Cloak: Transitioning States on Legacy Blockchains Using Secure and Publicly Verifiable Off-Chain Multi-Party Computation

Qian Ren
qianren1024@gmail.com
SSC Holding Company Ltd.
Chengmai, China
Oxford-Hainan Blockchain Research Institute
Chengmai, China

Yingjun Wu
yingjun@oxhainan.org
SSC Holding Company Ltd.
Chengmai, China
Oxford-Hainan Blockchain Research Institute
Chengmai, China

Han Liu
liuhan0518@163.com
Oxford-Hainan Blockchain Research Institute
Chengmai, China
Tsinghua University
Beijing, China

Yue Li
liyue@oxhainan.org
Oxford-Hainan Blockchain Research Institute
Chengmai, China

Anne Victor
anne.victor@outlook.com
SSC Holding Company Ltd.
Chengmai, China
Oxford-Hainan Blockchain Research Institute
Chengmai, China

Hong Lei
leihong@oxhainan.org
Hainan University
Haikou, China
Oxford-Hainan Blockchain Research Institute
Chengmai, China

Lei Wang
wanglei@cs.sjtu.edu.cn
Shanghai Jiao Tong University
Shanghai, China

Bangdao Chen
bangdao@oxhainan.org
SSC Holding Company Ltd.
Chengmai, China
Oxford-Hainan Blockchain Research Institute
Chengmai, China

ABSTRACT

In recent years, the confidentiality of smart contracts has become a fundamental requirement for practical applications. While many efforts have been made to develop architectural capabilities for enforcing confidential smart contracts, a few works arise to extend confidential smart contracts to Multi-Party Computation (MPC), *i.e.*, multiple parties jointly evaluate a transaction off-chain and commit the outputs on-chain without revealing their secret inputs/outputs to each other. However, existing solutions lack public verifiability and require $O(n)$ transactions to enable negotiation or resist adversaries, thus suffering from inefficiency and compromised security.

In this paper, we propose Cloak, a framework for enabling Multi-Party Transaction (MPT) on existing blockchains. An MPT refers to transitioning blockchain states by an *publicly verifiable* off-chain MPC. We identify and handle the challenges of securing MPT by harmonizing TEE and blockchain. Consequently, Cloak secures the off-chain nondeterministic negotiation process (a party joins an MPT without knowing identities or the total number of parties until the MPT proposal settles), achieves public verifiability (the public can validate that the MPT correctly handles the secret inputs/outputs from multiple parties and reads/writes states on-chain), and resists Byzantine adversaries. According to our proof, Cloak achieves better security with only 2 transactions, superior to previous works that achieve compromised security at $O(n)$ transactions cost. By evaluating examples and real-world MPTs, the gas cost of Cloak reduces by 32.4% on average.

CCS CONCEPTS

• **Security and privacy** → **Security protocols**; **Privacy-preserving protocols**; *Distributed systems security*.

ACM Reference Format:
Qian Ren, Yingjun Wu, Han Liu, Yue Li, Anne Victor, Hong Lei, Lei Wang, and Bangdao Chen. 2022. Cloak: Transitioning States on Legacy Blockchains Using Secure and Publicly Verifiable Off-Chain Multi-Party Computation. In *Annual Computer Security Applications Conference (ACSAC '22), December 5–9, 2022, Austin, TX, USA.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3564625.3567995

1 INTRODUCTION

With the rapid development of blockchains, privacy issues have become one of the top concerns for smart contracts. Unfortunately, despite the importance of smart contract privacy, most existing blockchains are designed *without privacy* by nature [28, 45]. For

---

ACSAC '22, December 5–9, 2022, Austin, TX, USA

ACM ISBN 978-1-4503-9759-9/22/12…$15.00
https://doi.org/10.1145/3564625.3567995

Table 1: Comparison of Cloak with related works. Here, ●, ◗, ❍, × denotes full, partial, not matched and not related, respectively. "Adversary Model" denotes how many entities' misbehavior are considered, where an executor denotes a server hosting TEE. "min(#TX)" denotes how many transactions are required by the approach. "Public Verifiability" denotes all elements are committed on-chain and state transition can be validated, where x denotes transaction parameter, s, s' denotes contract old and new states respectively, f denotes target function, r denotes return value, and $\mathcal{P}$ denotes privacy policy that includes party-input bindings, *etc.* "Financial Fairness" denotes that honest parties never lose their collateral without obtaining outputs.

Approach	Adversary Model		Chain Agnostic	min(#TX)	Confidentiality	Nondeterministic Negotiation	Public Verifiability						Financial Fairness
	#Parties	#Executors					x	s	f	r	s'	$\mathcal{P}$	
Ethereum [45]	1^*	×	×	$O(1)$	×	×	●	●	●	●	●	●	×
Ekiden [13]	1^*	$m^* - 1$[1]	●	$O(1)$	●	×	❍[2]	●	●	❍[2]	●	●	×
Confide [27]	1^*	$\lfloor m^*/3 \rfloor$[3]	❍	$O(1)$	●	×	●	●	●	●	●	●	×
Hawk [25]	n^*	×	●	$O(n)$	◗[4]	❍	●	❍	●	●	❍	❍	●
ZEXE [7]	n^*	1^*	❍	$O(1)$	◗	❍	●	●	●	●	●	❍	×
Fastkitten [16]	$(n^* + 1^*) - 1$		❍	$O(n)$	◗	❍	❍	❍	❍	●	❍	❍	●
LucidiTEE [37]	n^*	$m^* - 1$	●	$O(n)$	●	●	●	◗[5]	●	●	◗[5]	◗[5]	×
Cloak	$(n^* + 1^*) - 1$[6]		●	$O(1)$	●	●	●	●	●	●	●	●	●

[1] The * denotes the total number of specific kinds of entities assumed in the system, *e.g.*, 1^* denotes the unique party/executor, n^* denotes all n parties, and m^* denotes all executors in the system. [2] Transaction parameters x (resp. return values r) in Ekiden are received (resp. delivered) off-chain while not committed on-chain. [3] We assume Confide's undeclared consensus is BFT. [4] The manager is expected not to leak parties' private data. [5] Fastkitten does not commit the inputs and states of MPT. [6] LucidiTEE does not consider verifying the state transition with policy on-chain.

example, miners of Ethereum verify transactions in a block by re-executing them with the exact input and states. Consequently, the transaction data have to be shared within the entire network.

Confidential smart contract with MPC. To address the aforementioned problem, researchers have proposed various *confidential smart contract* solutions, *i.e.*, keeping transaction inputs and contract states secret from non-participants. In parallel, a few works expand the transaction of confidential smart contracts to Multi-Party Computation (MPC)s, which means allowing multiple parties jointly evaluate a transaction off-chain and commit the outputs on-chain without revealing their secret inputs/outputs to each other. These works fall into two categories. The first adopts cryptographic MPC primitives (based on secret sharing [41], HE [39], and ZKP [7, 25], *etc.*) to let multiple parties jointly evaluate a transaction off-chain and optionally record or partially prove the evaluation on-chain. The other category adopts TEE to collect sealed inputs from different parties, reveal the inputs and evaluate a program inside enclaves to obtain the outputs [16, 26]. While both categories achieve confidentiality of MPC, few of them achieve public verifiability. Qian *et al.* [32] call the need for publicly verifiable MPC transaction in reason that the transaction should prove to non-participants of its MPC, especially regulators or miners, to let them trust the state transition the transaction caused. Qian *et al.* [32] furthermore firstly define a problem Multi-Party Transaction (MPT), which refers to multiple parties jointly evaluating a transaction off-chain based on publicly verifiable MPC to transition states on-chain, while keeping each secret input/output confidential to its corresponding party.

Limitations and Challenges. In this paper, we aim to support MPT-enabled confidential smart contracts on legacy blockchains, which poses several challenges that existing efforts fail to handle.

C1: Nondeterministic negotiation with minimal cost. In the real world, users should be allowed to join an MPT without knowing other parties' information prior, *e.g.*, bidders can independently decide to join an auction, as the number and identities of all bidders are settled only when the bidding phase closes. We call this nondeterministic negotiation. However, practically secure nondeterministic negotiation is nontrivial. Previous approaches either assume that protocols start with pre-known settings [16, 25, 42] (program, parties, time duration, *etc.*) to bypass the challenge, or require each party to send transactions on-chain thereby causing $O(n)$ transactions [37]), or assume parties negotiate by P2P communications or assistance of a semi-honest Trusted Third Party (TTP) [12], thus vulnerable to Byzantine adversary. Consequently, securing off-chain negotiation under Byzantine adversary at the cost of $O(1)$ transactions is still a challenge.

C2: Publicly verifiable MPC-based state transition. To transition blockchain states by off-chain MPCs [16, 37, 41], Qian *et al.* [32] stress that the public, including miners, should also verify the state transition the MPC caused without trusting any parties or executors of the MPC, and identifies the problem as a new problem MPT. However, Qian *et al.* [32] fails to present an corresponding capable and secure protocol. Existing cryptographic solutions for MPC under malicious adversaries perform well on achieving confidentiality, but cannot achieve the public verifiability required by MPT. Specifically, [16, 24, 37, 42] sporadically record part information of an off-chain MPC evaluation (inputs, outputs, states, *etc.*) on the blockchain, failing to uniquely identify the evaluation, not to mention prove it. Moreover, for miners/regulators who neither are nor trust any MPC participants, the participants cannot only record or multi-sign messages to convince miners/regulators that the MPC-caused state transition holds authenticity and correctness. Consequently, it is still a challenge to construct a general and succinct $proof$ to achieve MPTs.

C3: Byzantine adversary resistance with minimal cost. To punish off-chain misbehaviors like aborting protocols, previous work [16, 31, 43, 44] involve fine-tuned challenge-response mechanisms. These mechanisms require all parties to independently deposit collateral on-chain before the protocol starts, thus leading to at least $O(n)$ transactions, which is impractical for scalability.

Our work. In this paper, we propose a novel MPT-enabled confidential smart contract framework, Cloak, by harmonizing the blockchain with a unique TEE-enabled executor. Furthermore, we design and prove a currently most secure and practical protocol for serving MPT, under the assumption that a Byzantine adversary can arbitrarily corrupt parties or the executor but cannot break the integrity of the TEE itself.

Contributions. Our main contributions are as follows:

- We propose a novel confidential smart contract framework, which can transition the state of existing blockchains by transactions based on publicly verifiable MPC, *i.e.*, MPT.
- With Byzantine adversary assumed, we design a protocol to achieve trusted off-chain nondeterministic negotiation (against **C1**), public verifiability (against **C2**), and financial fairness (against **C3**) for MPT, at the cost of only 2 transactions.
- We formally define and prove the security properties that Cloak achieved in a Byzantine adversary model.
- We have applied Cloak in several real-world scenarios. Cloak achieves MPT with both lower gas costs and better performance.

2 RELATED WORK

In this section, we elaborate how Cloak is distinct from current confidential smart contract solutions. Table 1 shows the comparison between Cloak and some representative solutions.

TEE-enforced confidential smart contracts. Ekiden [13, 38] reveals and executes smart contracts in SGX to conceal transaction parameters, return values, and contract states. CCF [33] supports any typescript or C++-based application in a TEE-based permissioned blockchain. Confide [27] synchronizes a common public key between all SGX and runs EVM and WASM in SGX to support various contracts. CCF, Confide, and Ekiden integrate TEE into their own consensus pipeline, thus making them chain-specific. In contrast, Cloak enables MPT on an existing blockchain by deploying merely a contract facility and is thus chain-agnostic. For scalability, each transaction of [13, 27, 33, 38] is from a single sender and validated by all consensus nodes. Therefore, they do not consider negotiation, and serving MPT with these solutions takes at least $O(n)$ transactions from parties. For public verifiability, CCF and Confide ignore off-chain inputs/outputs. Ekiden considers off-chain inputs/outputs and can verify state transition on-chain. However, Ekiden does not commit off-chain data on-chain, therefore the transaction cannot be identified. Conversely, Cloak considers and commits all necessary elements of MPT evaluation on-chain, enabling it to identify the evaluation. For security, none of the above solutions can punish misbehaving transaction sender or executors. Instead, Cloak secures off-chain inputs submission and outputs delivery, and achieves financial fairness.

Cryptography-based smart contracts with MPC. A cryptography-based contract with MPC refers to multiple parties jointly evaluating a transaction based on cryptographic schemes. In terms of scalability, MPC-based approaches [31, 43, 44] allow m-round MPC with penalties over Bitcoin but rely on claim-or-refund functionality, which necessitates complex and expensive transactions and collateral, thus making these solutions impractical. Hawk [25] also requires $O(n)$ transactions to punish misbehaved parties. In terms of confidentiality, Hawk requires a credible manager to withhold information, thus achieving limited confidentiality. ZEXE [7] proves the satisfaction of predicates with ZKP without revealing party secrets to the public. However, to generate the proof for a predicate, a party must be privy to the predicate's secrets, thus violating inter-party confidentiality. Instead, Cloak keeps parties' confidential data confidential from each other and the public. In terms of negotiation, both Hawk and ZEXE start from pre-specified parties, assuming that parties have previously negotiated through off-chain P2P communications or a TTP. In terms of public verifiability, the manager of Hawk updates contract state with ZKP proof, making the off-chain multi-party process transparent and unverifiable to verifiers, *e.g.*, miners. Public auditable MPC (PA-MPC) [5] achieves the publicly verifiable MPC, allowing multiple parties jointly evaluate a program and prove it. Nevertheless, existing PA-MPC primitives are not designed for committing data or proving state transitions, *e.g.*, MPCs expressed in Solidity that operate both on- and off-chain inputs/outputs. Moreover, they have flaws at inefficiency and weaker adversary model, and still fail in practically supporting nondeterministic negotiation or achieving financial fairness. Specifically [5, 34] requires trusted setup or un-corrupted parties. [4] is function-limited. [30] very recently achieves general-purpose PA-MPC but only support circuit-compatible operations. None of above solutions are for confidential smart contracts or can punish adversaries.

TEE-based smart contracts with MPC. For confidentiality, Fastkitten [16] does not consider confidentiality. For negotiation, both Fastkitten and Speedster [26] assume parties are known prior. LucidiTEE [37] requires parties to independently send transactions on-chain to join a MPT; therefore, it achieves nondeterministic negotiation but flaws at $O(n)$ transactions. Cloak enables negotiation with the cost of 1 transaction. For public verifiability, both FastKitten and Speedster records only the final outputs of m-round MPC on the blockchain. LucidiTEE does not distinguish between states of different parties. Cloak proves and uniquely identifies the evaluation on-chain. For security, while LucidiTEE and Speedster lack collateral systems to punish malicious parties, both Hawk and FastKitten adopt challenge-response protocols to punish malicious parties. FastKitten achieves financial fairness but requires each party to deposit collateral for each evaluation, suffering from $O(n)$ transaction cost. In contrast, Cloak achieves financial fairness with the cost remaining $O(1)$.

To the best of our knowledge, Cloak is the first to enable existing blockchains to transition states by MPTs. Cloak is also advanced in securing off-chain nondeterministic negotiation process and resisting Byzantine adversary by normally only 2 transactions.

3 CLOAK OVERVIEW

This section presents an overview of Cloak, a novel MPT-enabled confidential smart contract framework. We first model the architecture of Cloak, and then introduce the workflow of Cloak, as well as our adversary model. Finally, we introduce our security goals and design challenges.

3.1 System model

Conceptually, to support multiple parties evaluate MPT. Cloak adopts a hybrid architecture combining a blockchain and a unique executor with TEE. Figure 1 depicts the architecture of Cloak and a workflow of the Cloak protocol. There are mainly four entities:

Blockchain (*BC*). A blockchain in Cloak is a normal contract-enabled blockchain, *e.g.*, Ethereum. It is responsible for maintaining the commitments of parameters, returns, states, and the MPC of MPT and validating the state transition.

Parties ($\bar{P}$). Parties are participants of MPT. They interact with the TEE enclave to negotiate the MPT setting, feed inputs and receive outputs. They also interact with the blockchain to monitor MPT status and punish the misbehaving executor.

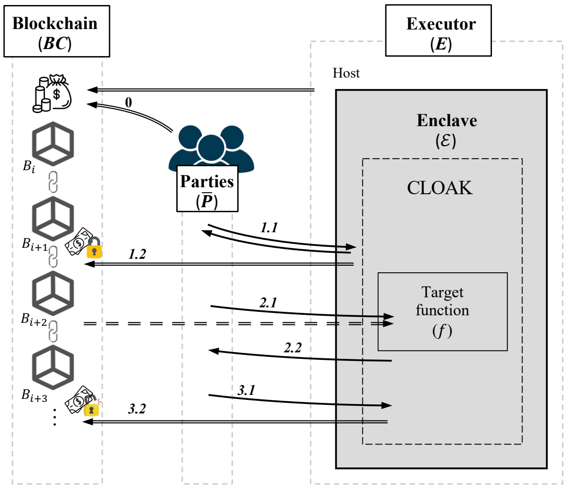

Figure 1: Overall workflow of Cloak framework

Executor (*E*). The executor is a server holding a TEE. It is responsible for instantiating the TEE enclave and relaying messages between parties, the blockchain and the enclave to proceed with the Cloak protocol.

Enclave ($\mathcal{E}$). The enclave runs the Cloak enclave program (Algorithm 2). It is responsible for receiving inputs from parties and the blockchain, evaluating the MPT inside the enclave, and delivering the outputs to the blockchain and parties.

3.2 Adversary model

In our system, n parties and a unique executor ***E*** (who owns the TEE $\mathcal{E}$) follows the Cloak protocol π_{Cloak} to enable MPT on an existing blockchain. Our assumptions and threat model are as follows:

TEE. We assume that the adversary cannot break the integrity and confidentiality of TEE. Although we instantiate the TEE as SGX, our design is TEE-agnostic. We stress that although recent research showed some attacks against TEE, the confidentiality and integrity guarantees of TEE are still trustworthy, making our assumption of TEE practical. We elaborate the rationality of this TEE assumption in Appendix B

Blockchain. We assume the common prefix, chain quality, and chain growth of the blockchain are held so that the blockchain constantly processes and confirms new transactions and is always available. In particular, we assume that the blockchain supports Turing-complete smart contracts, *e.g.*, Solidity, so that we can deploy a contract program (Algorithm 1) on the blockchain to manage the life cycle of MPT. Finally, while Cloak is designed to be agnostic to the underlying consensus protocol, *we assume the blockchain can construct a Proof of Publication (PoP) of transactions for proving that a transaction has been confirmed on the blockchain, which is similarly assumed by [11, 13, 16] for resisting Eclipse attacks.*

Parties. Honest parties trust their own platforms but never trust other parties or ***E***. Honest parties trust the data accessed from blockchain and attested TEE.

Threat model. We assume a *Byzantine adversary* is present in our system. The adversary can corrupt all but one subject among parties and ***E***. On compromised parties or ***E***, the adversary can behave arbitrarily, *e.g.*, scheduling processes as well as reordering, delaying, or mutating messages but can never break the integrity and confidentiality of TEE.

3.3 Design goals

We aim to achieve the following five security properties. These properties are formally defined in Appendix C.3.

Correctness. If an MPT succeeds, its outputs are the correct outputs of the program, the target function, applied to the committed inputs.

Confidentiality. Without any compromised TEE, Cloak guarantees that both the inputs and outputs of MPT are kept secret to their corresponding parties.

Public verifiability. The public with only on-chain data can verify that a state transition is correctly caused by an MPT, and the MPT is jointly evaluated using committed program, privacy policy, old states, parameters, causing committed return values and new states.

Executor balance security. If the executor ***E*** honestly behaves, it cannot lose money.

Financial Fairness. Honest parties should never lose their collateral. Specifically, if at least one subject among both parties and ***E***, is honest, then either (i) the negotiation failed, and all parties stay financial neutral, or (ii) the protocol correctly evaluates the MPT and all parties stay financial neutral, or (iii) all honest subjects among parties and ***E*** know the protocol has aborted and stay financially neutral and at least one malicious subject among parties and ***E***, must be financially punished.

In addition to the foregoing goals of this paper, we also clarify what are not our goals here. The confidentiality broken by parties voluntarily revealing their own secrets to anyone else except TEE lies outside our consideration. Resisting the potential secret leakage caused by the MPC's target function is also not our goal.

3.4 Workflow

As shown in Figure 1, parties follow the Cloak protocol to interact with the ***BC*** and ***E*** in order to send MPT. Our protocol proceeds in four phases. In particular, while the setup phase (***0***) occurs only once for the ***E*** and each party, the remaining phases (***1-3***) are repeated for each MPT. We assume that the public key $pk_{\mathcal{E}}$ and address $adr_{\mathcal{E}}$ of enclave $\mathcal{E}$ are both previously registered on-chain. All parties have verified that $\mathcal{E}$ has been initialised correctly by inspecting its attestation report. Utilizing verified $pk_{\mathcal{E}}$, parties can establish secure channels with $\mathcal{E}$. In the following, as ***E*** is responsible for relaying input and output messages of its $\mathcal{E}$, we illustrate the protocol in the same way that $\mathcal{E}$ directly communicates with ***BC*** and $\bar{P}$, rather than explicitly marking ***E***'s relaying behaviour each time.

- **(Global) Setup phase (*0*)**: ***E*** and all parties independently deposit their coins to $\mathcal{E}$'s address $adr_{\mathcal{E}}$. In subsequent phases, the collateral required to join each MPT is deducted from these coins.
- **(MPT) Negotiation phase (*1.1-1.2*)**: One of the parties sends an MPT proposal to $\mathcal{E}$ without knowing the identities of other parties. Then, $\mathcal{E}$ starts a *nondeterministic negotiation protocol* (***1.1***). To begin, $\mathcal{E}$ generates an id for the proposal and broadcasts the signed proposal with the id to all parties. If any party is interested in or required by the proposal, it responds an acknowledgment to $\mathcal{E}$ with signed commitments of parameters. $\mathcal{E}$ continues to collect parties' acknowledgments until the negotiation phase ends or the collected acknowledgments satisfy the negotiation's settlement condition. Following that, $\mathcal{E}$ sends a TX_p to ***BC*** (***1.2***). The TX_p

publishes the MPT proposal along with all parties' identities and parameter commitments to indisputably announce the negotiation outcome on-chain. Additionally, TX_p deducts collateral from $\boldsymbol{E}$ and all parties for the MPT in case any of them aborts the MPT after the negotiation succeeds.

- **(MPT) Execution phase (*2.1*-*2.2*)**: After TX_p is confirmed on ***BC***, each party sends their signed inputs (*i.e.*, parameters and old states) to $\mathcal{E}$. Upon receiving these inputs, $\mathcal{E}$ reads the blockchain view to ensure that TX_p was indeed been confirmed on ***BC***. The confirmation of TX_p indicates that parties' collateral has been successfully deducted and parameters committed (***2.1***). Then, $\mathcal{E}$ verifies that if the collected parameters and old states match their on-chain commitments. If the verification succeeds, $\mathcal{E}$ evaluates the MPT program to obtain outputs (*i.e.*, return values and new states). Following that, $\mathcal{E}$ only delivers the ciphertext of outputs to parties off-chain (***2.2***).
- **(MPT) Distribution phase (*3.1*-*3.2*)**: After $\mathcal{E}$ collects all parties' receipts (***3.1***), $\mathcal{E}$ sends a TX_{com} to commit the outputs on ***BC*** along with the encryption key of delivered output ciphertext (***3.2***). Thus, all parties accessing the key in TX_{com} can decrypt the output ciphertext received in ***2.2***.

3.5 Design challenges and highlights

In this section, we highlight the challenges handled in the workflow and high-level ideas of their corresponding countermeasures.

3.5.1 *Securing off-chain nondeterministic negotiation (against **C1**).*

In a decentralized and open network, there are undoubtedly scenarios in which a party joins an MPT unaware of the other parties, *e.g.*, a public auction in which bidders self-select to participate without knowing others until the bidding process closes. The nondeterministic negotiation is for parties to negotiate a MPT proposal without knowing others until the proposal is settled. An MPT proposal can be exemplified as $p' \leftarrow (C_f, C_{\mathcal{P}}, q, \bar{\boldsymbol{P}}, C_x)$, where C_* denotes the hash commitment of $*$. Therefore, the proposal p' specifies which MPT f to evaluate, which policy $\mathcal{P}$ to enforce, which parties $\bar{\boldsymbol{P}}$ are required to participate and their corresponding inputs x, and how much collateral of misbehaving parties to punish. Previous works [16, 25, 42] assume that MPT proposal is known prior. Although [37] enables parties to autonomously bind inputs on-chain to join a specific MPT proposal, it incurs a cost of $O(n)$ transactions. One may believe that we can require all parties to communicate with a TEE off-chain in order to negotiate with other parties without interacting with a blockchain. However, if we do not settle the negotiation on-chain, the blockchain will neither know when the MPT begins (which is critical for timeout judgement) nor capable of freezing all collateral of parties and the executor before the evaluation. Consequently, the adversary can arbitrarily drops or delays off-chain data without being identified or penalised.

In this paper, we propose an *nondeterministic negotiation* subprotocol to support the nondeterministic participation of MPT parties. The main idea is to allow a party to initiate a negotiation process by sending an MPT proposal. After parties agree on the MPT proposal, they can send their acknowledgements and parameter commitments to join the MPT. When the negotiation is complete, the TEE attaches party identities and parameter commitments to the proposal to obtain a settled proposal. The settled proposal is then published on the blockchain by TEE. Thus, both parties and TEE proceed to the next phase based on the blockchain-confirmed proposal. The blockchain knowing when the MPT begins is capable of judging whether the MPT timeouts.

3.5.2 *Achieving public verifiability of MPT (against **C2**).*

The challenge of achieving public verifiability of MPT is constructing an interpretable $proof$ whose size is independent of x, s, f, r, s' and the privacy policy $\mathcal{P}$. $\mathcal{P}$ denotes meta-transaction settings, *e.g.*, party-input bindings [37].

To create a succinct and general $proof$, we use TEE to endorse the enforcement of MPT. Let $\mathcal{E}$ denote TEE. $\mathcal{E}$ is expected to receive inputs x, s, run f, deliver outputs r, s', generate $proof$, and enforce $\mathcal{P}$ throughout the process. Let $H(*)$ denote $\mathsf{hash}(*)$. When $\mathcal{E}$ successfully evaluates an MPT, $\mathcal{E}$ sends a signed transaction TX_{com} that includes a $proof \leftarrow [H_{C_{\mathcal{P}}}, H_{C_f}, H_{C_s}, H_{C_x}, H_{C_{s'}}, H_{C_r}]$. The signed $proof$ demonstrates $\mathcal{E}$'s endorsement of the state transition from s to s' caused by MPT. Thus, if $H_{C_{\mathcal{P}}}, H_{C_f}$, and H_{C_s} in TX_{com} match their previously registered commitments on-chain, the blockchain then accepts the state transition.

3.5.3 *Resisting adversary with minimal transactions (against **C3**).*

The interaction of an TEE with the environment is controlled by the ***E***. As a result, a malicious ***E*** can stop the TEE from running or present Eclipse Attacks [13] during the protocol. Malicious parties can also abort at any point during the protocol to launch a DoS attack. [16] allows parties to punish the aborted ***E*** after a certain amount of time has passed. Because the system relies on Bitcoin-specific time-delay transactions, it cannot be used on other platforms. [16] also uses an enhanced *challenge-response* subprotocol to distinguish between the malicious ***E*** dropping party inputs and malicious parties failing to submission inputs. However, all those works on defending against the foregoing attacks, such as [16, 25, 43], require both parties and the ***E*** to deposit collateral at the start of the protocol. Even if all parties and the ***E*** are honest, these works result in $O(n)$ transactions for each MPT, which is expensive and inefficient.

In this paper, we adopt a *challenge-response* subprotocol similar to [16] to identify adversary in the input submission phase. The idea behind both subprotocols is that the protocol penalise ***E*** by default unless ***E*** can show the TEE that it has publicly challenged parties on-chain but received no reply. These two subprotocols require $O(m)$ transactions when m malicious parties present. Furthermore, we design a *one-deposit-multiple-transact* method. The method requires only two constant transactions when all subjects behave honestly. Specifically, ***E*** and parties globally deposit coins as account balances to an address managed by TEE. Before evaluating an MPT, TEE only deducts MPT-specific collateral from MPT-involved party accounts by a TX_p. If the MPT succeeds, the TEE refunds the frozen MPT-specific collateral to parties via TX_{com}. As a result, each party depositing coins once can join (sequentially or concurrently) numerous MPT, as long as the total amount of deducted MPT-specific collateral does not exceed the amount of coins deposited by the party. Finally, because *challenge-response* subprotocols are rarely executed due to the high financial cost of adversary, we achieve $O(1)$ transactions per MPT in normal cases.

4 CLOAK PROTOCOL

In this section, we illustrate how Cloak protocol π_{Cloak} enforces MPT in detail. Given a blockchain $\boldsymbol{BC}$, a party set $\bar{\boldsymbol{P}}(|\bar{P}| = n)$ participating the MPT, and an executor $\boldsymbol{E}$, Figure 2 depicts the detailed phases and messages of the Cloak protocol. Each $P_i \in \bar{\boldsymbol{P}}$ communicates with $\mathcal{E}$ by secure channels[1]. For simplicity, we only mark the ciphertext not for building secure channels, *e.g.*, the ciphertext in each transaction sent to the blockchain.

As described in Section 3, our protocol π_{Cloak} proceeds in four phases. To summarise, before sending an MPT, $\boldsymbol{E}$ and each party are required to *deposit* some coins Q in the *global setup phase.* Both subjects go through the setup phase only once. Then, three phases follow to evaluate an MPT. During a *negotiation phase*, all parties negotiate off-chain to join the MPT, and finally, $\boldsymbol{E}$ commits a settled MPT proposal with parties' input commitments and deducts their collateral on-chain. Next, an *execution phase* follows for collecting plaintext of parameters and old states from parties and executing MPT in the enclave to obtain outputs and deliver the output ciphertext to parties. Finally, the protocol enters a *distribution phase* to commit outputs and reveal the encryption key to complete the MPT. We now explain the detailed protocol phases. Protocol security parameters such as t_*, τ_* are quantified in Appendix D.

4.1 Negotiation phase

This phase uses a *nondeterministic negotiation protocol* ($\boldsymbol{Proc}_{\text{noneg}}$) to guide parties to reach a consensus on an MPT proposal and commit parameters x_i on-chain[2]. $\boldsymbol{Proc}_{\text{noneg}}$ proceeds in two stages.

1.1: One party wishing to call an MPT sends an MPT proposal $p \leftarrow (C_f, C_{\mathcal{P}}, q)$ to $\mathcal{E}$, where q refers to required collateral for punishing adversary. Then, $\mathcal{E}$ generates a random identifier for proposal p, id_p, and broadcasts the signed (id_p, p) to parties $\bar{\boldsymbol{P}}$. After receiving (id_p, p), each P_i interested in the MPT computes the commitment C_{x_i} of its parameter x_i and sends a signed acknowledgment $ACK_i \leftarrow (id_p, C_{x_i})$ to $\mathcal{E}$ before t_n, where t_n refers to the negotiation phase's completion time. $\mathcal{E}$ knows that P_i is interested in MPT when it receives ACK_i .

1.2: If the collected acknowledgements satisfy the settlement condition[3] specified in p, $\mathcal{E}$ creates a settled proposal p'. p' expands p by adding the addresses of parties $\bar{\boldsymbol{P}}$ and the array containing all parties' parameter commitments C_x. Then, $\mathcal{E}$ sends TX_p to the blockchain to confirm p'. Additionally, TX_p deducts q collateral from each party and $n * q$ from the executor prior to executing MPT. Finally, $\mathcal{E}$ enters the *execution phase.* Otherwise, if the settlement condition is not satisfied and the duration exceeds t_n, the negotiation of p fails, and $\mathcal{E}$ terminates the protocol.

4.2 Execution phase

This phase collects plaintext inputs from parties and evaluates the MPT to obtain outputs. It normally contains two stages. When some subjects misbehave, an additionally *challenge-response submission* stage arises.

2.1: Upon confirmation of TX_p on $\boldsymbol{BC}$, $\mathcal{E}$ reads the view of $\boldsymbol{BC}$ in order to validate the PoP [11, 13] of TX_p, *i.e.*, (PoP_p), and reads the old state commitments C_s from $\boldsymbol{BC}$. Additionally, any party P_i that is aware of its involvement in TX_p submits inputs (*i.e.*, parameters x_i and old states s_i) to $\mathcal{E}$. $\mathcal{E}$, upon receiving inputs from P_i, recomputes commitments of x_i, s_i in order to match them to their corresponding commitments C_{x_i}, C_{s_i} on $\boldsymbol{BC}$. If all inputs from all parties are collected and matched, $\mathcal{E}$ proceeds to ***2.2***. Otherwise, if $\mathcal{E}$ discovers that certain parties' inputs conflict with their on-chain commitments or that some parties' inputs are not received before t_e, $\mathcal{E}$ flags these parties as potentially misbehaving and returns $\bar{\boldsymbol{P}}_M$ to $\boldsymbol{E}$. Then, $\boldsymbol{E}$ invokes $\mathcal{E}.challenge$ to send TX_{cha}. TX_{cha} publicly challenges all parties in $\bar{\boldsymbol{P}}_M$ on-chain. Following that, $\mathcal{E}$ proceeds to the ***challenge-response submission*** stage.

challenge-response submission: After TX_{cha} is confirmed on-chain, parties belong to $\bar{\boldsymbol{P}}_M$ but are honest send a TX_{res} to publish ciphertext of their inputs x_i, s_i. All published TX_{res} must be confirmed prior to the block $h_{cp} + \tau_{sub}$. After all published TX_{res} are confirmed, $\mathcal{E}$ reads published TX_{res} and verifies PoP_{res} (recall that PoP is for proving a message has been confirmed on the blockchain). If $\mathcal{E}$ successfully reads matching inputs from TX^i_{res}, $\mathcal{E}$ deletes P_i from $\bar{\boldsymbol{P}}_M$. Otherwise, if PoP_{res} shows that no TX^i_{res} of $P_i \in \bar{\boldsymbol{P}}_M$ has been published or the inputs in TX^i_{res} are still mismatched, $\mathcal{E}$ maintains P_i in $\bar{\boldsymbol{P}}_M$. Subsequently, if $\bar{\boldsymbol{P}}_M$ becomes empty after the challenge-response stage, indicating that all inputs have been collected, $\mathcal{E}$ proceeds to ***2.2***. Conversely, if $\bar{\boldsymbol{P}}_M$ remains nonempty, indicating that the misbehavior of remaining parties have been proven, $\mathcal{E}$ flags the remain parties as misbehaving parties $\bar{\boldsymbol{P}}'_M$. After that, $\mathcal{E}$ sends TX_{pns}. TX_{pns} refunds all parties' deducted collateral only to honest parties on average and terminates the MPT with `ABORT`.

2.2: If all parties' inputs are correctly collected, $\mathcal{E}$ replaces the state of the EVM inside the enclave with old state s, then runs $f(x)$ based on s to obtain MPT outputs, *i.e.*, return values r and new state s'. Following that, $\mathcal{E}$ generates a one-time symmetric key k and use it to encrypts r and s' to obtain their ciphertext. Finally, $\mathcal{E}$ delivers the ciphertext $Enc_k(s'_i, r_i)$ to each P_i correspondingly.

4.3 Distribution phase

Briefly, this phase aims to publish the encryption key k and commit the outputs to the blockchain after ensuring that all parties have received the output ciphertext encrypted by k. It normally contains two stages. When some subjects misbehave, an additionally *challenge-response delivery* stage arises.

3.1: $\mathcal{E}$ waits for receipts the of output ciphertext from all parties. If all receipts are collected, $\mathcal{E}$ proceeds to ***3.2***. Alternatively, similar to *challenge-response submission*, if $\mathcal{E}$ discovers that certain parties' receipts are invalid or have not received some parties' receipts prior to t_d, $\mathcal{E}$ flags these parties as potentially misbehaving parties and returns $\bar{\boldsymbol{P}}_M$ to $\boldsymbol{E}$. Then, $\boldsymbol{E}$ calls $\mathcal{E}.challenge$ to send TX_{cha}. TX_{cha} challenges all parties in $\bar{\boldsymbol{P}}_M$ on-chain with their output ciphertext additionally encrypted by their own public keys. Subsequently, $\mathcal{E}$ proceeds to the ***challenge-response delivery*** stage.

challenge-response delivery: After TX_{cha} is confirmed, parties present in $\bar{\boldsymbol{P}}_M$ but are honest send TX_{res} to publish their receipts of output s'_i, r_i on $\boldsymbol{BC}$. All published TX_{res} must be confirmed prior to the block $h_{cp} + \tau_{rec}$. When all published TX_{res} have been confirmed,

---

[1] Messages sent from a party P_i to $\boldsymbol{E}$ are signed by P_i and encrypted by $pk_{\mathcal{E}}$, while messages sent from $\mathcal{E}$ to P_i are also signed by $\mathcal{E}$.

[2] The old state s_i is already committed on-chain before starting this MPT

[3] Settlement conditions of negotiation can be specified in $\mathcal{P}$, *e.g.*, requiring specific parties joining MPT or the number of parties exceeding a specific number.

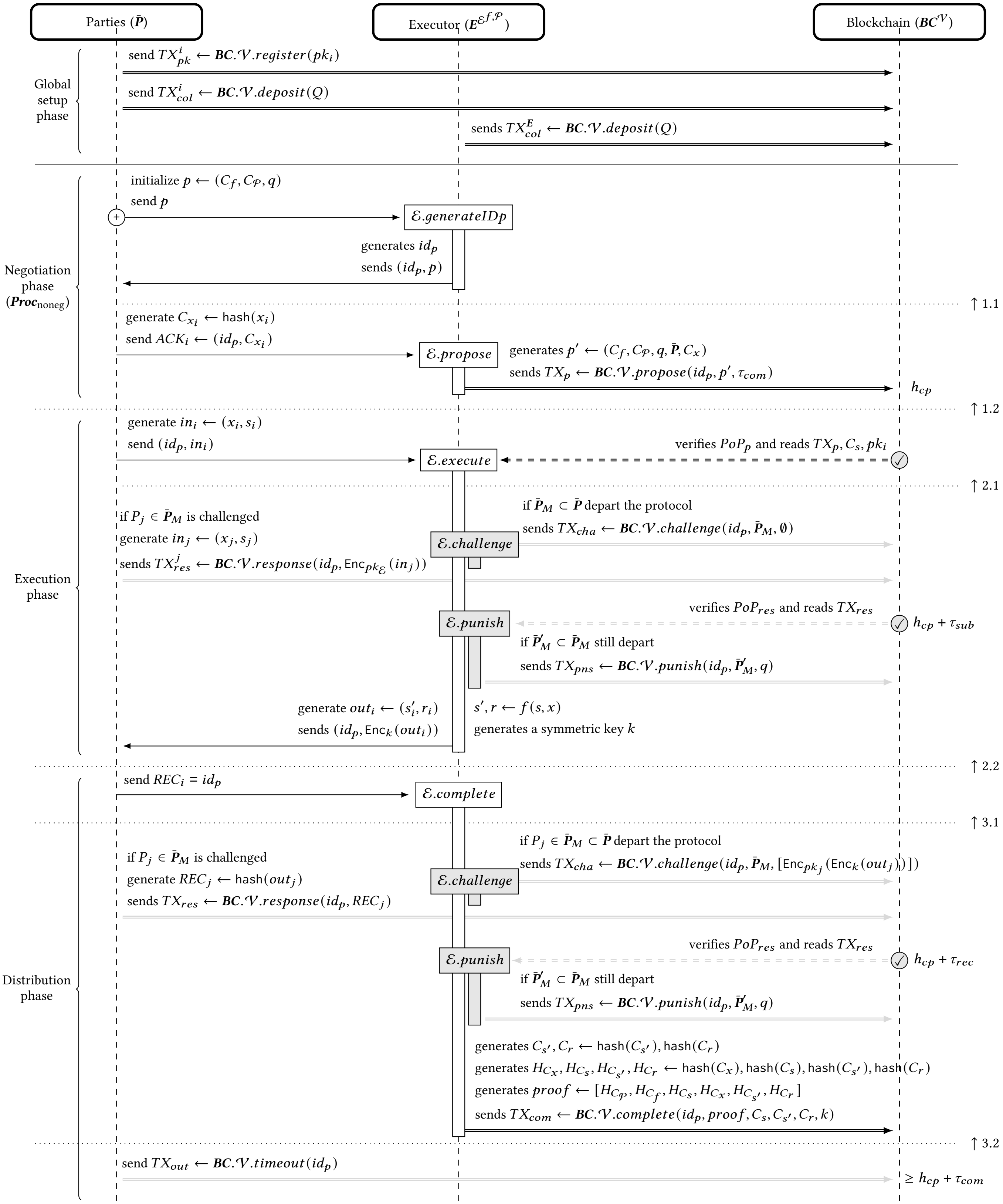

Figure 2: Detailed Cloak protocol π_{Cloak}. $\bar{P}$ refers to parties participating in an MPT. $E^{\mathcal{E}f,\mathcal{P}}$ refers to an executor holding a TEE enclave $\mathcal{E}$ with deployed $f, \mathcal{P}$. $BC^{\mathcal{V}}$ refers to the blockchain with deployed contract program $\mathcal{V}$, where $\mathcal{V}$ is an on-chain verifier that manages the life cycle of MPT and accepts output commitments by verifying the state transition proved by $proof$. Double dashed arrows refer to reading from the blockchain and double arrows refer to sending a transaction to the blockchain. Normal arrows indicate off-chain communication. $Proc_{\text{noneg}}$ refers to nondeterministic negotiation protocol. All parties P_i communicate with the executor in secure channels; thus, we omit marking ciphertext in communications between $\bar{P}$ and $\mathcal{E}$ for simplicity but explicitly mark the ciphertext in transactions published on $BC^{\mathcal{V}}$.

$\mathcal{E}$ reads the published TX_{res} and verifies its PoP_{res}. For P_i in $\bar{\boldsymbol{P}}_M$, if $\mathcal{E}$ successfully reads a party's receipt REC_i from its TX^i_{res}, $\mathcal{E}$ deletes P_i from $\bar{\boldsymbol{P}}_M$. Otherwise, if PoP_{res} shows that no TX^i_{res} has been published, $\mathcal{E}$ maintains P_i in $\bar{\boldsymbol{P}}_M$. Subsequently, if $\bar{\boldsymbol{P}}_M$ becomes empty after this challenge-response stage, indicating that all parties' receipts have been collected, $\mathcal{E}$ proceeds to ***3.2***. Conversely, if $\bar{\boldsymbol{P}}_M$ remains nonempty, proving that the remaining parties misbehave, $\mathcal{E}$ flags these misbehaving parties as $\bar{\boldsymbol{P}}'_M$. After that, $\mathcal{E}$ sends TX_{pns}. This transaction refunds all parties' deducted collateral only to honest parties on average and terminates the MPT with ABORT.

3.2: $\mathcal{E}$ publishes k and the commitments of s'_i, r_i on-chain. Specifically, $\mathcal{E}$ computes the commitments of s'_i, r_i, yielding $C_{s'_i}, C_{r_i}$. Then, $\mathcal{E}$ sends TX_{com} to the blockchain along with k and $proof$. $proof \leftarrow [H_{C_{\mathcal{P}}}, H_{C_f}, H_{C_s}, H_{C_x}, H_{C_{s'}}, H_{C_r}]$, where $H_{C_{\mathcal{P}}}$ (resp. H_{C_f}) denotes the commitment hash of $\mathcal{P}$ (resp. f) and H_{C_s} denotes $\texttt{hash}([C_{s_i}|_{1..n}])$. $proof$ in TX_{com} achieves public verifiability of MPT in the following way: The signed TX_{com} indicates that $\mathcal{E}$ endorses that by enforcing a privacy policy $\mathcal{P}$ (matching $H_{C_{\mathcal{P}}}$), it takes private x, s (matching H_{C_x}, H_{C_s}) from $\bar{\boldsymbol{P}}$, evaluates f (matching H_{C_f}), and obtains private outputs s', r (matching $H_{C_{s'}}, H_{C_r}$). Therefore, by trusting the integrity and confidentiality of $\mathcal{E}$, if $\boldsymbol{BC}$ verifies that $H_{C_{\mathcal{P}}}, H_{C_f}$ in $proof$ matches the previously registered $H_{C_{\mathcal{P}}}, H_{C_f}$, and H_{C_s} in $proof$ matches the existing state commitments, $\boldsymbol{BC}$ then accepts the state transition, updates its existing state commitments with $C_{s'}$, and signals COMPLETE of the MPT. Otherwise, if $\mathcal{E}$ neither completes (via TX_{com}) or terminates (via TX_{pns}) the MPT before the block height τ_{com}, it indicates that $\boldsymbol{E}$ misbehaves. Then, any P_i can send TX_{out} to punish $\boldsymbol{E}$ and refund their collateral with additional compensation. TX_{out} also terminates MPT with ABORT.

5 IMPLEMENTATION

For prototyping, Cloak instantiates the TEE as Intel SGX and the blockchain as Ethereum.

5.1 Contract facility

We use Ganache [40] to simulate an legacy Ethereum, *i.e.*, the $\boldsymbol{BC}$ in Cloak. To enable the MPT on existing $\boldsymbol{BC}$, we require the executor to deploy a contract program $\mathcal{V}$ to $\boldsymbol{BC}$. As is shown in Algorithm 1, $\mathcal{V}$ is constructed by the config of $\mathcal{E}$, *e.g.*, $pk_{\mathcal{E}}$ and $adr_{\mathcal{E}}$, so that parties can attest the integrity of $\mathcal{E}$ by its IAS report and then build secure channels with $\mathcal{E}$ by $pk_{\mathcal{E}}$. Moreover, $\mathcal{V}$ provides several functions to manage life cycles of MPT.

5.2 Enclave facility

The enclave program was implemented as an App of CCF in C++ for reusing CCF's key generation and synchronization functionality. Specifically, CCF is a TEE-based consortium framework. It is easy for us constantly synchronizing a common key pair among all TEE devices in a CCF network. Therefore, we can easily add new TEE device to improve the availability of the executor, without breaking the assumption and requirement of our protocol. Moreover, both CCF and our enclave program are developed based on Openenclave [18]. Openenclave provides TEE-agnostic API for developing enclave programs, making our implementation easier to adapt to different TEE platforms.

Our enclave program is presented in Algorithm 2. When the executor $\boldsymbol{E}$ instantiates a TEE enclave using the enclave program,

Algorithm 1: Contract program ($\mathcal{V}$)

```
// This contract is constructed by the config of the executor
   and the enclave. $pk_{\mathcal{E}}, adr_{\mathcal{E}}$ is the public key and address of
   the enclave $\mathcal{E}$, where $pk_{\mathcal{E}}$ is used for parties building
   secure channels with the $\mathcal{E}$ and $adr_{\mathcal{E}}$ manages coins
   deposited by parties and the executor. $adr_{\mathbf{E}}$ is the address
   of the executor. For simplicity, we omit access control
   logic here, but remark it each function.
Function constructor($pk_{\mathcal{E}}, adr_{\mathcal{E}}, adr_{\boldsymbol{E}}$)
    $pk_{\mathcal{E}}, adr_{\mathcal{E}} \leftarrow pk_{\mathcal{E}}, adr_{\mathcal{E}}$ // for secure channel
    $adr_{\boldsymbol{E}} \leftarrow adr_{\boldsymbol{E}}$
    $Proposals \leftarrow []$
Function register($pk_i$)
    // called by $TX_{pk}$
    $PartyPKs[msg.sender] \leftarrow pk_i$
Function deposit($Q$)
    // called by $TX_{col}$ from parties and the executor
    $Coins[msg.sender] \leftarrow Coins[msg.sender] + Q$
Function propose($id_p, p', \tau_{com}$)
    // called by $TX_p$ from $\mathcal{E}$ to settle an MPT proposal
    require($Proposals[id_p] = \emptyset$)
    $Proposals[id_p].\{C_f, C_{\mathcal{P}}, \bar{\boldsymbol{P}}, C_x, q, \tau_{com}, h_{cp}\} \leftarrow$
      $p'.\{C_f, C_{\mathcal{P}}, \bar{\boldsymbol{P}}, C_x, q\}, \tau_{com}, \boldsymbol{BC}$.getHeight()
    // deduct collaterals before execution
    $Coins[adr_{\boldsymbol{E}}] \leftarrow Coins[adr_{\boldsymbol{E}}] - |\bar{\boldsymbol{P}}| * Proposals[id_p].q$
    require($Coins[adr_{\boldsymbol{E}}] \geq 0$)
    for $P_i \in \bar{\boldsymbol{P}}$ do
        $Coins[P_i] \leftarrow Coins[P_i] - Proposals[id_p].q$
        require($Coins[P_i] \geq 0$)
    $Proposals[id_p].st \leftarrow$ SETTLE
Function challenge($id_p, \bar{\boldsymbol{P}}_M, data_{cha}$)
    // called by $TX_{cha}$ from $\mathcal{E}$ to challenge specific parties
    require($Proposals[id_p].st =$ SETTLE)
    for $P_i \in \bar{\boldsymbol{P}}_M$ do
        $Proposals[id_p].\bar{\boldsymbol{P}}[P_i].challenge \leftarrow data_{cha}$
Function response($id_p, data_{res}$)
    // called by $TX_{res}$ from parties being challenged
    require($Proposals[id_p].st =$ SETTLE)
    $Proposals[id_p].\bar{\boldsymbol{P}}[msg.sender].response \leftarrow data_{res}$
Function punish($id_p, \bar{\boldsymbol{P}}'_M$)
    // called by $TX_{pus}$ from $\mathcal{E}$
    require($Proposals[id_p].st =$ SETTLE)
    $refunds = Proposals[id_p].q * (1 + \frac{|\bar{\boldsymbol{P}}'_M|}{|Proposals[id_p].\bar{\boldsymbol{P}} - \bar{\boldsymbol{P}}'_M| + 1})$
    for $P_i \in (Proposals[id_p].\bar{\boldsymbol{P}} - \bar{\boldsymbol{P}}'_M)$ do
        $Coins[P_i] \leftarrow Coins[P_i] + refunds$
    $Coins[adr_{\boldsymbol{E}}] \leftarrow Coins[adr_{\boldsymbol{E}}] + refunds$
    $Proposals[id_p].st \leftarrow$ ABORT
Function complete($id_p, proof, C_s, C_{s'}, C_r, k$)
    // called by $TX_{com}$ from $\mathcal{E}$ to punish misbehaved parties
    require($Proposals[id_p].st =$ SETTLE)
    if verify($proof, C_f, C_{\mathcal{P}}, Proposals[id_p].C_x, C_s, C_{s'}, C_r$) then
        setNewState($C_{s'}$)
        $Proposals[id_p].\{C_r\} \leftarrow \{C_r\}$
        $refunds = Proposals[id_p].q$
        for $P_i \in Proposals[id_p].\bar{\boldsymbol{P}}$ do
            $Coins[P_i] \leftarrow Coins[P_i] + refunds$
        $Coins[adr_{\boldsymbol{E}}] \leftarrow Coins[adr_{\boldsymbol{E}}] + refunds$
        $Proposals[id_p].st \leftarrow$ COMPLETE
Function timeout($id_p$)
    // called by $TX_{out}$ from parties
    require($\boldsymbol{BC}$.getHeight() ¡ $Proposals[id_p].h_{cp} + \tau_{com}$)
    require($Proposals[id_p].st =$ SETTLE)
    $refunds = Proposals[id_p].q * 2$
    for $P_i \in Proposals[id_p].\bar{\boldsymbol{P}}$ do
        $Coins[P_i] \leftarrow Coins[P_i] + refunds$
    $Proposals[id_p].st \leftarrow$ ABORT
```

the enclave becomes $\mathcal{E}$. In more detail, $\boldsymbol{E}$ set up $\mathcal{E}$ with a secure parameter κ and a checkpoint b_{cp} of blockchain, then, publishes its $\mathcal{E}$'s IAS Attestation Report REP_{ias}.

To verify/sign transactions as well as building secure channels with parties inside enclave, we port OpenSSL and secp256k1 [14] to support needed ECDSA. To allow flexible specification of MPT, *e.g.*, specifying identities who are able/required to join an MPT, we implement an *policy engine* inside enclave to interpret and enforce JSON-based privacy policy $\mathcal{P}$ of MPT. The target function of MPTs are expressed in Solidity 0.8.10 [19] and we port EVM [20] to CCF [33].

5.3 Optimization

Instead of reading/writing the whole state of the contract which is adopted in [13]. Cloak synchronize states with blockchain as need. We pre-specify the states I/O of MPT in its privacy policy to inform the $\mathcal{E}$ what old states are needed for evaluating the MPT and what state would be mutated. More details of the policy refer to [32]. Admittedly, reading/writing states according to pre-defiend policy requires that all possible states I/O of an MPT should be statically recognized before evaluation, so that disallow inputs-depends states I/O logic. We stress that the problem can be totally solved by hooking EVM instructions `sload` and `sstore` like [38]. We leave this for our future work.

6 SECURITY ANALYSIS

We informally claim that Cloak protocol satisfies five properties: ***correctness***, ***confidentiality***, ***public verifiability***, ***executor balance security***, and ***financial fairness*** in the following theorem. In Appendix C.3 and D, we formally define the security properties, state the theorem, and proves our protocol.

Theorem 1 (Informal statement). *The protocol π_{CLOAK} satisfies* ***correctness, confidentiality, public verifiability, executor balance security,*** *and* ***financial fairness***

Particularly, as Cloak claims to resist a Byzantine adversary, it includes resisting the single-point failure and rollback attack. For the former, Cloak always punishes the executor presenting single-point failure, thinking an honest executor can improve its availability by various schemes, *e.g.*, multiple TEEs with consensus and synchronized keys (as we implemented in Section 5). For the later, first, $\mathcal{V}$ always rejects MPT's outputs from unmatched states, *e.g.*, rollbacked states. Second, the TEE is initiated with a checkpoint block b_{cp}, and it always validates the PoP and updates the b_{cp} when read data (*e.g.*, contract states and parameter commitments) on blockchain. Thanks to PoP which ensures that the data to read have been finalized by the consensus, TEE ensures that it always read on-chain data from a monotonically increasing main chain.

7 EVALUATION

Methodology and setup. To evaluate the effectiveness of Cloak, we propose 3 research questions.

- **Q1**: Does Cloak fit real-world needs of publicly verifiable MPT?
- **Q2**: What's the cost of the deployment and global setup for enabling publicly verifiable MPT on a blockchain by using Cloak?
- **Q3**: What's the cost of evaluating a MPT by using Cloak?

Algorithm 2: Enclave program ($\mathcal{E}$)

```
// $\mathcal{E}$ is set up with a secure parameter $\kappa$ and a checkpoint $b_{cp}$
   of blockchain. $\kappa$ is used for generating an asymmetric key
   pair $(pk_{\mathcal{E}}, sk_{\mathcal{E}})$ for building secure channels, an blockchain
   account $(adr_{\mathcal{E}}, key_{\mathcal{E}})$ for managing coins and sending
   transactions on-chain, and block intervals $\tau_{com}$ for judging
   MPT timeout on-chain. For simplicity, we omit the logic of
   setting up $f, \mathcal{P}, adr_{\mathcal{V}}$
Function setup($\kappa, b_{cp}$)
    $pk_{\mathcal{E}}, sk_{\mathcal{E}}, adr_{\mathcal{E}}, key_{\mathcal{E}} \leftarrow Gen(1^{\kappa})$
    $t_n, t_e, t_d, \tau_{sub}, \tau_{rec}, \tau_{com} \leftarrow Gen(1^{\kappa})$
    return $pk_{\mathcal{E}}, adr_{\mathcal{E}}$
Function generateIDp($p$)
    $id_p \leftarrow Gen(1^{\kappa})$
    $t_{cp} \leftarrow$ currTime()
    $step \leftarrow$ propose
    return $(id_p, p)$
Function propose($ACK$)
    if $step <$ propose or SatiPolicy($ACK, \mathcal{P}$) $< 1$ or
      currTime() $\geq t_{cp} + t_n$ then abort
    $p' \leftarrow (p.C_f, p.C_{\mathcal{P}}, p.q, \bar{\mathbf{P}}, ACK.C_x)$
    $step \leftarrow$ execute
    return $TX_p(id_p, p', \tau_{com})$
Function execute($in, TX_p, PoP_p$)
    if $step <$ execute or veriPoP($b_{cp}, TX_p, PoP_p$) $< 1$ then abort
    $\bar{\boldsymbol{P}}_M \leftarrow \emptyset$
    for $P_i$ in $\bar{\boldsymbol{P}}$
        if $in.\{x_i, s_i\} = \emptyset$ or hash($x_i$) $< PoP_p.TX_p.C_{x_i}$
            or hash($s_i$) $< PoP_p.C_{s_i}$ then
            $\bar{\boldsymbol{P}}_M \leftarrow \bar{\boldsymbol{P}}_M \cup \{P_i\}$
    if $|\bar{\boldsymbol{P}}_M|$ ¡ 0 then
        return $(id_p, \bar{\boldsymbol{P}}_M)$
    $out \leftarrow s', r \leftarrow f(s, x)$ // evaluates $f(x)$ on old states $s$
    $b_{cp} \leftarrow PoP_p$.getLastBlock()
    $k \leftarrow Gen(1^{\kappa})$ // generates a symmetric key
    $step \leftarrow$ complete
    return $[(id_p, \text{Enc}_k(out_i))]$
Function challenge($id_p, \bar{\boldsymbol{P}}_M$)
    if $|\bar{\boldsymbol{P}}_M|$ Ÿ 0 then abort
    if $step =$ execute and currTime() $\leq t_{cp} + t_e$ then
        return $TX_{cha}(id_p, \bar{\boldsymbol{P}}_M, \emptyset)$
    elif $step =$ complete and currTime() $\leq t_{cp} + t_d$ then
        return $TX_{cha}(id_p, \bar{\boldsymbol{P}}_M, [\text{Enc}_{pk_i}(\text{Enc}_k(out_i))])$
    else abort
Function punish($TX_{cha}, TX_{res}, PoP_{res}$)
    $\bar{\boldsymbol{P}}'_M \leftarrow \emptyset$
    for $P_i \in TX_{cha}.\bar{\boldsymbol{P}}_M$ do
        if $step =$ execute then
            if veriPoP($b_{cp}, TX^i_{res}, PoP_{res}, \tau_{sub}$) $< 1$
                or hash($TX^i_{res}.x_i$) $< PoP_{res}.TX_p.C_{x_i}$
                or hash($TX^i_{res}.s_i$)) $< C_{s_i}$ then $\bar{\boldsymbol{P}}'_M \leftarrow \bar{\boldsymbol{P}}'_M \cup \{P_i\}$
        elif $step =$ complete then
            if veriPoP($b_{cp}, TX^i_{res}, PoP_{res}, \tau_{rec}$) $< 1$
                or $TX_{res}.REC_i <$ hash($out_i$) then $\bar{\boldsymbol{P}}'_M \leftarrow \bar{\boldsymbol{P}}'_M \cup \{P_i\}$
        else abort
    if $|\bar{\boldsymbol{P}}'_M|$ ¡ 0 then
        $step \leftarrow \perp$
        return $TX_{pns}(id_p, \bar{\boldsymbol{P}}'_M)$
Function complete($REC$)
    if $step <$ complete or missed some $REC_i$ then abort
    $H_{C_{\mathcal{P}}}, H_{C_f} \leftarrow$ hash(hash($\mathcal{P}$)), hash(hash($f$))
    $C_{s'_i}, C_{r_i} \leftarrow$ hash($s'_i$), hash($r_i$)
    $H_{C_x}, H_{C_s}, H_{C_{s'}}, H_{C_r} \leftarrow$ hash($C_x$), hash($C_s$), hash($C_{s'}$), hash($C_r$)
    $proof \leftarrow [H_{C_{\mathcal{P}}}, H_{C_f}, H_{C_x}, H_{C_s}, H_{C_{s'}}, H_{C_r}]$
    $step \leftarrow \perp$
    return $TX_{com}(id_p, proof, C_s, C_{s'}, C_r, k)$
```

To answer **Q1**, we apply Cloak to 5 contracts with 10 different MPTs. As is shown in Table 2, these contracts vary from both LOC, scenarios, and number of participants. Specifically, their business involves energy, education, and blockchain infrastructure. The involved parties of these 10 MPT spans from 2 to 11.

Table 2: Contracts with MPT. "#MPT" denotes the number of MPT and "Scenarios" denotes typical business.

Name	#MPT	#LOC	Scenarios
`SupplyChain`	**1**	39	An example contract allowing suppliers to negotiate and privacy-preservedly bids off-chain, and commit the evaluation with their new balances on-chain
`Scores`	**1**	95	An example contract allowing students to join and get mean scores off-chain and commit the evaluation on-chain
`ERC20Token`	**3**	55	An example contract allowing accounts to pair and transfer without revealing balances off-chain, and commit the evaluation with new balances on-chain.
`YunDou`	**3**	105	A real-world token contract supporting co-managed accounts, in which a suffi--cient number of managers self-selectly vote to transfer tokens without revealing the votes.
`Oracle`	**2**	60	A real-world Oracle contract that allows parties to negotiate to join then jointly and verifiably generate random numbers

To answer **Q2** and **Q3**, We record the cost of time and gas for evaluating each MPT. We also compare Cloak with Fastkitten, the SOTA most related to our work. To obtain a comparative experimental setup, considering that Fastkitten is specific for Bitcoin, we implement its on-chain commitment logic as a Solidity smart contract and commit the party inputs of Fastkitten to achieve the comparable public verifiability with Cloak.

The experiment is based on Ubuntu 18.04 with 32G memory and 2.2GHz Intel(R) Xeon(R) Silver 4114 CPU. Although the gas cost of a specific transaction is deterministic, it also varies from transaction arguments. Therefore, we send each MPT 3 times with different arguments to get the average result.

7.1 Deployment and Setup Cost

To answer **Q2**, we discuss the gas cost of deploying the contract program $\mathcal{V}$. The result is shown in Figure 3.

Gas cost of deployment. In global initialization phase, Cloak costs 4.5M gas to deploy the contract program $\mathcal{V}$ to enable MPT in existing blockchains. This cost is only paid by Cloak service provider for once, thus is mostly irrelevant.

Gas cost of global setup. Each party pays 12.7k gas to *register* (`reg.`) its public key and 4.2k gas to *deposit* (`dep.`) its coins. Therefore, this global once paid gas cost is acceptable.

7.2 Transaction Cost

Gas cost of evaluating MPTs. The right part of Figure 3 shows the transaction costs of all 10 MPTs in 5 contracts. In general, Cloak reduces gas by 32.4% compare to Fastkitten, which requires $n + 1$ transactions. Specifically, for 4 MPTs with only 2 parties, Cloak cost 0.79-0.82X gas to Fastkitten. However, when the number of parties increases to 10 and 11, the cost of Cloak significantly decreases to 0.45-0.46X. Overall, we conclude that Cloak evaluates MPT in not only a securer adversary model but also a lower cost.

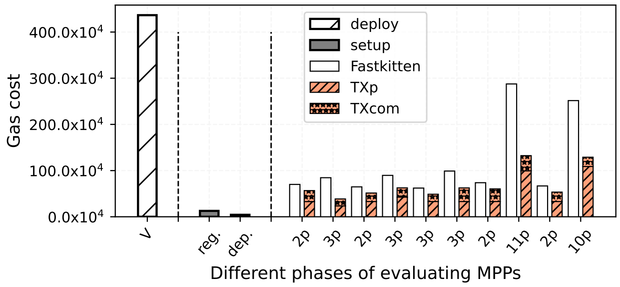

Figure 3: The gas cost of Cloak.

Off-chain latency of evaluating MPTs. The end-to-end time to evaluate an MPT is 10 minutes. However, only about 1s is spent on evaluation, with the remainder spent on waiting for the blockchain to generate an PoP of TX_p. Precisely, the negotiation phase takes 0.1-0.39s while the execution and distribution phases take 0.26-0.71s. While verifying the PoP_p takes 4-6s, the enclave have to wait 10 minutes for the confirmation of TX_p and the generation of PoP_p. We note that the time cost of PoP is common to that of other current TEE-Blockchain systems [11, 13, 16] and is acceptable in permissionless blockchains. More importantly, in widespread quorum-based consensus, the time cost of generating PoP can be reduced to milliseconds [3, 27, 33]. Therefore our protocol is ready for use in real-world applications.

8 CONCLUSION

In this paper, we developed a novel framework, Cloak, to enable confidential smart contracts with MPT on existing blockchains. To the best of our knowledge, Cloak advances in these aspects. Specifically, Cloak is the first to support parties to securely negotiate MPT proposals off-chain without knowing or communicating with others during the negotiation phase. Cloak is the first to achieve public verifiability of an MPT while considering both on-chain and off-chain inputs/outputs. Moreover, Cloak achieves financial fairness under a Byzantine adversary model. Finally, with all the above properties, Cloak requires only 2 transactions, far superior to previous work that allows nondeterministic negotiation and financial fairness but requires $O(n)$ transactions. During our evaluation of Cloak in both examples and real-world smart contracts, Cloak reduces the gas cost by 32.4%. In conclusion, Cloak achieves low-cost and secure MPT, thereby paving the way for the publicly verifiable and reusable off-chain MPCs.

ACKNOWLEDGMENTS

This work was supported in part by the National Key R&D Program of China (No. 2021YFB2700601); in part by the Finance Science and Technology Project of Hainan Province (No. ZDKJ2020009); in part by the National Natural Science Foundation of China (No. 62163011); in part by the Research Startup Fund of Hainan University under Grant KYQD(ZR)-21071.

REFERENCES

[1] Martín Abadi, Mihai Budiu, Ulfar Erlingsson, and Jay Ligatti. 2009. Control-flow integrity principles, implementations, and applications. *ACM Transactions on Information and System Security (TISSEC)* 13, 1 (2009), 1–40.

[2] AMD. 2020. AMD SEV-SNP: Strengthening VM Isolation with Integrity Protection and More. https://www.amd.com/system/files/TechDocs/SEV-SNP-strengthening-vm-isolation-with-integrity-protection-and-more.pdf.

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

[4] Foteini Baldimtsi, Aggelos Kiayias, Thomas Zacharias, and Bingsheng Zhang. 2020. Crowd Verifiable Zero-Knowledge and End-to-End Verifiable Multiparty Computation. In *Advances in Cryptology – ASIACRYPT 2020: 26th International Conference on the Theory and Application of Cryptology and Information Security, Daejeon, South Korea, December 7–11, 2020, Proceedings, Part III* (Daejeon, Korea (Republic of)). Springer-Verlag, Berlin, Heidelberg, 717–748. https://doi.org/10.1007/978-3-030-64840-4_24

[5] Carsten Baum, Ivan Damgård, and Claudio Orlandi. 2014. Publicly Auditable Secure Multi-Party Computation. In *Security and Cryptography for Networks*, Michel Abdalla and Roberto De Prisco (Eds.). Springer International Publishing, Cham, 175–196.

[6] Andrea Biondo, Mauro Conti, Lucas Davi, Tommaso Frassetto, and Ahmad-Reza Sadeghi. 2018. The Guard's Dilemma: Efficient Code-Reuse Attacks Against Intel SGX. In *27th {USENIX} Security Symposium ({USENIX} Security 18)*. USENIX Association, Baltimore, MD, 1213–1227. https://www.usenix.org/conference/usenixsecurity18/presentation/biondo

[7] Sean Bowe, Alessandro Chiesa, Matthew Green, Ian Miers, Pratyush Mishra, and Howard Wu. 2020. ZEXE: Enabling Decentralized Private Computation. *2020 IEEE Symposium on Security and Privacy* (2020). https://doi.org/10.1109/sp40000.2020.00050

[8] Ferdinand Brasser, Urs Müller, Alexandra Dmitrienko, Kari Kostiainen, Srdjan Capkun, and Ahmad-Reza Sadeghi. 2017. Software Grand Exposure: SGX Cache Attacks Are Practical. In *11th USENIX Workshop on Offensive Technologies (WOOT 17)*. USENIX Association, Vancouver, BC. https://www.usenix.org/conference/woot17/workshop-program/presentation/brasser

[9] Jo Van Bulck, Marina Minkin, Ofir Weisse, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Thomas F. Wenisch, Yuval Yarom, and Raoul Strackx. 2018. Foreshadow: Extracting the Keys to the Intel SGX Kingdom with Transient Out-of-Order Execution. In *27th {USENIX} Security Symposium ({USENIX} Security 18)*. USENIX Association, Baltimore, MD, 991–1008. https://www.usenix.org/conference/usenixsecurity18/presentation/bulck

[10] Nathan Burow, Scott A Carr, Joseph Nash, Per Larsen, Michael Franz, Stefan Brunthaler, and Mathias Payer. 2017. Control-flow integrity: Precision, security, and performance. *ACM Computing Surveys (CSUR)* 50, 1 (2017), 1–33.

[11] Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, Jonathan Katz, Iddo Bentov, Yan Ji, Fan Zhang, Lorenz Breidenbach, Philip Daian, and Ari Juels. 2019. Tesseract: Real-Time Cryptocurrency Exchange Using Trusted Hardware *(Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security)*. 1521–1538. https://doi.org/10.1145/3319535.3363221

[12] BNB Chain. 2022. Multi-Party Threshold Signature Scheme. https://github.com/bnb-chain/tss-lib.

[13] Raymond Cheng, Fan Zhang, Jernej Kos, Warren He, Nicholas Hynes, Noah Johnson, Ari Juels, Andrew Miller, and Dawn Song. 2019. Ekiden: A Platform for Confidentiality-Preserving, Trustworthy, and Performant Smart Contracts. *2019 IEEE European Symposium on Security and Privacy (EuroS&P)* 00 (2019), 185–200. https://doi.org/10.1109/eurosp.2019.00023 arXiv:1804.05141

[14] Bitcoin Core. 2022. libsecp256k1. https://github.com/bitcoin-core/secp256k1.

[15] Victor Costan, Ilia Lebedev, and Srinivas Devadas. 2016. Sanctum: Minimal hardware extensions for strong software isolation. In *25th {USENIX} Security Symposium ({USENIX} Security 16)*. 857–874.

[16] Poulami Das, Lisa Eckey, Tommaso Frassetto, David Gens, Kristina Hostáková, Patrick Jauernig, Sebastian Faust, and Ahmad-Reza Sadeghi. 2019. FastKitten: Practical Smart Contracts on Bitcoin. In *28th {USENIX} Security Symposium ({USENIX} Security 19)*. USENIX Association, Santa Clara, CA, 801–818. https://www.usenix.org/conference/usenixsecurity19/presentation/das

[17] Lucas Vincenzo Davi, Alexandra Dmitrienko, Stefan Nürnberger, and Ahmad-Reza Sadeghi. 2013. Gadge me if you can: secure and efficient ad-hoc instruction-level randomization for x86 and ARM. In *Proceedings of the 8th ACM SIGSAC symposium on Information, computer and communications security*. 299–310.

[18] Open Enclave. 2021. Open Enclave SDK. https://github.com/openenclave/openenclave. https://github.com/openenclave/openenclave

[19] Ethereum. 2021. Solc 0.8.10. https://github.com/ethereum/solidity/releases/tag/v0.8.10. https://github.com/ethereum/solidity/releases/tag/v0.8.10

[20] Ethereum Foundation. 2020. Ethereum Virtual Machine. https://ethereum.org/en/developers/docs/evm/

[21] Daniel Gruss, Julian Lettner, Felix Schuster, Olya Ohrimenko, Istvan Haller, and Manuel Costa. 2017. Strong and efficient cache side-channel protection using hardware transactional memory. In *26th {USENIX} Security Symposium ({USENIX} Security 17)*. 217–233.

[22] Intel. 2021. Resources and Response to Side Channel L1 Terminal Fault. https://www.intel.com/content/www/us/en/architecture-and-technology/l1tf.html?wapkw=l1tf

[23] Yuekai Jia, Shuang Liu, Wenhao Wang, Yu Chen, Zhengde Zhai, Shoumeng Yan, and Zhengyu He. 2022. HyperEnclave: An Open and Cross-platform Trusted Execution Environment. In *2022 USENIX Annual Technical Conference (USENIX ATC 22)*. USENIX Association, Carlsbad, CA, 437–454. https://www.usenix.org/conference/atc22/presentation/jia-yuekai

[24] S. Kanjalkar, Y. Zhang, S. Gandlur, and A. Miller. 2021. Publicly Auditable MPC-as-a-Service with succinct verification and universal setup. In *2021 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*. IEEE Computer Society, Los Alamitos, CA, USA, 386–411. https://doi.org/10.1109/EuroSPW54576.2021.00048

[25] Ahmed Kosba, Andrew Miller, Elaine Shi, Zikai Wen, and Charalampos Papamanthou. 2016. Hawk: The Blockchain Model of Cryptography and Privacy-Preserving Smart Contracts. *2016 IEEE Symposium on Security and Privacy (SP)* (2016), 839–858. https://doi.org/10.1109/sp.2016.55

[26] Jinghui Liao, Fengwei Zhang, Wenhai Sun, and Weisong Shi. 2022. Speedster: An Efficient Multi-party State Channel via Enclaves. In *Proceedings of the 2022 ACM on Asia Conference on Computer and Communications Security*. 637–651.

[27] David Maier, Rachel Pottinger, AnHai Doan, Wang-Chiew Tan, Abdussalam Alawini, Hung Q Ngo, Ying Yan, Changzheng Wei, Xuepeng Guo, Xuming Lu, Xiaofu Zheng, Qi Liu, Chenhui Zhou, Xuyang Song, Boran Zhao, Hui Zhang, and Guofei Jiang. 2020. Confidentiality Support over Financial Grade Consortium Blockchain. 2227–2240. https://doi.org/10.1145/3318464.3386127

[28] Satoshi Nakamoto. 2008. Bitcoin: A peer-to-peer electronic cash system. *Decentralized Business Review* (2008), 21260.

[29] Job Noorman, Pieter Agten, Wilfried Daniels, Raoul Strackx, Anthony Van Herrewege, Christophe Huygens, Bart Preneel, Ingrid Verbauwhede, and Frank Piessens. 2013. Sancus: Low-cost trustworthy extensible networked devices with a zero-software trusted computing base. In *22nd {USENIX} Security Symposium ({USENIX} Security 13)*. 479–498.

[30] Alex Ozdemir and Dan Boneh. 2022. Experimenting with Collaborative zk-SNARKs: Zero-Knowledge Proofs for Distributed Secrets. In *31st {USENIX} Security Symposium ({USENIX} Security 22)*. USENIX Association, Boston, MA, 4291–4308. https://www.usenix.org/conference/usenixsecurity22/presentation/ozdemir

[31] Indrajit Ray, Ninghui Li, Christopher Kruegel, Ranjit Kumaresan, Tal Moran, and Iddo Bentov. 2015. How to Use Bitcoin to Play Decentralized Poker. *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security* (2015). https://doi.org/10.1145/2810103.2813712

[32] Qian Ren, Han Liu, Yue Li, and Hong Lei. 2021. Demo: Cloak: A Framework For Development of Confidential Blockchain Smart Contracts. In *2021 IEEE 41st International Conference on Distributed Computing Systems (ICDCS)*. 1102–1105. https://doi.org/10.1109/ICDCS51616.2021.00111

[33] Mark Russinovich, Edward Ashton, Christine Avanessians, Miguel Castro, Amaury Chamayou, Sylvan Clebsch, and et al. 2019. *CCF: A Framework for Building Confidential Verifiable Replicated Services*. Technical Report. Microsoft Research and Microsoft Azure.

[34] Berry Schoenmakers and Meilof Veeningen. 2015. Universally Verifiable Multiparty Computation from Threshold Homomorphic Cryptosystems. In *Applied Cryptography and Network Security*, Tal Malkin, Vladimir Kolesnikov, Allison Bishop Lewko, and Michalis Polychronakis (Eds.). Springer International Publishing, Cham, 3–22.

[35] Jaebaek Seo, Byoungyoung Lee, Seong Min Kim, Ming-Wei Shih, Insik Shin, Dongsu Han, and Taesoo Kim. 2017. SGX-Shield: Enabling Address Space Layout Randomization for SGX Programs.. In *NDSS*.

[36] Ming-Wei Shih, Sangho Lee, Taesoo Kim, and Marcus Peinado. 2017. T-SGX: Eradicating Controlled-Channel Attacks Against Enclave Programs.. In *NDSS*.

[37] Rohit Sinha. 2020. LucidiTEE: A TEE-Blockchain System for Policy-Compliant Multiparty Computation with Fairness.

[38] Second State and Oasis Labs. 2020. *Confidential Ethereum Smart Contracts*. Technical Report.

[39] Samuel Steffen, Benjamin Bichsel, Roger Baumgartner, and Martin Vechev. 2022. ZeeStar: Private Smart Contracts by Homomorphic Encryption and Zero-knowledge Proofs. In *2022 IEEE Symposium on Security and Privacy (SP)*. 179–197. https://doi.org/10.1109/SP46214.2022.9833732

[40] Truffle Suite. 2021. Ganache. https://github.com/trufflesuite/ganache. https://github.com/trufflesuite/ganache

[41] David Cerezo Sánchez. 2018. Raziel: Private and Verifiable Smart Contracts on Blockchains. *arXiv* (2018). arXiv:1807.09484

[42] Bhavani Thuraisingham, David Evans, Tal Malkin, Dongyan Xu, Arka Rai Choudhuri, Matthew Green, Abhishek Jain, Gabriel Kaptchuk, and Ian Miers. 2017. Fairness in an Unfair World: Fair Multiparty Computation from Public Bulletin

Boards *(Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security)*. 719–728. https://doi.org/10.1145/3133956.3134092

[43] Edgar Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew Myers, Shai Halevi, Ranjit Kumaresan, and Iddo Bentov. 2016. Amortizing Secure Computation with Penalties. *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security* (2016). https://doi.org/10.1145/2976749.2978424

[44] Edgar Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew Myers, Shai Halevi, Ranjit Kumaresan, Vinod Vaikuntanathan, and Prashant Nalini Vasudevan. 2016. Improvements to Secure Computation with Penalties. *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security* (2016), 406–417. https://doi.org/10.1145/2976749.2978421

[45] Gavin Wood et al. 2014. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper* (2014).

A SYMBOLS AND TERMINOLOGY

EUF-CMA	Existential Unforgeability under CMA.
HE	Homomorphic Encryption.
IND-CCA2	Indistinguishability under Adaptive CCA.
MPC	Multi-Party Computation.
MPT	Multi-Party Transaction.
PoP	Proof of Publication.
TEE	Trusted Execution Environment.
TTP	Trusted Third Party.
ZKP	Zero-Knowledge Proof.

B ASSUMPTION ANALYSIS

In this section, we mainly demonstrate why our assumption about TEE is practical and rational and discuss how to fine-tune our protocol to tolerant TEE compromisation.

B.1 TEE assumption rationality

Here we elaborate why assuming that the correctness and confidentiality of TEE is practical and rational. Specifically, we assume that $\boldsymbol{E}$ has full control over the machine and consequently can execute arbitrary code with root privileges. First, a malicious $\boldsymbol{E}$ can exploit memory-corruption vulnerabilities [6] in enclave code through the API between the host process and the enclave. We assume a common code-reuse defense such as control-flow correctness (CFI) [1, 10], or fine-grained code randomization [17] to be in place and active. Then, we consider architectural side-channel attacks, *e.g.*, based on caches [8]. These attacks can expose access patterns from SGX enclaves (and therefore our Cloak prototype). However, this prompted the community to develop several software mitigations [21, 35, 36] and new hardware-based solutions [15, 29]. A more serious Micro-architectural side-channel attacks like Foreshadow [9] can extract plaintext data and effectively undermine the attestation process Cloak relies on, leaking secrets and enabling the enclave to run a different application than agreed on by the parties; however, the vulnerability enabling Foreshadow was already patched by Intel [22]. Therefore, since all existing attacks targeting SGX are either patched, function-limited, or having accessible countermeasures, it is still practical to assume that confidentiality and integrity of attested SGX devices hold.

B.2 Tolerating TEE compromisation

While we assume that the integrity and confidentiality of $\mathcal{E}$ hold, we stress that it is easy to loose the assumption of π_{Cloak} to that (i) a factor of TEE devices, or even (ii) all TEEs in some types are compromised.

For (i), informally, we can instantiate a $\mathcal{E}$ as $3f + 1$ TEEs with an Byzantine-tolerant consensus (*e.g.*, PBFT). Meanwhile, those TEEs jointly evaluate any MPT by MPC protocols that tolerant $\leq f$ Byzantine nodes, *i.e.*, compromised TEEs here. Then, by regarding messages multi-signed by $\geq 2f + 1$ TEEs as messages from a unique $\mathcal{E}$, our assumption holds, as well as all claimed properties of π_{Cloak}.

For (ii), we have two ways, *i.e.*, adapting π_{Cloak} to another secure TEE product or integrating heterogeneous TEE products. For the first, since Cloak 's design is TEE agnostic, it is convenient to adapt a π_{Cloak} implementation from a type of compromised TEE product to another secure TEE without sacrificing any claimed properties. Specifically, while we instantiate the TEE as SGX, other Cloak-compatible and high-application-portability TEE products exist. For example, AMD SEV-SNP [2] has also supported remote VM attestation. HyperEnclave [23] presents a cross-platform attestable TEE where SGX programs can run with little/no code changes. Therefore, it is easy to adapt Cloak to another securer TEEs to maintain the security properties we claimed. For the second, we can instantiate a $\mathcal{E}$ as a group of heterogeneous TEE products, then follow a similar countermeasure of (i) to achieve (ii). For example, say 4 different TEEs, a SGX, SEV-SNP, HyperEncalve, and Keystone maintain a PBFT consensus and MPC as in handling (i). Consequently, if adversary corrupt no more than one type of TEE product, the $\mathcal{E}$ holds integrity and confidentiality and π_{Cloak} holds claimed properties.

C DEFINITIONS AND NOTATIONS

In this section, we introduce notations of both our architecture and protocol, and formally define the security properties we claimed in Section 6.

We refer $\mathcal{S}$ to a set, and denote the n-ary Cartesian power of the $\mathcal{S}$ to $\mathcal{S}^n \leftarrow \mathcal{S} \times \mathcal{S} \times \cdots \times \mathcal{S}$. For each vector s, $\mathsf{s} \in \mathcal{S}^n$, we refer the i-th coordinate of the vector s to $\mathsf{s}[i]$. Furthermore, an n-by-m matrix of elements from $\mathcal{S}$ is denoted as $\mathsf{S} \in \mathcal{S}^{n\times m}$. The element in the i-th row and j-column of S is $\mathsf{S}[i][j]$. The i-th row of S is denoted by $\mathsf{S}[i][\cdot]$ and the j-th column of S is denoted by $\mathsf{S}[\cdot][j]$.

C.1 Coins and multi-party programs

To be simple and chain-agnostic, we define a *coin domain* $\mathcal{D}_{coin}$ as a subset of non-negative rational numbers. The *deposit domain* is denoted by $\mathcal{D}_{dep} \leftarrow \mathcal{D}_{coin}\backslash\{0\}$. Every parties should agree on a MPT proposal with a deposit vector $\mathsf{q} \in \mathcal{D}^n_{dep}$, which means that every party has to make a deposit. The vector q' defines the final payout to each party of the contract. It must hold that $\sum_{i\in[n]} \mathsf{q}'[i] \leq \sum_{i\in[n]} \mathsf{q}[i]$. This restrictions guarantees that parties cannot create money by evaluating a program.

For secret input/output of parties, let $\mathcal{D}_s, \mathcal{D}_x, \mathcal{D}_{s'}, \mathcal{D}_r$ refer to the state domain, parameter domain, new state domain and return value domain respectively. These domains are application-specific and defined by the program. Then a old state vector is denoted by $s \in \mathcal{D}^n_s$ and a parameter vector is denoted by $x \in \mathcal{D}^n_x$. As both the s, x are parties' input of the protocol, we furthermore denote a input matrix as $\mathsf{in} \in \mathcal{D}^n_s \times \mathcal{D}^n_x$. Specifically, $\mathsf{in} \leftarrow [s^{\mathsf{T}}, x^{\mathsf{T}}]$, so that $\mathsf{in}[i][\cdot]$ refers to the input of the party $P_i \in \bar{\boldsymbol{P}}$, $\mathsf{in}[\cdot][0]$ refers to s, and $\mathsf{in}[\cdot][1]$ refer to x. Similarly, $s' \in \mathcal{D}^n_{s'}$ refers to an new state vector and $r \in \mathcal{D}^n_r$ refers to a a return value vector. We denote the output matrix as $\mathsf{ou} \in \mathcal{D}^n_{s'} \times \mathcal{D}^n_r$. $\mathsf{ou} \leftarrow [s'^{\mathsf{T}}, r^{\mathsf{T}}]$, where $\mathsf{ou}[i][\cdot]$ refers to the output of the party $P_i \in \bar{\boldsymbol{P}}$, $\mathsf{ou}[\cdot][0]$ refers to s' and $\mathsf{ou}[\cdot][1]$ refer to r. We model an n-party program f as a polynomial time Turing machine. The program f takes a input matrix in will output a output matrix ou. We denote a program f with specific policy $\mathcal{P}$ as $f_{\mathcal{P}}$. We stress that the policy $\mathcal{P}$ just holds meta-execution data of an MPT, *e.g.*, the mapping between party identities and inputs, thus is transparent to the execution process.

For on-chain commitments, we denote the domain of cryptographic commitments as $\mathcal{D}_{cm}$. Then we have $\mathsf{C}_s, \mathsf{C}_x, \mathsf{C}_{s'}, \mathsf{C}_r \in \mathcal{D}^n_{cm}$, where $\mathsf{C}_s, \mathsf{C}_x, \mathsf{C}_{s'}, \mathsf{C}_r$ refer to the old state commitment vector, parameter commitment vector, new state commitment vector and return value commitment vector respectively. Since only the old state commitments are persisted on-chain before an MPT, we denote $\mathsf{cm}_{\mathsf{in}} \in \mathcal{D}^n_{cm}$ as the input commitment, *i.e.*, $\mathsf{cm}_{\mathsf{in}} \leftarrow \mathsf{C}_s$.

Finally, we model an n-party program $f_{\mathcal{P}}$ as a polynomial time Turing machine. To formally state our security properties, we formally define what is the *correct evaluation of a program* in Algorithm 3. Specifically, the `eval` takes an n-party program $f_{\mathcal{P}}$, a input matrix in and a input commitment matrix $\mathsf{cm}_{\mathsf{in}}$, and a agreed deposit vector q. The output of the algorithm is the tuple $(s', r, \mathsf{C}_{s'}, \mathsf{C}_r, \mathsf{C}_x, proof, \mathsf{q}')$, which includes a new state vector s', return value vector r, new state commitment vector $\mathsf{C}_{s'}$, return value commitment vector C_r, parameter commitment vector C_x, a coin distribution vector q', and the status $st \leftarrow$ `COMPLETE` and $proof$ of the evaluation.

Algorithm 3: Evaluation function (`eval`)

```
Input: An n-party program $f_{\mathcal{P}}$, a deposit vector $\mathsf{q}$, a input matrix
       $\mathsf{in}$, a old state commitment vector $\mathsf{cm}_{\mathsf{in}}$
Output: A new state vector $s'$, return value vector $r$, new state
        commitment vector $\mathsf{C}_{s'}$, return value commitment vector
        $\mathsf{C}_r$, parameter commitment vector $\mathsf{C}_x$, a MPT $proof$, and
        a coin distribution $\mathsf{q}'$
Function eval($f_{\mathcal{P}}, \mathsf{q}, \mathsf{in}, \mathsf{cm}_{\mathsf{in}}$)
    foreach $\mathsf{in}.s$ // foreach $\mathsf{in}[\cdot][0]$
        assert $\mathsf{hash}(\mathsf{in}.s[i]) = \mathsf{cm}_{\mathsf{in}}[i]$ // $\mathsf{hash}(s[i]) = \mathsf{C}_{s[i]}$
    $s', r \leftarrow f_{\mathcal{P}}(\mathsf{in}.s, \mathsf{in}.x)$
    $\mathsf{C}_{s'[i]}, \mathsf{C}_{r[i]}, \mathsf{C}_{x[i]} \leftarrow$
      $\mathsf{hash}(s'[i]), \mathsf{hash}(r[i]), \mathsf{hash}(\mathsf{in}[i][1])$
    $proof \leftarrow [H_{\mathsf{C}_{\mathcal{P}}}, H_{C_f}, H_{C_s}, H_{C_x}, H_{\mathsf{C}_{s'}}, H_{\mathsf{C}_r}]$
    $\mathsf{q}' \leftarrow \mathsf{q}$
    return $(s', r, \mathsf{C}_{s'}, \mathsf{C}_r, \mathsf{C}_x, proof, \mathsf{q}')$
```

C.2 Protocol execution

We consider n parties $\bar{\boldsymbol{P}}$ and a unique executor $\boldsymbol{E}$ proceeds Cloak protocol π_{Cloak}. We denote the set including all parties and the executor as $\bar{\boldsymbol{P}}^+ \leftarrow \bar{\boldsymbol{P}} \cup \boldsymbol{E}$.

We assume that all parties $P_i \in \bar{\boldsymbol{P}}$ communicate with the executor $\boldsymbol{E}$ in authenticated channels. According to our adversary model in Section 3, a protocol is proceeded in presence of an strong adversary $\mathcal{A}$ who can arbitrarily corrupt subjects in $\bar{\boldsymbol{P}}^+$. On corrupted subjects, the $\mathcal{A}$ takes complete control so that $\mathcal{A}$ can learns and decide the inputs, outputs and also the internal state of the subjects. The input of the protocol execution is an n-party program $f_{\mathcal{P}}$, a specified deposit vector q, a input matrix in, and a vector of account balances $\mathsf{Q} \in \mathcal{D}^{n+1}_{dep} \subseteq \mathcal{D}^{n+1}_{coin}$, *i.e.*, the vector of coins pre-deposited to the address $pk_{\mathcal{E}}$ by all subjects in $\bar{\boldsymbol{P}}$. The account domain $\mathcal{D}^n_{dep}$ is defined such that $\forall P_i \in \bar{\boldsymbol{P}} : \mathsf{Q}[i] \geq \mathsf{q}[i]$ and $\mathsf{Q}[i+1] \geq \sum_{P_i \in \bar{\boldsymbol{P}}} \mathsf{q}[i]$. This restriction guarantees that every subject in $\bar{\boldsymbol{P}}^+$ has enough coins for joining an MPT. We define the execution of the Cloak protocol π_{Cloak} in presence of an adversary $\mathcal{A}$ as

$$\mathsf{ou}, \mathsf{cm}_{\mathsf{ou}}, proof, st, \mathsf{Q}' \leftarrow REAL_{\pi,\mathcal{A}}(\mathsf{Q}, f_{\mathcal{P}}, \mathsf{q}, \mathsf{in}, \mathsf{cm}_{\mathsf{in}})$$

The out commitment matrix is denoted as $\mathsf{cm}_{\mathsf{ou}} \in \mathcal{D}^{n*3}_{cm}$, where $\mathsf{cm}_{\mathsf{ou}} \leftarrow [\mathsf{C}^{\mathsf{T}}_{s'}, \mathsf{C}^{\mathsf{T}}_r, \mathsf{C}^{\mathsf{T}}_x]$, *i.e.*, $\mathsf{cm}_{\mathsf{ou}}[\cdot][0]$ refers to $\mathsf{C}_{s'}$, $\mathsf{cm}_{\mathsf{ou}}[\cdot][1]$ refers to C_r and $\mathsf{cm}_{\mathsf{ou}}[\cdot][2]$ refers to C_x. The $\mathsf{Q}' \in \mathcal{D}^{n+1}_{coin}$ is the balance vector after the protocol execution. The status of the MPT $st \in \{\emptyset, \texttt{COMPLETE}, \texttt{ABORT}\}$, where $\emptyset$ denotes that the negotiation does not succeeds, the `ABORT` means that the negotiation succeeds but the evaluation does not successfully complete. In the case where all subjects in $\bar{\boldsymbol{P}}^+$ are honest, we write the protocol execution as

$$\mathsf{ou}, \mathsf{cm}_{\mathsf{ou}}, proof, st, \mathsf{Q}' \leftarrow REAL_{\pi}(\mathsf{Q}, f_{\mathcal{P}}, \mathsf{q}, \mathsf{in}, \mathsf{cm}_{\mathsf{in}})$$

C.3 Security definitions

To better sketch the ability of the $\mathcal{A}$, we denote the set of honest parties in $\bar{\boldsymbol{P}}^+$ as $\bar{\boldsymbol{P}}^+_H$, and the set of malicious subjects in $\bar{\boldsymbol{P}}^+$ as $\bar{\boldsymbol{P}}^+_M$, *i.e.*, $\bar{\boldsymbol{P}}^+_M \leftarrow \bar{\boldsymbol{P}}^+ \backslash \bar{\boldsymbol{P}}^+_H$. Similarly, the honest parties in only $\bar{\boldsymbol{P}}$ is denoted as $\bar{\boldsymbol{P}}_H$ and the malicious parties in only $\bar{\boldsymbol{P}}$ is denoted as $\bar{\boldsymbol{P}}_M \leftarrow \bar{\boldsymbol{P}} \backslash \bar{\boldsymbol{P}}_H$.

We first define the basic *correctness* property. Intuitively, *correctness* states that if all subjects in $\bar{\boldsymbol{P}}^+$ behave honestly, every party $P_i \in \bar{\boldsymbol{P}}$ get correct output and get their collateral back.

Definition 1 (Correctness). *Protocol π_{Cloak} run by honest subjects $\bar{\boldsymbol{P}}^+$ satisfies the* correctness *property if for every n-party program $f_{\mathcal{P}}$, $\mathsf{q} \in \mathcal{D}^n_{dep}$, $s \in \mathcal{D}^n_s$, $x \in \mathcal{D}^n_x$ and $\mathsf{Q} \in \mathcal{D}^{n+1}_{dep}$, the output of the protocol $REAL_{\pi}(\mathsf{Q}, f_{\mathcal{P}}, \mathsf{q}, \mathsf{in}, \mathsf{cm}_{\mathsf{in}})$ is that $\forall P_i \in \bar{\boldsymbol{P}}$:*

$$Pr[\mathsf{ou}[i][\cdot] \leftarrow [s'[i], r[i]] \ and \ \mathsf{Q}'[i] \geq \mathsf{Q}[i]] = 1$$

The $(s', r, \mathsf{C}_{s'}, \mathsf{C}_r, \mathsf{C}_x, proof, \mathsf{q}') \leftarrow \texttt{eval}(f_{\mathcal{P}}, \mathsf{q}, \mathsf{in}, \mathsf{cm}_{\mathsf{in}})$.

Next, we define the *confidentiality.*

Definition 2 (Confidentiality). *Protocol π_{Cloak} run by subjects $\bar{\boldsymbol{P}}^+$ satisfies the* confidentiality *property if for every n-party program $f_{\mathcal{P}}$, for every adversary $\mathcal{A}$ corrupting parties from $\bar{\boldsymbol{P}}^+$, for every $\mathsf{q} \in \mathcal{D}^n_{coin}$, $s \in \mathcal{D}^n_s$, $x \in \mathcal{D}^n_x$ and $\mathsf{Q} \in \mathcal{D}^{n+1}_{dep}$, the protocol $REAL_{\pi,\mathcal{A}}(\mathsf{Q}, f_{\mathcal{P}}, \mathsf{q}, \mathsf{in}, \mathsf{cm}_{\mathsf{in}})$ is such that: $\forall s'_* \in \mathcal{D}_{s'}, r_* \in \mathcal{D}_r$, it holds that $\forall \mathcal{A}$ corrupting parties in $\bar{\boldsymbol{P}}_M \cup \{\boldsymbol{E}\}$ where $\bar{\boldsymbol{P}}_M \ \$ \ \bar{\boldsymbol{P}}$*

$$Pr[\mathsf{ou}[j][\cdot] = [s'[j], r[j]] \mid P_j \in \bar{\boldsymbol{P}}_H] = Pr[\mathsf{ou}[j][\cdot] = [s'_*, r_*]]$$

The $(s', r, \mathsf{C}_{s'}, \mathsf{C}_r, \mathsf{C}_x, proof, \mathsf{q}') \leftarrow \texttt{eval}(f_{\mathcal{P}}, \mathsf{q}, \mathsf{in}, \mathsf{cm}_{\mathsf{in}})$.

satisfies the *State availability* property if for every n-party program $f_{\mathcal{P}}$, for every adversary $\mathcal{A}$ corrupting parties from $\bar{\boldsymbol{P}}^+$, for every $\mathsf{q} \in \mathcal{D}^n_{coin}$, $s \in \mathcal{D}^n_s$, $x \in \mathcal{D}^n_x$ and $\mathsf{Q} \in \mathcal{D}^{n+1}_{dep}$, the output of the protocol $REAL_{\pi,\mathcal{A}}(\mathsf{Q}, f_{\mathcal{P}}, \mathsf{q}, \mathsf{in}, \mathsf{cm}_{\mathsf{in}})$ is such that $\forall P_i \in \bar{\boldsymbol{P}}_H$:

Next, we formally define the *public verifiability.*

Definition 3 (Public verifiability). *Protocol π_{Cloak} run by subjects $\bar{\boldsymbol{P}}^+$ satisfies the* Public verifiability *property if for every n-party program $f_{\mathcal{P}}$, for every adversary $\mathcal{A}$ corrupting parties from $\bar{\boldsymbol{P}}^+$, for every*

$\mathsf{q} \in \mathcal{D}^n_{coin}$, $s \in \mathcal{D}^n_s$, $x \in \mathcal{D}^n_x$ and $\mathsf{Q} \in \mathcal{D}^{n+1}_{dep}$, the output of the protocol $REAL_{\pi,\mathcal{A}}(\mathsf{Q}, f_{\mathcal{P}}, \mathsf{q}, \mathsf{in}, \mathsf{cm}_{\mathsf{in}})$ is such that both of the following must be true:

- $\forall\ (s', r, \mathsf{C}_{s'}, \mathsf{C}_r, \mathsf{C}_x, proof, \mathsf{q}') \leftarrow \mathtt{eval}(f_{\mathcal{P}}, \mathsf{q}, \mathsf{in}, \mathsf{cm}_{\mathsf{in}})$:
  $Pr[\mathtt{verify})(proof, \mathsf{cm}_{\mathsf{in}}, \mathsf{cm}_{\mathsf{ou}}, H_f, H_{\mathcal{P}}) = 1] = 1$
- $\forall\ (s', r, \mathsf{C}_{s'}, \mathsf{C}_r, \mathsf{C}_x, proof, \mathsf{q}') \leftarrow \mathtt{eval}(f_{\mathcal{P}}, \mathsf{q}, \mathsf{in}, \mathsf{cm}_{\mathsf{in}})$:
  $Pr[\mathtt{verify}(proof, \mathsf{cm}_{\mathsf{in}}, \mathsf{cm}_{\mathsf{ou}}, H_f, H_{\mathcal{P}}) = 1] = 0$

Then we define the security property *executor balance security* which means the executor cannot lose money if it behaves honestly.

Definition 4 (Executor balance security). *Protocol π_{CLOAK} run by subjects $\bar{\mathbf{P}}^+$ satisfies the* executor balance security *property if for every n-party program $f_{\mathcal{P}}$, for every adversary $\mathcal{A}$ corrupting only parties from $\bar{\mathbf{P}}$ (the executor is honest), for every $\mathsf{q} \in \mathcal{D}^n_{coin}$, $s \in \mathcal{D}^n_s$, $x \in \mathcal{D}^n_x$ and $\mathsf{Q} \in \mathcal{D}^{n+1}_{dep}$, the output of the protocol $REAL_{\pi,\mathcal{A}}(\mathsf{Q}, f_{\mathcal{P}}, \mathsf{q}, \mathsf{in}, \mathsf{cm}_{\mathsf{in}})$ is such that:*

$$Pr[\mathsf{Q}'[n+1] \geq \mathsf{Q}[n+1]] = 1$$

Finally, we define *financial fairness* which in high level states that if at least one party $P_i \in \bar{\mathbf{P}}$ is honest, then must cause one of the following two events: (i) the protocol correctly evaluates the program and delivers the outputs; (ii) all honest parties output `ABORT`, stay financially neutral and at least one corrupt party must have been punished on-chain.

Definition 5 (Financial fairness). *Protocol π_{CLOAK} run by subjects $\bar{\mathbf{P}}^+$ satisfies the* financial fairness *property if for every n-party program $f_{\mathcal{P}}$, for every adversary $\mathcal{A}$ corrupting parties from $\bar{\mathbf{P}}^+_M \subsetneq \bar{\mathbf{P}}^+$, for every $\mathsf{q} \in \mathcal{D}^n_{coin}$, $s \in \mathcal{D}^n_s$, $r \in \mathcal{D}^n_r$ and $\mathsf{Q} \in \mathcal{D}^{n+1}_{dep}$, the output of the protocol $REAL_{\pi,\mathcal{A}}(\mathsf{Q}, f_{\mathcal{P}}, \mathsf{q}, \mathsf{in}, \mathsf{cm}_{\mathsf{in}})$ is such that one of the following statements must be true:*

(i) $st = \emptyset$, $\forall P_i \in \bar{\mathbf{P}}_H : \mathsf{Q}'[i] \geq \mathsf{Q}[i]$

(ii) $st = \mathtt{ABORT}$, $\forall P_i \in \bar{\mathbf{P}}_H :\ \mathsf{Q}'[i] \geq \mathsf{Q}[i]$ *and* $\sum_{j\in\bar{P}^+_M} \mathsf{Q}'[j] < \sum_{j\in\bar{P}^+_M} \mathsf{Q}[j]$

(iii) $st = \mathtt{COMPLETE}$, $\forall P_i \in \bar{\mathbf{P}}_H : \mathsf{Q}'[i] \geq \mathsf{Q}[i] - \mathsf{q}[i] + \mathsf{q}'[i]$

D SECURITY PROOF OF CLOAK PROTOCOL

We have informally explained the main theorem of π_{CLOAK} in Section 6. Here we formally state and prove the theorem.

Theorem 1 (Formal statement). *Assume a EUF-CMA secure signature scheme, a IND-CCA2 encryption scheme, a hash function that is collision-resistant, preimage and second-preimage resistant. a Trusted Execution Environment emulating the TEE ideal functionality and a blockchain emulating the blockchain ideal functionality, the* CLOAK *protocol π_{CLOAK} satisfies* ***correctness***, ***confidentiality***, ***public verifiability***, ***executor balance security***, *and* ***financial fairness*** *properties.*

D.1 Proof of correctness

As is defined by *correctness*, we consider the scenario when all subjects in $\bar{\mathbf{P}}^+$ are honest. To evaluate an MPT, π_{CLOAK} starts from *Negotiation phase.* Each party in $\bar{\mathbf{P}}$ independently interacts with both blockchain and the executor $\boldsymbol{E}$ to agree to an MPT proposal. Once the proposal is confirmed by TX_p on the blockchain, the collateral of each party $P_i \in \bar{\mathbf{P}}$ is also deducted, which means the coin balance of each party $P_i \in \bar{\mathbf{P}}$ becomes $\mathsf{Q}[i] - \mathsf{q}[i]$. Next, the protocol proceeds to the *Execution phase.* In this phase, for all $P_i \in \boldsymbol{P}$ the following holds: (1) P_i sends input $in_i \leftarrow (s[i], x[i])$ to the executor $\boldsymbol{E}$. The $\boldsymbol{E}$ (2) confirms that the input is correctly signed, then loads the input vector in with the blockchain view into the enclave $\mathcal{E}$. The $\mathcal{E}$ will again verify the signatures of parties and (3) additionally verify that the input $(s[i], x[i])$ match the confirmed input commitments on the blockchain. Then $\mathcal{E}$ evaluate the program $f_{\mathcal{P}}$ as $out \leftarrow (s', r) \leftarrow f_{\mathcal{P}}(in.s, in.x)$. Finally, the protocol moves to *Distribution phase.* The $\mathcal{E}$ first generate a symmetric key k and deliver the ciphertext of the output to each parties $Enc_k(out[i])$. When $\mathcal{E}$ ensures that all parties have received their corresponding output ciphertext by receiving parties' receipts, the $\mathcal{E}$ outputs transaction $TX_{com}(id_p, proof, \mathsf{C}_s, \mathsf{C}_{s'}, \mathsf{C}_r, k)$, which refunds $\mathsf{q}'[i]$ to party P_i and release k publicly. Hence, since $\mathsf{q}' = \mathsf{q}$, for every $P_i \in \bar{\mathbf{P}}$ it holds that $\mathsf{Q}'[i] \leftarrow \mathsf{Q}[i] - \mathsf{q}[i] + \mathsf{q}'[i] \geq \mathsf{Q}[i]$.

D.2 Proof of public verifiability

Recall that H_* refers to $\mathtt{hash}(*)$, for a specific MPT evaluation, $proof$ is $[H_{\mathsf{C}_{\mathcal{P}}}, H_{C_f}, H_{C_s}, H_{C_x}, H_{\mathsf{C}_{s'}}, H_{\mathsf{C}_r}]$ signed by $\mathcal{E}$. Since we assume the integrity of $\mathcal{E}$ is guaranteed, the $proof$ is therefore correctly computed inside $\mathcal{E}$ and signed by $pk_{\mathcal{E}}$. Since we assume that the correctness of hash function and π_{CLOAK}, the correctness of $proof$ holds. Furthermore, as the signature scheme is EUF-CMA, the signature of TX_{com} is unforgable, as well as the signature of $proof$. Therefore, the *public verifiability* holds.

D.3 Proof of executor balance security

We distinguish the following cases when the executor is honest.

(i): If the *negotiation phase* failed, it means that either parties in $\bar{\mathbf{P}}$ do not successfully fit the settlement condition of the negotiation so that none TX_p is released, or the release TX_p failed in being confirmed on the blockchain. In both scenarios, the collateral of the executor for that MPT will not be deducted on blockchain.

(ii): If the parties in $\bar{\mathbf{P}}$ agree on an MPT proposal during the *Negotiation phase*, it means the proposal is successfully confirmed on the blockchain so that the collateral of both all parties in $\bar{\mathbf{P}}$ and the executor is successfully deducted. Then, if at least one party does not provide correct signed input in *Execution phase* even after the ***challenge-response submission*** case, then the enclave will output the transaction $TX_{pns}(id_p, \bar{\mathbf{P}}'_M, q)$ that returns the deposit back to the executor.

(iii): Similar to *(ii)*, if both the *Negotiation phase* and the *Execution phase* successfully completes, the enclave then deliver the ciphertext to all parties. Next, if at least one party does not provide correctly the signed receipt, even after the ***challenge-response delivery*** case, then the enclave will also output the transaction $TX_{pns}(id_p, \bar{\mathbf{P}}'_M, q)$ to return the collateral back to the executor.

(iv) If the *Negotiation phase*, *Execution phase*, and delivering the ciphertext of outputs successfully completes, the enclave outputs the transaction $TX_{com}(id_p, proof, \mathsf{C}_s, \mathsf{C}_{s'}, \mathsf{C}_r, k)$ that returns the collateral of the MPT to the executor.

It remains to discuss whether the transactions in cases *(ii)* and *(iii)* are valid when posted to the blockchain by the executor, *i.e., st* =

SETTLE when these transactions are posted. The only transaction that modifies st on-chain is TX_{out}, which is posted by parties and only accepted after the τ_{com}-th block after h_{cp}, where h_{cp} is the height of block with TX_p. Let δ upper bound the block number from a transaction is published to transaction pool to it is included in a block, γ upper bound the block time from a transaction is included in a block to it is confirmed, λ upper bound the block number for executing the off-chain program, ϵ upper bound the block number waiting for collecting party inputs and receipts. Then, we set $\tau_{com} \geq 5(\delta+\gamma)+\lambda+2\epsilon$. Specifically, starting from the block height h_{cp} where TX_p is included on chain, confirming TX_p costs γ blocks first. Because waiting for party inputs costs ϵ, we set $t_e \leftarrow \gamma+\epsilon$. Next, a possible challenge-response submission stage needs to publish and confirm two transactions, which costs $2(\delta+\gamma)$ blocks. Therefore, we set $\tau_{sub} \leftarrow t_e+2(\delta+\gamma)$. After that, executing the program in enclave costs λ blocks and waiting for receipts costs ϵ, so we have $t_d \leftarrow \tau_{sub}+\lambda+\epsilon$. As a possible challenge-response delivery stage costs $2(\delta+\gamma)$, we set $\tau_{rec} \leftarrow \tau_{sup}+\lambda+\epsilon+2(\delta+\gamma)$. Finally, since publishing TX_{com} and including it in a block costs δ blocks, we set $\tau_{com} \leftarrow \tau_{rec}+\delta$. In conclusion, the honest executor always has time to publish TX_{pns} or TX_{com} with specified τ_{com}.

D.4 Proof of financial fairness

We prove the financial fairness of π_{CLOAK} phases by phases. First, we consider the *Negotiation phase*. Briefly, we will show that if the *Negotiation phase* does not complete successfully then all honest parties in $\bar{\boldsymbol{P}}_H$ will not go into SETTLE and stay financially neutral.

Lemma 2. *If there exist an honest party P_i stay at $st = \emptyset$, then the statement (i) of the financial fairness property holds.*

Proof: According to π_{CLOAK}, there are only one case when an honest party P_i will stay at $\emptyset$:

- No TX_p is confirmed on the blockchain.

Specifically, this scenario happens in following reasons: Parties in $\bar{\boldsymbol{P}}$ either fail to agree on an MPT proposal, preventing the enclave from constructing and releasing the TX_p, or the TX_p is released but fails to be confirmed on blockchain for reasons like that at least one subject in $\bar{\boldsymbol{P}}^+$ does not have enough coins to be deducted as the collateral for the MPT. However, no matter what reasons fail the confirmation of TX_p, all parties and the executor in π_{CLOAK} only identify the st of an MPT by reading its confirmed status from the blockchain. Since we have assumed that the blockchain has ideal consistency and availability, all honest parties can access the consistent blockchain view. Therefore, if the TX_p is successfully confirmed on-chain which means that parties' collateral has been successfully deducted, honest parties will immediately identify the MPT as SETTLE. In other words, if at least an honest party stay at $\emptyset$, the TX_p must be not successfully confirmed so that none of parties' collateral is deducted, *i.e.*, $\mathsf{Q}'[i] \geq \mathsf{Q}[i]$.

Next, we show that the financial fairness also holds even if MPT failed by ABORT after an successful *Negotiation phase.*

Lemma 3. *If there exist an honest party P_i such that st = ABORT, then the statement (ii) of the financial fairness property holds.*

Proof: Three cases exists when an honest party P_i outputs ABORT:

- (i) Before the τ_{com}-th block succeeding to the block confirming the TX_p, The $TX_{pns}(id_p, \bar{\boldsymbol{P}}'_M, p)$ is published on the blockchain after a *challenge-response submission* stage.
- (ii) Before the τ_{com}-th block succeeding to the block confirming the TX_p, The $TX_{pns}(id_p, \bar{\boldsymbol{P}}'_M, p)$ is published on the blockchain after a *challenge-response delivery* stage.
- (iii) After the τ_{com}-th block succeeding to the block confirming the TX_p, The $TX_{out}(id_p)$ is published on the blockchain.

We first consider the case (i) where malicious parties do not provide inputs after the negotiation succeeded. According to Algorithm 2, the enclave $\mathcal{E}$ release a transaction $TX_{pns}(id_p, \bar{\boldsymbol{P}}'_M, q)$ if and only if the executor calls the $\mathcal{E}.punish$ with a blockchain view which shows that parties in the non-empty set $\bar{\boldsymbol{P}}'_M$ did not provide their inputs even though they were challenged. By the definition of TX_{pns}, all honest parties, *i.e.*, parties not in $\bar{\boldsymbol{P}}'_M$, and the executor will get their collateral back and the parties in $\bar{\boldsymbol{P}}'_M$ get nothing. In other word, for $\forall P_i \in \bar{\boldsymbol{P}}'_M$ it holds that $\mathsf{Q}'[i] = \mathsf{Q}[i] - \mathsf{q}[i]$. Since $\mathsf{q}[i]$ ¡ 0 and $b\bar{m}P'_M < \emptyset$, at least one malicious party lost coins. Therefore, the inequality (ii) in Definition 5 holds.

We then consider the case (ii) where malicious parties do not response receipts when they received the ciphertext of outputs. According to the definition of Algorithm 2, again, the $\mathcal{E}$ output a transaction TX_{pns} if and only if the executor proves that the ciphertext of outputs has been publicly sent to parties as challenges on the blockchain but the parties being challenged did not response with their receipts. Similar to the case (i), for $\forall P_i \in \bar{\boldsymbol{P}}'_M$ it also holds that $\mathsf{Q}'[i] = \mathsf{Q}[i] - \mathsf{q}[i]$, *i.e.*, the inequality (ii) in Definition 5 holds.

Finally we consider the case (iii) where indicates a malicious executor. In this case, the timeout transaction TX_{out} is posted on the blockchain which mean that every $P_i \in \bar{\boldsymbol{P}}$ gets $\mathsf{q}[i]$ coins back, *i.e.*, $\mathsf{Q}'[i] = \mathsf{Q}[i]$, and the executor loses all its collateral for the evaluation, *i.e.*, $\mathsf{Q}'[n+1] = \mathsf{Q}[n+1] - \sum_{P_i \in \bar{P}} \mathsf{q}[i]$. Because the malicious executor lost collateral and no other malicious party earned any collateral, the inequality (ii) in Definition 5 holds.

Lemma 4. *If there exist an honest party P_i such that st 8 ABORT, then the statement (iii) of the financial fairness property holds.*

Proof: According to Algorithm 1 , the protocol outputs $\mathsf{ou}[i]$ 8 ABORT if and only if a transaction $TX_{com}(id_p, proof, \mathsf{C}_s, \mathsf{C}_{s'}, \mathsf{C}_r, k)$ is posted on the blockchain before the $h_{cp}+\tau_{com}$-th block. Furthermore, by definition of enclave program in Algorithm 2, the enclave $\mathcal{E}$ releases the TX_{com} if and only if all parties have received the ciphertext of their outputs before the $h_{cp}+\tau_{com}$-th block. Since the unique confirmed TX_{com} publishes the k to all parties, each party $P_i \in \bar{\boldsymbol{P}}$ can decrypt the output ciphertext to get $\mathsf{ou}[i]$. Besides, as the TX_{com} also refund the collateral $\mathsf{q}'[i]$ of each party P_i back, we know $\forall P_i \in \bar{\boldsymbol{P}}$ the $\mathsf{Q}'[i] = \mathsf{Q}[i] - \mathsf{q}[i] + \mathsf{q}'[i]$ holds. Therefore, the inequality (i) in Definition 5 holds.