrightarrow{0}$, cp)
append (ccid, (ck, cid, st, cp)) to channels
σ = sign(sk, pk$_R$||inp), cert = (pk||pk$_R$||inp||σ)
return (cert, state^0, outp)

(4) **On receive** ("openMulti", cid, {ccid}*)
for each ccid' $\in$ {ccid}*:
assert channels[ccid'] $\neq\perp$
extract pk' from channels[ccid']
cp := {{pk'}* $\cup$ pk}; ccid := H($SORT$(cp))
assert channels[ccid] = $\perp$
gk $\leftarrow_{\$}$ $\{0,1\}^*$ //Group key
(st, outp) := Contract$_{\text{cid}}$(sk$_i$, state, $\overrightarrow{0}$, cp)
append (ccid, (gk, cid, st, cp)) to channels
ct := Enc (gk, outp)
return (ct)

(5) **On receive** ("authenticate", ccid, $\mathcal{R}$, cert)
abort if certs[ccid][pk$_{\mathcal{R}}$] $\neq\perp$
parse cert as (msg, σ)
assert Verify (pk$_{\mathcal{R}}$, msg, σ)
set certs[ccid][pk$_{\mathcal{R}}$] to cert

(6) **On receive** ("send", ccid, inp):
assert certs[b] $\neq\perp$
(ck, cid, st, cp) := channles[ccid]
(st', outp) := Contract$_{\text{cid}}$(sk, state, st, inp)
update channels[ccid] to (ck, cid, st', cp)
msg := (pk||inp||st'||outp); ct := Enc (ck, msg)
return (ct)

(7) **On receive** ("claim")
freeze **send** function
tx := {cert}*||state; σ := Sign (sk,tx)
return (tx||σ)

Figure 9. $\text{prog}_{enclave}$ program of $\Pi_{\text{SPEEDSTER}}$

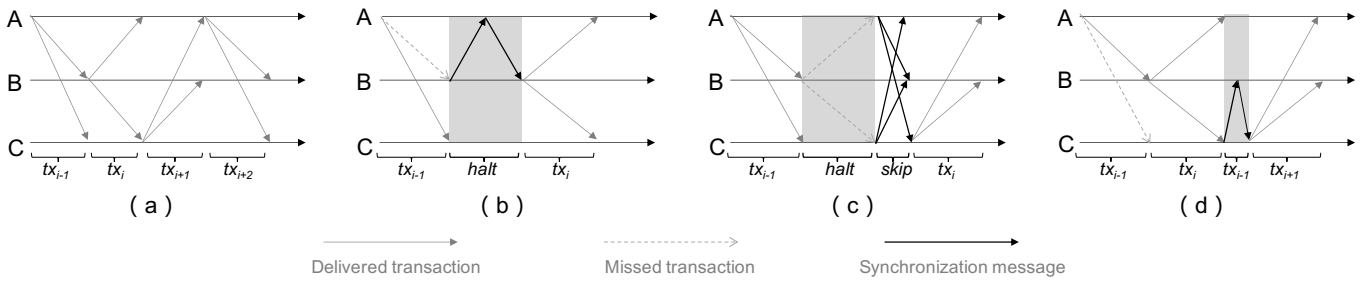

Figure 10. Multi-party state channel example: three-party state channel with the order of nodes A, B, and C. Arrowhead line demonstrates the channel transactions, indexed by i; dash line means that the transaction is lost. Subfigure (a) shows the ordered message broadcast; (b) demonstrates a message lost; (c) shows a late-response node; (d) shows the process to skip a non-responding node.