DeCloak: Enable Secure and Cheap Multi-Party Transactions on Legacy Blockchains by a Minimally Trusted TEE Network

Qian Ren*, Yue Li*, Yingjun Wu, Yuchen Wu, Hong Lei, Lei Wang, and Bangdao Chen

Abstract— **The crucial blockchain privacy and scalability demand has boosted off-chain contract execution frameworks for years. Some have recently extended their capabilities to transition blockchain states by off-chain multi-party computation while ensuring public verifiability. This new capability is defined as Multi-Party Transaction (MPT). However, existing MPT solutions lack at least one of the following properties crucially valued by communities: data availability, financial fairness, delivery fairness, and delivery atomicity.**

This paper proposes a novel MPT-enabled off-chain contract execution framework, DeCloak. Using TEEs, DeCloak solves identified properties with lower gas costs and a weaker assumption. Notably, DeCloak is the first to achieve data availability and also achieve all of the above properties. This achievement is coupled with its ability to tolerate all-but-one Byzantine parties and TEE executors. Evaluating 10 MPTs in different businesses, DeCloak reduces the gas cost of the SOTA, Cloak, by 65.6%. This efficiency advantage further amplifies with an increasing number of MPT's parties. Consequently, we establish an elevated level of secure and cheap MPT, being the first to demonstrate the feasibility of achieving gas costs comparable to Ethereum transactions while evaluating MPTs.

Index Terms—**Confidential Smart Contract, Multi-Party Computation, Trusted Execution Environment**

I. Introduction

While blockchains are rapidly developed and adopted in various scenarios, *e.g.*, DeFi and IoT, blockchain privacy and scalability issues have become two top concerns. These concerns motivate off-chain smart contract execution frameworks.

This work was supported in part by the National Key R&D Program of China (No. 2021YFB2700600); 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. (*Corresponding author: Hong Lei*)

* These authors contributed to the work equally and should be regarded as co-first authors.

Q. Ren is with The Blockhouse Technology Ltd., Oxford OX2 6XJ, UK. (email: qianren1024@gmail).

Y. Li is with the School of Computer Science, Peking University, Beijing, China 100871. (email: liyue_cs@pku.edu.cn)

L. Wang is with the Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai, China 200240. (email: wanglei@cs.sjtu.edu.cn)

H. Lei is with the School of Cyberspace Security (School of Cryptology), Hainan University, Hainan, China 570228.

Yi. Wu, Yu. Wu, H. Lei, and B. Chen are with the Oxford-Hainan Blockchain Research Institute and SSC Holding Company Ltd., Wok Park, Laocheng, Chengmai, Hainan, 571924 China. (email: yingjun, yuchen, leihong, bangdao@oxhainan.org).

Off-chain contract execution with MPC. The common idea of off-chain contract execution frameworks [1]–[3] is to offload the contract execution from the blockchain to off-chain systems, where the blockchain functions as a trust anchor to verify executions and store states. Subsequently, some promising solutions extend the off-chain contract to Multi-Party Transaction (MPT) [4], [5], including auction [6], personal finance [7] and deal matching [8], [9]. MPT refers to transitioning blockchain states by a publicly verifiable off-chain MPC, where the MPC takes on- and off-chain inputs from, and delivers on- and off-chain outputs to multiple parties, without leaking their inputs/outputs to the public or each other. For example, in a second-price auction [6], multiple mutually distrustful parties perform an auction on their confidential on-chain balances and off-chain bids jointly. When the auction finishes, the party with the highest bid wins and pays the second-highest price on-chain. To enable MPT, two kinds of solutions exist. The first is cryptography-based solutions, which adopt MPC [10]–[12] or Homomorphic Encryption (HE) [13] to allow parties jointly and confidentially evaluate a program off-chain, then commit the evaluation status/outputs on-chain. The second, TEE-based solutions [2], [5], [7], uses TEE to collect private data from parties, evaluates a program with the data inside enclaves, and finally commits the evaluation status/outputs on-chain.

Limitations. However, existing solutions of MPT suffer from at least one of the following flaws: (i) Do not achieve data availability, making them vulnerable to data lost when off-chain systems fail. For example, even with ZKP or TEE to prove the correct state transitions, users cannot know their balances if the system withholds updated states. This property is identified and keenly required by the Ethereum community [14], and the community has designed a series of measures to uphold it, *e.g.*, `calldata` [15]–[17] and `blob` [18], which are keys of the coming Cancun upgrade [19]; (ii) Do not achieve financial fairness, so they can only assume a rate of honest nodes exists but cannot monetarily urge profit-driven nodes to behave honestly or punish misbehaved nodes; (iii) Do not achieve delivery fairness, which requires delivering outputs to corresponding parties at almost the same time. Formally, we say an MPC protocol achieves Δ-fairness if the time of different parties receiving their outputs distributes in a Δ-bounded period. A large Δ will lead to several attacks, *e.g.*, if a party before others knows that an MPT buys an ERC20 token and changes the trade rate, it can front-run an arbitrage transaction, so-called front-running attacks, *e.g.*, MEV [20]; (iv) Do not achieve delivery atomicity, *i.e.*, either committing new states and delivering off-chain outputs, or none happens. The lack of atomicity enables the adversary to obtain off-chain outputs before the new states are committed on-chain, leading to rewinding attacks. It also leads to risks for parties to lose their committed outputs [1] permanently; (v) Require high-cost interactions with the blockchain.

Our work. We propose DECLOAK, a novel MPT-enabled off-chain contract execution framework based on TEE. DECLOAK solves all the above problems with lower gas cost and weaker assumption. Specifically, to enable MPTs on a legacy blockchain, *e.g.*, Ethereum [21], we require multiple TEE executors to register their TEEs on a deployed DECLOAK contract. The contract thus be aware of all TEEs and will specify a specific TEE to serve all MPT. Then, multiple parties interact with the specified TEE off-chain to send MPTs. To achieve data confidentiality and availability (*cf.*, i), we propose a novel data structure of commitments. The structure allows each party and TEE to independently access the newest states from the blockchain, even though all other entities are unavailable. To achieve financial fairness (*cf.*, ii) and low cost (*cf.*, v), we propose a novel challenge-response subprotocol. It enables the DECLOAK contract to identify the misbehaviour of the specified TEE and replace it with another TEE. With the subprotocol, all honest entities among parties and TEE executors will never lose money, and at least one misbehaved entity will be punished. To achieve atomicity (*cf.*, iv) and delivery fairness (*cf.*, iii), we require all TEEs to independently release the keys of output ciphertext only after verifying that the output commitments have been accepted and confirmed on-chain. This way, multiple parties obtain their corresponding outputs almost simultaneously. Consequently, DECLOAK achieves the data availability, financial fairness, delivery fairness, and delivery atomicity of MPTs simultaneously with only 34.4% gas cost of the SOTA, Cloak [5], while assuming at least one party and TEE executor are honest. Last, we demonstrate how to optimize or prune DECLOAK for simpler or less secure scenarios, including how further to trade some secure properties for lower gas costs.

Contributions. Our main contributions are as follows.

- We design a novel off-chain contract execution framework, DECLOAK, which enables MPTs on legacy blockchains.
- We propose a protocol which achieves confidentiality, data availability, financial fairness, delivery fairness, and delivery atomicity of MPT simultaneously, while requiring at least one party and at least one of TEE executors to be honest.
- We implement DECLOAK and evaluate it on 10 MPTs with varying parties from 2 to 11.
- We demonstrate how to optimize further or fine-tune DECLOAK protocol to make trade-offs between security and cost for simpler MPT scenarios.

***Organization*.** We organize the paper as follows. Section II introduces MPT, primitives and symbols we used. Section III introduces a comparison of DECLOAK and related work. Section IV sketches DECLOAK. Section V details the DECLOAK protocol. Section VI introduce the DECLOAK prototype. In Section VII, we conduct a security analysis of DECLOAK. In Section IX, we discuss how to optimize the DECLOAK protocol and make trade-offs between the security and gas cost when degenerating MPT to simpler scenarios. Finally, we evaluate DECLOAK in Section VIII and conclude in Section X.

II. BACKGROUNG AND PRIMITIVES

In this section, we brief the problem Multi-Party Transaction (MPT) we target and primitives. The symbols used in this and the following sections are summarized in Table I.

A. *Multi-Party Transaction*

Informally, Multi-Party Transaction (MPT) refers to a transaction which transitions states on-chain by a publicly verifiable off-chain MPC. The off-chain MPC in an MPT takes on- and off-chain inputs and delivers on- and off-chain outputs. Formally, MPT is modeled as below [4], [5].

$$c_{s_1},\dots,c_{s_n} \overset{c_f,\ c_{x_1},\dots,c_{x_n}}{\Longrightarrow} c_{s'_1},\dots,c_{s'_n},c_{r_1},\dots,c_{r_n},proof$$
$$\mid s_1,\dots,s_n \overset{f(x_1,\dots,x_n)}{\Longrightarrow} s'_1,\dots,s'_n,r_1,\dots,r_n$$

For a blockchain and an array of parties $\boldsymbol{P}$ where $|\boldsymbol{P}| = n$ $(n \in \mathbf{Z}^* \wedge n > 1)$, we denote a party $\boldsymbol{P}[i]$ as P_i. An MPT takes as inputs a secret parameter x_i and old state s_i from each P_i, confidentially evaluates f off-chain, then delivers a secret return value r_i and new state s'_i to P_i, while publishing their commitments $c_{x_i}, c_{s_i}, c_f, c_{s'_i}, c_{r_i}$ and a $proof$ on-chain. An MPT should satisfy the following properties.

- *Correctness*: When each P_i providing x_i, s_i obtains s'_i, r_i, it must hold that
$$s_1,\dots,s_n \overset{f(x_1,\dots,x_n)}{\Longrightarrow} s'_1,\dots,s'_n,r_1,\dots,r_n$$
- *Confidentiality*: Each P_i cannot know $\{x_j, s_j, s'_j, r_j | j \neq i\}$ except those that can be derived from public info and the secrets it provides.
- *Public verifiability*: With $proof$, any node can verify that the state transition from the old state commitments array $\boldsymbol{c}_s \leftarrow [c_{s_i}|_{1..n}]$ to new state commitments array $\boldsymbol{c}_{s'} \leftarrow [c_{s'_i}|_{1..n}]$. And it is caused by an unknown function f (committed by c_f) which takes unknown parameter x_i (committed by c_{x_i}) and old state s_i (committed by c_{s_i}) as inputs , and outputs unknown new state s'_i (committed by $c_{s'_i}$) and return value r_i (committed by c_{r_i}).

The generality of MPT makes it easy to be applied to various scenarios [5], [10]. For example, recall the second-price auction in Section I. The bids should keep private to their corresponding parties, *i.e.*, *confidentiality* is held; The public (*e.g.*, blockchain miners) ought to verify that the output is the correct output of a claimed joint auction, *i.e.*, the *correctness* and *public verifiability* hold. We demonstrate more MPT scenarios in Section VIII.

B. *Trusted Execution Environment*

Trusted Execution Environment (TEE) is a technology which enables an isolated execution environment, a so-called enclave, to ensure the integrity and confidentiality of its inside

Table I
A SUMMARY OF MAIN SYMBOLS

Topic	Symbol	Name	Description
MPT	$\boldsymbol{P}$ (P_i)	Parties	An array of an MPT's participants where $P_i \leftarrow \boldsymbol{P}[i]$
	x_i, s_i, f, s'_i, r_i	-	P_i's parameter, old state, logic/program, new state and return value
	$c_{x_i}, c_{s_i}, c_f, c_{s'_i}, c_{r_i}$	-	The commitment of P_i's parameter, old state, new state and return value
	$\mathscr{P}, proof$	-	An MPT's logic f, policy $\mathscr{P}$, and publicly verifiable $proof$
Framework	BC	Blockchain	A BC enables Turing-complete smart contracts
	DN $(\boldsymbol{E}, \boldsymbol{\mathcal{E}}, E_i, \mathscr{E}_i)$	DECLOAK network	A network DN consisting of an array of executors $\boldsymbol{E}$ and TEEs $\boldsymbol{\mathcal{E}}$, where $E_i \leftarrow \boldsymbol{E}[i]$ and $\mathscr{E}_i \leftarrow \boldsymbol{\mathcal{E}}[i]$
Protocol	$E^*, \mathscr{E}^*$	Specified TEE/executor	The specified TEE $\mathscr{E}^*$ and its executor E^* where $\mathscr{E}^* \leftarrow \boldsymbol{\mathcal{E}}[0]$
	(sk_i, pk_i, ad_i)	Party account	The private key, public key, and address of the party P_i's account
	$(sk_{\boldsymbol{\mathcal{E}}}, pk_{\boldsymbol{\mathcal{E}}}, ad_{\boldsymbol{\mathcal{E}}})$	Network account	The private key, public key, and address of the network account shared among $\boldsymbol{\mathcal{E}}$
	$Proc_{\mathrm{nneg}}, Proc_{\mathrm{fdel}}$ $Proc_{\mathrm{dcmt}}, Proc_{\mathrm{rcha}}$	Subprotocols	Nondeterministic negotiation, Δ-fair delivery, data commitment, and challenge-response subprotocols, respectively
	p, p'	proposals	An initiated MPT proposal p and its corresponding settled proposal p'
	TX_{chaT}	*challengeTEE*	A transaction from the specified TEE $\boldsymbol{\mathcal{E}}[0]$ to publicly challenge the malicious executor
	TX_{ack_i}	*acknowledge*	A transaction from the party P_i to publicly join the MPT proposal
	TX_{fneg}	*failNegotiation*	A public response from the specified TEE $\boldsymbol{\mathcal{E}}[0]$ to TX_{chaT} to signal the negotiation failure
	TX_{chaP}	*challengeParties*	A transaction from the specified TEE $\boldsymbol{\mathcal{E}}[0]$ to publicly challenge the malicious parties
	TX_{resP_i}	*partyResponse*	A public response from the party P_i to TX_{chaP}
	TX_{cmt}	*commit*	A transaction from the specified TEE $\boldsymbol{\mathcal{E}}[0]$ to commit and lock the MPT outputs
	TX_{com}	*complete*	A public response from the specified TEE $\boldsymbol{\mathcal{E}}[0]$ to TX_{chaT} to complete the MPT
	TX_{pnsP}	*punishParties*	A public response from the specified TEE $\boldsymbol{\mathcal{E}}[0]$ to TX_{chaT} to punish malicious parties
	TX_{pnsT}	*punishTEE*x	A transaction from anyone to punish the misbehaved TEE

code. Different TEEs have been presented, including open-source TEEs such as Keystone [22] as well as not, *e.g.*, Intel SGX [23], AMD SEV [24], ARM TrustZone [25]. To design TEE-agnostic protocols, we follow [2], [26] to model the ideal functionality of attestable TEEs. Specifically, each TEE is initialized with a pair of keys $(msk, mpk) \leftarrow \Sigma.\texttt{keyGen}(1^\lambda)$ by its manufacturer. Σ is a digital signature algorithm. msk is the TEE's built-in master secret key. mpk is the corresponding public key. For a program code $prog$, a TEE's ideal functionality provides the following APIs:

- $mpk \leftarrow \texttt{getpk}()$: query the mpk of the TEE.
- $eid \leftarrow \texttt{install}(prog)$: install the program $prog$ into the TEE as a new enclave of which the enclave id is eid.
- $(outp, \sigma) \leftarrow \texttt{resume}(eid, inp)$: resume the enclave eid to execute its program $prog$ with the fed input inp. $\sigma \leftarrow \Sigma.\texttt{sign}_{msk}(eid, prog, outp)$ is the signature of TEE which endorses that $outp$ is the obtained output.

With the obtained mpk, anyone trusts output $outp$ when $\Sigma.\texttt{verify}(\sigma, eid, prog, outp) = 1$, indicating the success of signature verification.

C. *Elliptic Curve Diffie–Hellman Key Exchange*

Elliptic Curve Diffie–Hellman (ECDH) is an Elliptic Curve variant of the Diffie-Hellman key exchange protocol. For two parties, each has an elliptic-curve public-private key pair, ECDH allows them to anonymously establish a shared secret over an insecure channel. Technically, a trivial ECDH is as follows [27], [28]. Say we have an ECC elliptic curve with a generator point G. Two parties P_0, P_1 want to establish a shared secret. Their key pairs are (sk_0, pk_0) and (sk_1, pk_1), respectively, where $pk_0 \leftarrow sk_0 \cdot G$ and $pk_1 \leftarrow sk_1 \cdot G$. Each party must know the other party's public key before the protocol. Then, each party P_i independently derives a shared secret by $k \leftarrow sk_i \cdot pk_{1-i}$. Note that no party other than these two can derive the same secret unless that party can solve the elliptic curve discrete logarithm problem. As the problem is believed to be more computationally difficult than the classical finite field discrete logarithm problem and elliptic curves, offer certain advantages in terms of key sizes and performance, ECDH has gained popularity and is increasingly replacing the traditional Diffie-Hellman key exchange protocol in various applications [29], *e.g.*, Google Chrome and Safari.

While the trivial ECDH above is vulnerable to man-in-the-middle attacks, the issue can be solved by requiring one of the two parties' public keys to be static, *i.e.*, the key's authenticity is assured by third-party means [30] like certificate, TEE attestation or blockchain. It is also suggested to corporate an ephemeral public key with the static key to achieve forward secrecy *etc.* [27], such as ECC MQV [30]. Moreover, instead of using the shared secret directly, deriving other keys, such as symmetric keys for AES, DES, *etc.*, from the secret to establish secure channels can reduce the risk of exposing parties' private keys. ECDH has been widely applied in many protocols, such as TLS. In this paper, we model the ECDH as $k \leftarrow \texttt{ecdh}(sk_i, pk_{1-i})$ to simplify the process where two parties P_0, P_1 derive a shared symmetric key k.

III. RELATED WORK

In this section, we first exemplify related work in categories, illustrate why they flaw key MPT properties, and then elaborate how DECLOAK superiors them by solving these flaws. We summarize the comparison in Table II.

TEE-based confidential smart contracts. Confide [31], Ekiden [1], CCF [34], and POSE [3] are designed to confide transaction inputs/outputs and states. However, they consider transactions as independent. Consequently, they do not involve multiple parties in their achieved properties, *e.g.*, verifiability and atomicity, nor involve multi-party specific properties, *e.g.*,

Table II

Comparing DeCloak with related work. The symbols ×, ❍, ◗ and ● refer to "non-related", "not-matched", "partially-matched" and "fully-matched" respectively. "**Adversary Model**" means how many Byzantine entities can be tolerant. "**Data availability**" means whether parties or TEEs can access states independently. "**Financial fairness**" means honest parties never lost money while at least one misbehaved node must be punished. "**Delivery fairness**" means either the MPT fails or parties obtain their outputs in almost the same time. "**Delivery atomicity**" means whether both committing of outputs and the delivery of output or none of them are guaranteed.

Approach	**Adversary Model**		**min(#TX)**	**Confidentiality**	**Public Verifiability**	**Data availability**		**Financial Fairness**	**Delivery Fairness**	**Delivery Atomicity**
	Parties	TEE Executors				Parties	TEEs			
Ekiden [1]	1^*	m^*-1	$O(1)$	●	◗	❍	❍	×	×	●
Confide [31]	1^*	$\lfloor(m^*-1)/2\rfloor$	$O(1)$	●	◗	❍	●	×	×	×
POSE [3]	1^*-1	m^*-1	$O(1)$	●	◗	❍	●	×	×	❍
Hawk [6]	n^*	×	$O(n)$	◗	◗	❍	×	●	❍	❍
ZEXE [32]	n^*	1^*	$O(1)$	◗	◗	❍	×	×	×	❍
Choudhuri *et al.* [33]	n^*	$m^* \|_{m=n}$	$O(1)$	●	◗	×	×	×	●	×
LucidiTEE [7]	n^*	m^*-1	$O(n)$	●	◗	❍	◗	×	◗	●
Fastkitten [2]	$(n^*+1^*)-1$		$O(n)$	◗	◗	❍	❍	●	❍	❍
Cloak [5]	$(n^*+1^*)-1$		$O(1)$	●	●	❍	●	●	❍	❍
DeCloak	n^*-1	m^*-1	$O(1)$	●	●	●	●	●	●	●

The * denotes the total number of the specific type of entities, *e.g.*, 1^* denotes the unique party/executor, n^* denotes all n parties, and m^* denotes all executors.

negotiation and fairness, while all the above is achieved by DeCloak.

Ekiden [1] decouples the consensus, execution, and key management to different nodes, where the last two kinds of nodes hold TEEs. However, users and executors' TEE do not hold data availability as they cannot access the on-chain states independently without requesting key management TEEs. Ekiden also proposes a two-phase protocol to achieve atomicity, which delivers the outputs' keys to users off-chain when the outputs have been committed on-chain. DeCloak, differently, achieves data availability for both users and executors' TEEs, by introducing a novel *data commitment subprotocol*. The subprotocol confidentially persists MPT commitments on the blockchain with low gas cost. Each party or TEE can access their data from the blockchain with only its known account private key. Therefore, data availability is still guaranteed even if the whole system is unavailable. DeCloak also achieves the atomicity with a similar two-phase protocol but releases the outputs' key to the blockchain for availability. Moreover, DeCloak adopts a different two-phase protocol, Δ-*fair delivery subprotocol*, in which each TEE interacts with blockchain instead of communicating with parties or other TEEs, to achieve delivery atomicity, conform confidential and data availability, minimizing the dependence on other nodes.

Confide [31] and CCF [34] are independent permissioned blockchains, rather than off-chain contract execution systems as DeCloak. Their TEE-enabled executors maintain a consensus, *e.g.*, RAFT, thereby tolerating only $< 1/2$ unavailable executors, while DeCloak tolerates all but one Byzantine executors. They store contract data (*e.g.*, code, states) by encrypting data with keys shared among TEEs, thereby achieving data availability of only TEEs. However, if TEE executors are unavailable, users will permanently lose their private data and on-chain assets. Instead, DeCloak achieves data availability for both users and TEEs.

POSE [3] proposes an off-chain contract execution system which features high system availability and no interaction with blockchain in optimistic cases. It introduces a challenge-response mechanism, ensuring the system's availability even if all-but-one executors are Byzantine. However, this mechanism failed in the multi-party scenario, which is solved by DeCloak . Moreover, similar to Ekiden, Confide and CCF, users of POSE cannot access their states independently. As POSE does not commit the off-chain state transition on-chain, it fails to achieve atomicity like DeCloak.

TEE-based smart contracts enabling MPCs. Choudhuri *et al.* [33], Fastkitten [2], LucidiTEE [7], Cloak [5] and **DeCloak** are all TEE-based smart contract frameworks enabling MPCs. However, only Cloak and DeCloak fully match the MPT properties, especially confidentiality and public verifiability, as other work fails to commit or verify all the off-chain MPC's elements on-chain, *e.g.*, state transitions or party identities. Moreover, DeCloak achieves more secure properties and lower gas cost than Cloak [5].

Choudhuri *et al.* [33] are the first to achieve Δ-fairness where $\Delta \approx 0$ for general-purpose MPC. They require each party to hold a TEE. Choudhuri *et al.* do not consider reading states on-chain, punishing misbehaved nodes, or committing states on-chain, thus being non-related to data availability, financial fairness, or delivery atomicity. DeCloak achieves all the above properties where the Δ-fairness is comparable without requiring any party to hold a TEE.

LucidiTEE [7] requires partial parties to hold TEEs for delivery fairness. However, the Δ of its fairness, *i.e.*, the period length in which the time of parties receiving outputs distributes, equals the generation time of Proof of Publication (PoP)[1] [1], [2], [35]. PoP is for proving TEE that a key-releasing transaction has been finalized on-chain, which costs over 50 block intervals on Ethereum [2]. Moreover, LucidiTEE requires each party to send a transaction to join an MPT or deposit, leading to $O(n)$ transactions. Instead, DeCloak

[1]Recall that PoP is a proof constructed for proving that a transaction has been confirmed on-chain

[2]For achieving ≤ 0.001 false negative and false positive under an adversary with $\leq 1/3$ computing power of Ethereum

achieves Δ of delivery fairness comparable to Choudhuri *et al.* and other MPT properties with the cost of $O(1)$.

Fastkitten [2] seeks to enable arbitrary multi-round MPC on Bitcoin. It lets parties execute a transaction with private inputs in TEEs and only submit new state commitments with TEE signatures on-chain, sacrificing public verifiability. It introduces a challenge-response mechanism to achieve financial fairness but requires each party to send a deposit transaction before each MPT, leading to $O(n)$ transactions. The mechanism also fails to work when the negotiation phase and multiple TEEs exist. Instead, DECLOAK achieves these properties additionally with data availability and delivery fairness while costing only $O(1)$ transactions.

Cloak [5] is the first to propose a *one-deposit-multi-transaction* mechanism, where each honest party deposits coins once globally and then joins MPTs infinitely. The mechanism reduces its required on-chain transactions to $O(1)$. DECLOAK adopts the same *one-deposit-multi-transaction* with Cloak. However, Cloak only commits the hash of MPT elements on-chain. Thereby their parties also cannot access their states without TEE executors, *i.e.*, lacking data availability. Its single TEE model fails in achieving delivery fairness and atomicity. These flaws are solved by DECLOAK without sacrificing efficiency.

Cryptography-based smart contracts enabling MPCs. Cryptography-based schemes usually combine MPC/HE with ZKP to enable MPTs. They underperform DECLOAK in both MPT properties and efficiency.

Without combining MPC/HE and ZKP, MPC/HE-based work like [36]–[38] achieves great confidentiality of off-chain multi-party programs but not targets public verifiability. ZKP-based solutions achieve public verifiability of the program but lack confidentiality inter parties. For example, Hawk [6] requires a tight-lipped manager to collect parties' secrets, execute a contract, and generate the ZKP proof. Thus the confidentiality of Hawk is limited. ZEXE [32] proves the satisfaction of predicates by ZKP proof without revealing party secrets to the public. However, generating the proof requires a party to know all predicate's secrets, thereby violating inter-party confidentiality.

Combining MPC with ZKP, public auditable MPC (PA-MPC) [10] is a new MPC primitive that achieves public verifiability, allowing multiple parties jointly evaluate a program and prove it to the public. 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 flaw efficiency and adversary models and fail to support nondeterministic negotiation or achieve financial fairness practically. Specifically, [10] requires trusted setup or un-corrupted parties. [39] is function-limited. [40] very recently achieves general-purpose PA-MPC but only supports circuit-compatible operations. None of the above solutions is for confidential smart contracts or can punish adversaries. Instead, using the same proof structure with Cloak, DECLOAK conforms to both confidentiality and public verifiability. While the underlining MPC of [33], [36]–[38] requires honest-majority parties, DECLOAK secures the system under a Byzantine adversary corrupting all-but-one parties and all-but-one TEE executors.

IV. DECLOAK DESIGN

In this section, we first overview the system model, adversary model, and system goals of DECLOAK. Then, we overview DECLOAK protocol and highlight the challenges we handled and corresponding countermeasures.

A. *System Model*

Figure 1 shows the framework of DECLOAK, *i.e.*, a system consisting of three components.

Blockchain (*BC*). A blockchain *BC* that can deploy and evaluate Turing-complete smart contracts. *BC* satisfies the common prefix, chain quality and chain growth [41], so it can handle and reach consistency on new transactions continuously. A Proof of Publication (PoP) scheme exists to prove TEEs that a transaction has been finalized on *BC*. The above assumptions are also adopted by [1], [2], [5], [35], and many well-known blockchains conforms, *e.g.*, Ethereum [21].

Parties (*P*). An array of parties $\boldsymbol{P}$ participating a specific MPT. An honest party can access the latest view of the blockchain *BC* and trust the data it reads from *BC*. The party trusts its platform and running code but not others. An honest party also trusts the integrity and confidentiality of all TEEs it attested. An honest party never reveals its secrets to others except attested TEEs.

DECLOAK network (*DN*). A network consists of multiple TEE executors. We assume each executor E holds a TEE and instantiate only one enclave in its TEE by $\mathscr{E}_{eid} \leftarrow TEE.\texttt{install}(\mathscr{E})$ where the program $\mathscr{E}$ is shown in Algorithm 2. Without ambiguity, we denote $\mathscr{E}_{eid}$ as $\mathscr{E}$, which refers to the enclave and the TEE simultaneously. An honest TEE executor can access *BC*'s latest view and trust the data it reads from *BC*. An honest executor also trusts its platform and running code but not others. It also trusts the integrity and confidentiality of attested TEEs. We denote the array of all executors as $\boldsymbol{E}$ and all TEEs as $\boldsymbol{\mathcal{E}}$.

B. *Adversary Model*

We assume a Byzantine adversary presents in a DECLOAK system. A Byzantine adversary can corrupt all-but-one parties and all-but-one TEE executors. A corrupted party or executor can behave arbitrarily, *e.g.*, mutating, delaying and dropping messages, but never break the integrity and confidentiality of any TEE. Moreover, the adversary cannot interfere with the communications among honest entities, *e.g.*, the communications between honest parties and executors.

C. *System Goals*

Informally, we seek to achieve the following properties. These properties are formally defined in Section VII-A1.

Correctness. The outputs of a succeeded MPT must be the outputs of the MPT's function applied to the fed inputs.

Confidentiality. The inputs and outputs of an MPT are always confidential to their corresponding parties.

Public verifiability. The public, including *BC*, can verify the correctness of the state transition caused by an MPT and its commitments.
Data availability. No matter how the adversary behaves, if an MPT succeeds, it holds that each honest party or TEE with honest executor can independently access the plaintext of the newest states.
Financial fairness. If over party is honest, then either (i) the protocol correctly completes the MPT or (ii) all honest parties know that negotiation of the MPT failed and stay financially neutral or (iii) all honest parties know the protocol aborted, stay financially neutral and at least one of misbehaved entities must have been financially punished.
Delivery fairness. If over one TEE executor is honest, then either (i) all parties know their corresponding plaintext return values and new states within a Δ-bounded period, or (ii) no one knows its plaintext.
Delivery atomicity. If over one TEE executor is honest, then either (i) parties know plaintext new states or return values, and the new states have been committed on-chain, or (ii) new states are not committed, and no one obtains its plaintext.

D. *Protocol Workflow*

Figure 1 briefs the DECLOAK protocol π_{DECLOAK}. We assume all TEEs have been previously registered on-chain as a TEE list $\boldsymbol{\mathcal{E}}$. Then, π_{DECLOAK} starts to serve MPTs in four phases, *i.e.*, *global setup*, *negotiation*, *execution*, and *delivery* phases. The *global setup* phase happens only once for any party. Other three phases happen for each MPT.

- ***(0)*** **Global setup phase**: All parties and TEEs deposit some coins to the network account $ad_{\boldsymbol{\mathcal{E}}}$ on *BC*, where the account's private key is shared among all registered TEEs.
- ***(1)*** **Negotiation phase**: A party sends an MPT proposal p to the first TEE $\mathscr{E}^*$ in the registered TEE list to initiate an MPT. Upon receiving the proposal, the TEE $\mathscr{E}^*$ starts a *non-deterministic negotiation subprotocol* $Proc_{\text{nneg}}$. Specifically, the $\mathscr{E}^*$ signs and broadcasts the proposal to all parties. If any party wants to join or is required by the p, it responds with an acknowledgement to $\mathscr{E}^*$. The acknowledgement is essentially a signature of the party against the proposal. The $\mathscr{E}^*$ keeps collecting parties' acknowledgements. When the collected acknowledgements match the settlement condition of the negotiation phase (*e.g.*, the number of parties exceeds the number specified by p), $\mathscr{E}^*$ settles the proposal, deducts parties' collaterals from their coins cached in $\mathscr{E}^*$, and broadcasts the settled proposal p' to all parties.
- ***(2)*** **Execution phase**: Upon receiving p', each party involved submits its signed plaintext inputs (*i.e.*, parameters) to $\mathscr{E}^*$. $\mathscr{E}^*$ first reads old states on *BC* with their PoP[3], then evaluates the MPT's function to obtain the outputs (*i.e.*, return values and new states) inside.
- ***(3.1-3.2)*** **Delivery phase**: $\mathscr{E}^*$ starts a Δ-*fair delivery subprotocol* $Proc_{\text{fdel}}$. First, it follows a novel *data commitment subprotocol* $Proc_{\text{dcmt}}$ to generate one-time symmetric keys to compute the commitments of the outputs and publish the commitments on *BC* with the ciphertext of the keys (encrypted by a key shared among TEEs). Upon the commitments being confirmed, each $\mathscr{E} \in \boldsymbol{\mathcal{E}}$ independently verifies the confirmation, obtains the symmetric keys on *BC*, and then sends a transaction to reveal the committed outputs to each party respectively.

[3]We use the same PoP as [2], [5], [35]

If misbehaviours occur during the above phases, we devise a novel *challenge-response subprotocol* $Proc_{\text{rcha}}$ to identify and punish the misbehaved entities.

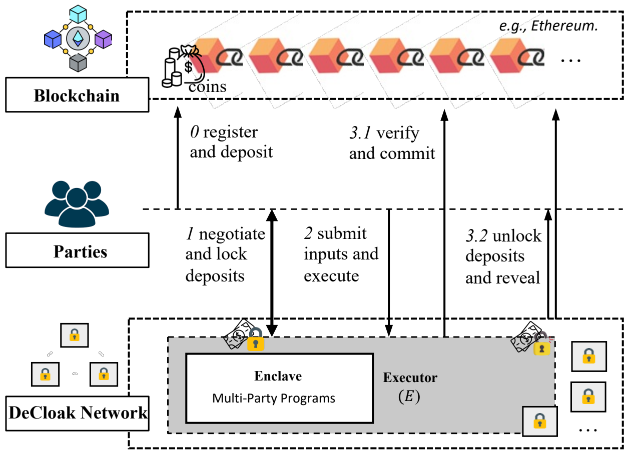

Figure 1. **The framework and workflow of DECLOAK.**

E. *Design Challenges and Highlights*

1) Achieve data availability of both TEEs and parties

The challenge here is how to (i) achieve the data availability of both parties and TEEs and (ii) harmonize the data confidentiality with the subprotocols for delivery atomicity and fairness. We introduce a novel *data commitment subprotocol* $Proc_{\text{dcmt}}$ based on ECDH to tackle these challenges. Specifically, we require all entities to commit P_i's private data d_i on blockchain in the following structure c_{d_i}.

$$c_{d_i} := [\mathtt{enc}_{k_{d_i}}(d_i), \mathtt{enc}_{k_{ie}}(k_{d_i}), P_i]$$

k_{d_i} denotes a one-time symmetric key for encrypting d_i. k_{ie} denotes generated symmetric key generated shared among P_i and all TEEs, $k_{ie} \leftarrow \mathtt{ecdh}(sk_i, pk_{\boldsymbol{\mathcal{E}}})$ and $k_{ie} \leftarrow \mathtt{ecdh}(sk_{\boldsymbol{\mathcal{E}}}, pk_i)$. Consequently, *cf.*, Section II-C, for (i), each party P_i and all TEEs can independently obtain k_{ie} without interacting with the other. And P_i needs only to hold its own private key sk_i to access/operate all its commitments on-chain. For (ii), DECLOAK first releases $c^*_{d_i}$, *i.e.*, c_{d_i} without $\mathtt{enc}_{k_{ie}}(k_{d_i})$, but with a extra k_{ie}'s ciphertext $\mathtt{enc}_{k_{\boldsymbol{\mathcal{E}}}}(k_{ie})$ where $k_{\boldsymbol{\mathcal{E}}} \leftarrow \mathtt{ecdh}(sk_{\boldsymbol{\mathcal{E}}}, pk_{\boldsymbol{\mathcal{E}}})$. So each TEE can independently obtain $k_{\boldsymbol{\mathcal{E}}}$ to decrypt k_{ie} as they share the network account $(sk_{\boldsymbol{\mathcal{E}}}, pk_{\boldsymbol{\mathcal{E}}}, ad_{\boldsymbol{\mathcal{E}}})$. Then, each TEE releases the missed $\mathtt{enc}_{k_{ie}}(k_{d_i})$ to *BC* only when the new state commitments have been finalized, achieving atomicity and fairness. Notably, without the ability to obtain $k_{\boldsymbol{\mathcal{E}}}$, an adversary neither can decrypt a corrupted parties' k_{ie}, k_{d_i} and further d_i during the waiting period of confirmation, nor decrypt honest parties' k_{ie} to break the confidentiality, although $k_{\boldsymbol{\mathcal{E}}}$ is shared among different MPTs

2) Achieve delivery fairness

When the specified TEE executor evaluated the MPT inside its TEE $\mathcal{E}^*$, $\mathcal{E}^*$ does not release the output immediately. Instead, $\mathcal{E}^*$ first follows the data commitment subprotocol to encrypt the outputs, then sends a transaction TX_{cmt} to publish the incomplete output ciphertext (*e.g.*, $c^*_{d_i}$) and their keys' ciphertext (*e.g.*, the ciphertext of k_{ie}) on-chain. The keys' ciphertext can be decrypted by all TEEs independently. However, each TEE obtains the keys and releases the missing party of the released output ciphertext (*e.g.*, $\mathtt{enc}_{k_{ie}}(k_{d_i})$) only when TX_{cmt} has been finalized on-chain. Since we assume the blockchain is ideally available, all honest TEE executors can feed the PoP of TX_{cmt} to their TEEs. Therefore, if at least one honest executor exists, parties communicating with all executors can simultaneously obtain the keys to decrypt the output ciphertext. We denote the above procedure as Δ-fair delivery subprotocol, $Proc_{\text{fdel}}$.

3) Resist Byzantine adversary with minimal transactions

This paper proposes a novel *challenge response subprotocol* $Proc_{\text{rcha}}$. At a high level, $Proc_{\text{rcha}}$ is designed with the following idea: when an honest party does not receive protocol messages off-chain from the specified TEE $\mathcal{E}^*$, it publicly challenges $\mathcal{E}^*$ with the proposal p on-chain. $\mathcal{E}^*$ can only avoid being punished if it can respond with expected outputs or prove that the problem is caused by some misbehaved parties rather than itself. Specifically, an MPT proposal only has three possible results: (i) NEGOFAILED, meaning the negotiation of the proposal failed; (ii) COMPLETED, meaning the MPT succeeded (iii) ABORTED, *i.e.*, some entities among parties and the specified executor misbehaved, making the MPT aborted. Therefore, the challenged $\mathcal{E}^*$ needs to respond with one of the following three results to prove its honesty: (i) sending a transaction TX_{fneg} to prove that the negotiation of the MPT failed; (ii) sending a transaction TX_{com} to complete the MPT and release its outputs; (iii) sending a transaction TX_{pnsP} to prove that it cannot complete the MPT as expected because some parties misbehaved after the negotiation succeeded rather than itself. If none of the above transactions can be sent, $\mathcal{E}^*$ will be punished. However, while (ii) is inherent in the success of MPT, how to achieve (i) and (iii) becomes challenging. To achieve (i), we require each proposal p to specify a block height h_{neg} to notify when the negotiation phase is expected to finish. Then, $\mathcal{E}^*$ can send a TX_{fneg} to fail the proposal on-chain if it verifies that the collected acknowledgements from both off-chain ***ack*** and on-chain $\boldsymbol{TX}_{ack}$ before h_{neg}-th block cannot satisfy the settlement condition. To achieve (iii), when $\mathcal{E}^*$ cannot complete the MPT, $\mathcal{E}^*$ needs to challenge those misbehaved parties to prove that the reason is some parties did not submit their inputs rather than itself.

V. DECLOAK PROTOCOL

This section details the DECLOAK protocol π_{DECLOAK}. Given an array of parties $\boldsymbol{P}$, a blockchain BC and a DECLOAK Network DN with $|\boldsymbol{P}| = n$ and $|\boldsymbol{E}| = |\boldsymbol{\mathcal{E}}| = m$. We assume a DECLOAK contract $\mathcal{V}$ (Algorithm 1) has been deployed on BC, and all TEEs have been registered in $\mathcal{V}$ as a list $\boldsymbol{\mathcal{E}}$. A common TEE network account $(sk_{\boldsymbol{\mathcal{E}}}, pk_{\boldsymbol{\mathcal{E}}}, ad_{\boldsymbol{\mathcal{E}}})$ has been synchronized among all TEEs. Each party P_i has an account (sk_i, pk_i, ad_i), referring to the private key, public key and address of the account, respectively. As each party P_i is identified by its address, we indiscriminately refer P_i to ad_i. For conciseness, we simplify $TEE.resume(\mathcal{E}, func||in)$ as $\mathcal{E}.\mathtt{func}(in)$ to express calling func of the TEE $\mathcal{E}$ with inputs in. Implicitly, parties always $\Sigma.\mathtt{verify}$ the messages received from $\mathcal{E}$.

π_{DECLOAK} is shown in Figure 2. We use d_i to denote the private data of P_i (*e.g.*, x_i, s_i, k_{s_i}), $\boldsymbol{d}$ to denote an array $[d_i|_{i\in[n]}]$ including all d_i from n parties (*e.g.*, $\boldsymbol{x}, \boldsymbol{s}, \boldsymbol{k_s}$). We let H_{d_i} denote $\mathtt{hash}(d_i)$ and H_d denote $\mathtt{hash}([d_i|_{i\in[n]}])$ (*e.g.*, $H_{\boldsymbol{c_x}}$ denotes the hash of the parameter commitment array $\mathtt{hash}([c_{x_i}|_{i\in[n]}])$).

A. Global Setup Phase

Before evaluating any MPT, each party P_i is supposed to *register* their account public key pk_i and *deposit* some coins with amount Q_i to the DECLOAK contract $\mathcal{V}$ (Algorithm 1). We stress that each party only needs to do it once.

B. Negotiation Phase

An MPT is started from its negotiation phase, where DECLOAK uses the *nondeterministic negotiation subprotocol* ($Proc_{\text{nneg}}$) to guide parties to reach an agreement on an MPT proposal. In detail, $Proc_{\text{nneg}}$ proceeds in two steps.

1.1: A party who wants to call an MPT f sends an MPT proposal $p \leftarrow (H_f, H_{\mathscr{P}}, q, h_{neg})$ to the first TEE $\mathcal{E}^*$ in the registered TEE list, *i.e.*, $\mathcal{E}^* = \boldsymbol{\mathcal{E}}[0]$, to initiate an MPT f. We call $\mathcal{E}^*$ as the specified TEE. Sending proposals to other TEEs will be rejected by the TEEs. $\mathscr{P}$ denotes a policy of f, capturing the settlement condition of the negotiation.[4] q denotes the collateral required for joining the proposed MPT. h_{neg} denotes that the proposal p is expected to be negotiated before the block height h_{neg}. Then, $\mathcal{E}^*$ computes $\mathtt{hash}(p)$ to be the proposal id id_p and broadcasts a signed (id_p, p) to parties.

1.2: Upon receiving (id_p, p), each P_i interested in the MPT autonomously responds with a signed acknowledgement ack_i to $\mathcal{E}^*$. The $\mathcal{E}^*$ receiving ack_i knows P_i's intent of joining the proposal id_p. $\mathcal{E}^*$ keeps collecting each party's acknowledgement ack_i until the acknowledgements match the settlement condition[5] in $\mathscr{P}$. Then, $\mathcal{E}^*$ constructs a settled proposal p' that expands p with the settled parties' addresses $\boldsymbol{P}$. Meanwhile, $\mathcal{E}^*$ locks q collateral from its and parties' cached on-chain coin balances, ensuring that any involved entity has enough collateral to be punished when it misbehaves. Then, $\mathcal{E}^*$ broadcasts (id_p, p') to notify the involved parties.

Otherwise, if $\mathcal{E}^*$ does not collect satisfied acknowledgements, a *challenge-response subprotocol* $Proc_{\text{rcha}}$ in section V-E will be triggered to identify misbehaviours.

[4]Please refer [5] for more details of $\mathscr{P}$.

[5]Settlement condition of negotiation is flexible, *e.g.*, the number of parties exceeds a specified threshold.

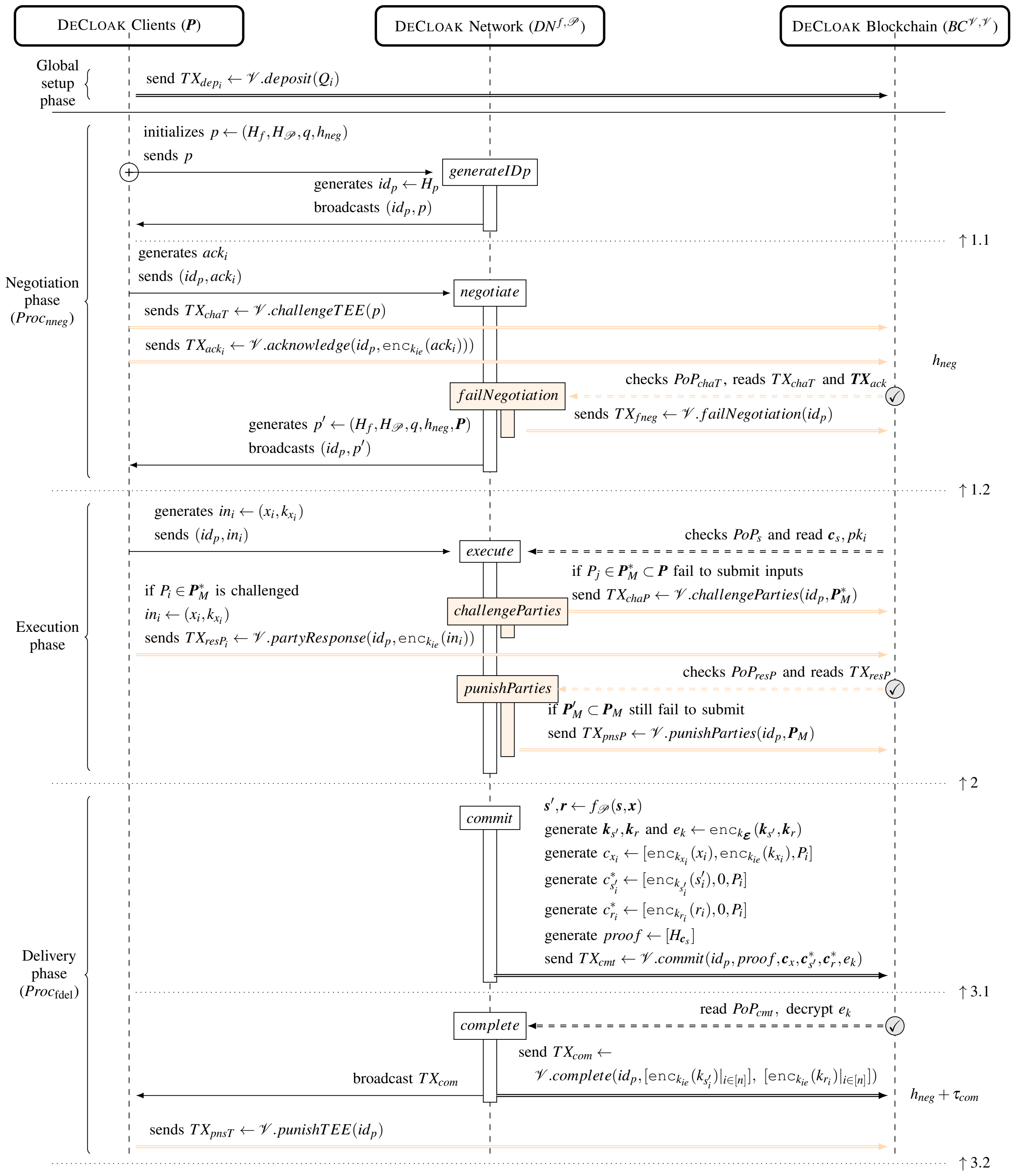
DeCloak Clients ($\boldsymbol{P}$)
DeCloak Network ($DN^{f,\mathscr{P}}$)
DeCloak Blockchain ($BC^{\mathscr{V},\mathscr{V}}$)
Global setup phase
send $TX_{dep_i} \leftarrow \mathscr{V}.deposit(Q_i)$
Negotiation phase ($Proc_{nneg}$)
initializes $p \leftarrow (H_f, H_{\mathscr{P}}, q, h_{neg})$
sends p
generateIDp
generates $id_p \leftarrow H_p$
broadcasts (id_p, p)
↑ 1.1
generates ack_i
sends (id_p, ack_i)
negotiate
sends $TX_{chaT} \leftarrow \mathscr{V}.challengeTEE(p)$
sends $TX_{ack_i} \leftarrow \mathscr{V}.acknowledge(id_p, \texttt{enc}_{k_{ie}}(ack_i)))$
h_{neg}
checks PoP_{chaT}, reads TX_{chaT} and $\boldsymbol{TX}_{ack}$
failNegotiation
sends $TX_{fneg} \leftarrow \mathscr{V}.failNegotiation(id_p)$
generates $p' \leftarrow (H_f, H_{\mathscr{P}}, q, h_{neg}, \boldsymbol{P})$
broadcasts (id_p, p')
↑ 1.2
Execution phase
generates $in_i \leftarrow (x_i, k_{x_i})$
sends (id_p, in_i)
execute
checks PoP_s and read $\boldsymbol{c}_s, pk_i$
if $P_j \in \boldsymbol{P}^*_M \subset \boldsymbol{P}$ fail to submit inputs
send $TX_{chaP} \leftarrow \mathscr{V}.challengeParties(id_p, \boldsymbol{P}^*_M)$
if $P_i \in \boldsymbol{P}^*_M$ is challenged
$in_i \leftarrow (x_i, k_{x_i})$
sends $TX_{resP_i} \leftarrow \mathscr{V}.partyResponse(id_p, \texttt{enc}_{k_{ie}}(in_i))$
challengeParties
checks PoP_{resP} and reads TX_{resP}
punishParties
if $\boldsymbol{P}'_M \subset \boldsymbol{P}_M$ still fail to submit
send $TX_{pnsP} \leftarrow \mathscr{V}.punishParties(id_p, \boldsymbol{P}_M)$
↑ 2
Delivery phase ($Proc_{\mathrm{fdel}}$)
commit
$\boldsymbol{s}', \boldsymbol{r} \leftarrow f_{\mathscr{P}}(\boldsymbol{s}, \boldsymbol{x})$
generate $\boldsymbol{k}_{s'}, \boldsymbol{k}_r$ and $e_k \leftarrow \texttt{enc}_{k_{\mathcal{E}}}(\boldsymbol{k}_{s'}, \boldsymbol{k}_r)$
generate $c_{x_i} \leftarrow [\texttt{enc}_{k_{x_i}}(x_i), \texttt{enc}_{k_{ie}}(k_{x_i}), P_i]$
generate $c^*_{s'_i} \leftarrow [\texttt{enc}_{k_{s'_i}}(s'_i), 0, P_i]$
generate $c^*_{r_i} \leftarrow [\texttt{enc}_{k_{r_i}}(r_i), 0, P_i]$
generate $proof \leftarrow [H_{\boldsymbol{c}_s}]$
send $TX_{cmt} \leftarrow \mathscr{V}.commit(id_p, proof, \boldsymbol{c}_x, \boldsymbol{c}^*_{s'}, \boldsymbol{c}^*_r, e_k)$
↑ 3.1
read PoP_{cmt}, decrypt e_k
complete
send $TX_{com} \leftarrow$
$\mathscr{V}.complete(id_p, [\texttt{enc}_{k_{ie}}(k_{s'_i})|_{i\in[n]}], [\texttt{enc}_{k_{ie}}(k_{r_i})|_{i\in[n]}])$
broadcast TX_{com}
$h_{neg} + \tau_{com}$
sends $TX_{pnsT} \leftarrow \mathscr{V}.punishTEE(id_p)$
↑ 3.2

Figure 2. **The DeCloak protocol π_{DeCloak}.** $DN^{f,\mathscr{P}}$ denotes a DeCloak Network in which all executors hold TEEs with deployed $f, \mathscr{P}$. $BC^{\mathscr{V}}$ denotes a blockchain with deployed DeCloak contract $\mathscr{V}$. $Proc_{\mathrm{nneg}}$ and $Proc_{\mathrm{fdel}}$ denote the nondeterministic negotiation, and Δ-fair delivery subprotocols, respectively. Double dashed arrows denote reading BC and double arrows denote writing BC. Orange arrows denote challenge-response messages. Other arrows denote off-chain communications in secure channels. Specifically, messages sent by parties are signed by parties and encrypted by k_{ie} of DN, where $k_{ie} \leftarrow \texttt{ecdh}(sk_i, pk_{\mathcal{E}})$. All messages broadcast by DN are plaintext in default and signed by $sk_{\mathcal{E}}$. For simplicity, we omit to mark out the ciphertext of messages that parties are sending to DN, but mark the ciphertext explicitly in each transaction sent to BC.

C. Execution Phase

In this phase, $\mathcal{E}^*$ collects plaintext inputs from parties and executes f to obtain outputs inside TEE.

2: Upon receiving (id_p, p'), each party P_i knowing they are involved in the settled proposal p' feeds their inputs (*i.e.*, parameters x_i) to $\mathcal{E}^*$. The $\mathcal{E}^*$ keeps collecting parties' inputs and reads old states $\boldsymbol{s}$ from BC. If all involved parties' inputs are collected, $\mathcal{E}^*$ executes $f(\boldsymbol{s},\boldsymbol{x})$ to obtain the MPT outputs (*i.e.*, return values $\boldsymbol{r}$ and new states $\boldsymbol{s}'$) inside. Then, $\mathcal{E}^*$ goes to the step ***3.1***.

Otherwise, if some parties do not submit their inputs as expected, $Proc_{\text{rcha}}$ will identify them and punish them. We defer the detail in section V-E.

D. Delivery Phase

This phase adopts an Δ-*fair delivery subprotocol* ($Proc_{\text{fdel}}$) to reveal the plaintext outputs (*i.e.*, s'_i, r_i) to corresponding parties in a Δ-bounded period. $Proc_{\text{fdel}}$ proceeds in two steps.

3.1 $\mathcal{E}^*$ generates two arrays of symmetric keys $\boldsymbol{k}_{s'}, \boldsymbol{k}_r$ to compute the commitments of new states and return values s'_i, r_i, *i.e.*, $c_{s'_i}, c_{r_i}$, and generates a $proof \leftarrow [H_{\boldsymbol{c}_s}]$. The transaction with $proof$ signed by $\mathcal{E}^*$ can prove the MPT-caused state transition. Then, $\mathcal{E}^*$ sends a *commit* transaction $TX_{cmt} \leftarrow \mathcal{V}.\texttt{commit}(id_p, proof, \boldsymbol{c}^*_{s'}, \boldsymbol{c}^*_r, e_k)$ to commit the outputs on-chain. We note that the published $\boldsymbol{c}^*_{s'}, \boldsymbol{c}^*_r$ do not include the ciphertext of $\boldsymbol{k}_s, \boldsymbol{k}_r$ so that parties cannot reveal the commitments of $\boldsymbol{s}', \boldsymbol{r}$. Instead, $\mathcal{E}^*$ encrypts the keys with $k_{\boldsymbol{\mathcal{E}}}$, where $k_{\boldsymbol{\mathcal{E}}} \leftarrow \texttt{ecdh}(sk_{\boldsymbol{\mathcal{E}}}, pk_{\boldsymbol{\mathcal{E}}})$, and attaches the obtained ciphertext $e_k \leftarrow \texttt{enc}_{k_{\boldsymbol{\mathcal{E}}}}(\boldsymbol{k}_{s'}, \boldsymbol{k}_r)$ in TX_{cmt}. So when TX_{cmt} is confirmed, each $\mathcal{E} \in \boldsymbol{\mathcal{E}}$ can read $\boldsymbol{k}_{s'}, \boldsymbol{k}_r$ on-chain without interacting with others. Moreover, the $proof$ in TX_{cmt} proves the validity of state transition caused by the MPT f. $\mathcal{V}$ will validate the $proof$ and lock the on-chain states corresponding to old and new states, which signals the acceptance of the state transition and prevents its corresponding on-chain states from being updated by other concurrent MPTs before this MPT completes.

3.2: When TX_{cmt} becomes confirmed on-chain, each $E \in \boldsymbol{E}$ feeds the PoP_{cmt} (The PoP of the transaction TX_{cmt} which is an enough long and timely block sequence that contains TX_{cmt} to prove TX_{cmt} has been finalized) of TX_{cmt} to its $\mathcal{E}$. Each $\mathcal{E}$ reads key array $\boldsymbol{k}_{s'}, \boldsymbol{k}_r$ from the TX_{cmt}, then sends an transaction $TX_{com} = \mathcal{V}.\texttt{complete}(id_p, [\texttt{enc}_{k_{ie}}(k_{s'_i})], [\texttt{enc}_{k_{ie}}(k_{r_i})])$ to add the ciphertext of $\boldsymbol{k}_{s'}, \boldsymbol{k}_r$ to $\boldsymbol{c}^*_{s'}, \boldsymbol{c}^*_r$. The TX_{com} signals the COMPLETED of this MPT.

Here, the delivery fairness is achieved as follows: In ***3.1***, each party P_i has received the incomplete output commitments $\boldsymbol{c}^*_{s'}, \boldsymbol{c}^*_r$ but cannot decrypt them without corresponding $k_{s'_i}, k_{r_i}$. In ***3.2***, each $\mathcal{E}$ first verifies PoP_{cmt} to ensure that MPT outputs have been committed on BC. Then, each $\mathcal{E}$ sends a TX_{com} to complete the protocol with COMPETED. Since parties can directly communicate with all $\boldsymbol{E}$ to obtain TX_{com}, they can obtain the $\boldsymbol{k}_s, \boldsymbol{k}_r$ within the network latency Δ, as long as at least one $E \in \boldsymbol{E}$ honestly respond with TX_{com}. Otherwise, if TX_{cmt} is rejected by $\mathcal{V}$, any E cannot feed valid PoP_{cmt} to its TEE $\mathcal{E}$, implying that no TEE can release TX_{com} to reveal the plaintext outputs or complete the MPT before $h_{neg} + \tau_{com}$-th block. Therefore, DECLOAK guarantees the Δ-fairness of delivery, where Δ is the network latency of the blockchain.

E. Challenge-response Subprotocol

If in any phase one of the honest parties did not receive TEE's protocol messages as expected, the party can initiate an *challenge-response subprotocol* $Proc_{\text{rcha}}$. Specifically, it sends a *challengeTEE* transaction TX_{chaT} to challenge the specified TEE $\mathcal{E}^*$ on-chain publicly. The $\mathcal{E}^*$ being challenged can only avoid being punished by successfully responding with one of the following transactions:

- (i) TX_{fneg}: If the h_{neg}-th block has not been produced, the TEE $\mathcal{E}^*$ should keep collecting $\boldsymbol{ack}$, which parties from off-chain channels send, and $\boldsymbol{TX}_{ack}$, which are sent by parties to the blockchain and accepted before the h_{neg}-th block. We note that the off-chain ack_i will be preferred to $\boldsymbol{TX}_{ack}$. This ensures that any party cannot benefit from resending a sent off-chain acknowledgement on-chain, resending a new acknowledgement on-chain different from its sent off-chain acknowledge, or sending an acknowledgement on-chain only. This effect guarantees that the default sending acknowledgements off-chain is the best choice, no matter whether a party is corrupted or not. If collected acknowledgements cannot satisfy the settlement condition, $\mathcal{E}^*$ is allowed to send a TX_{fneg} to fail p on-chain, which finishes the MPT as NEGOFAILED. In all other cases where the h_{neg}-th block has not been confirmed or the $\mathcal{E}^*$ has successfully settled the proposal, $\mathcal{E}^*$ can't release a TX_{fneg}.
- (ii) TX_{com}: If the negotiation phase succeeds and the MPT completes, a TX_{com} will be sent by one of $\mathcal{E} \in \boldsymbol{\mathcal{E}}$ to BC inherently. TX_{com} finishes the MPT as COMPLETED.
- (iii) TX_{pnsP}: If the negotiation phase succeeds but the $\mathcal{E}^*$ cannot complete the MPT as expected, both parties and $\mathcal{E}^*$'s executor E^* can be misbehaved entities. Therefore, to avoid being punished in default, E^* should call its $\mathcal{E}^*$ to challenge parties publicly. Specifically, if $\mathcal{E}^*$ does not receive some parties' inputs, $\mathcal{E}^*$ marks these parties as suspicious parties $\boldsymbol{P}^*_M$ and returns $\boldsymbol{P}^*_M$ to its host E^*. The E^* calls $\mathcal{E}^*.\texttt{challengeParties}$ to send a TX_{chaP} to challenge $\boldsymbol{P}^*_M$ on-chain. Honest parties in $\boldsymbol{P}^*_M$ are supposed to send a TX_{resP} to publish the ciphertext of their inputs. All published TX_{resP} are required to be confirmed before block height $h_{neg} + \tau_{resP}$. Otherwise, the late TX_{resP} will be regarded as invalid by $\mathcal{E}^*$. Upon the confirmation of the $h_{neg} + \tau_{resP}$-th block, $\mathcal{E}^*$ reads the PoP_{resP} of all $\boldsymbol{TX}_{resP}$. If $\mathcal{E}^*$ successfully reads inputs of a party $P_i \in \boldsymbol{P}^*_M$ from its TX_{resP_i}, it removes P_i from $\boldsymbol{P}^*_M$. Otherwise, if PoP_{resP} shows that no TX_{resP_i} is published on-chain or the inputs in TX_{resP_i} are still invalid, $\mathcal{E}^*$ retains P_i in $\boldsymbol{P}^*_M$. After that, if $\boldsymbol{P}^*_M$ becomes empty, meaning all inputs are collected, $\mathcal{E}^*$ goes to the step ***2***. Otherwise, if $\boldsymbol{P}^*_M$ remains non-empty, which means the misbehaviour of parties left is confirmed, $\mathcal{E}^*$ marks these parties as $\boldsymbol{P}_M$. Then, $\mathcal{E}^*$ sends a TX_{pnsP}. TX_{pnsP} calls $\texttt{punishParties}$ to punish $\boldsymbol{P}_M$ in finance and signal the MPT with ABORTED.

If $\mathcal{E}^*$ being challenged by a party either fails (by TX_{fneg}), stops (by TX_{pnsP}), or completes (by TX_{com}) the MPT, anyone can send a TX_{pnsT} after the $h_{neg} + \tau_{com}$-th block to punish $\mathcal{E}^*$ and signal the MPT with ABORTED.

VI. Implementation

While the design of π_{DeCloak} is chain-agnostic and TEE-agnostic, here we introduce how we instantiate and prototype DeCloak.

A. DeCloak *Contract*

We instantiate *BC* as Ethereum. Specifically, we implement the DeCloak contract ($\mathscr{V}$), as shown in Algorithm 1, in Solidity 0.8.10 [42]. $\mathscr{V}$ is constructed by the config of *DN*, *e.g.*, $pk_{\boldsymbol{\mathcal{E}}}, ad_{\boldsymbol{\mathcal{E}}}$, so that parties can authenticate and build secure channels with $\boldsymbol{\mathcal{E}}$. Moreover, $\mathscr{V}$ provides functions to manage the life cycle of each MPT. A party calls $\mathscr{V}$`.challengeTEE` by TX_{chaT} to challenge the specified TEE, and signal the negotiation as NEGOTIATED. When an MPT was evaluated, the specified TEE $\mathscr{E}^*$ calls $\mathscr{V}$`.commit` by TX_{cmt} to validate and commit the outputs. Finally, a $\mathscr{E}$ calls $\mathscr{V}$`.complete` by TX_{com} to release keys' ciphertext and signal the MPT as COMPLETED.

B. DeCloak *Network*

To construct *DN*, we instantiate each TEE $\mathscr{E}$ (Algorithm 2) based on SGX [23]. Anyone with an SGX can instantiate and register a $\mathscr{E}$ to become an executor *E*. The first registered $\mathscr{E}$ generates the network account $(sk_{\boldsymbol{\mathcal{E}}}, pk_{\boldsymbol{\mathcal{E}}}, ad_{\boldsymbol{\mathcal{E}}})$ to initialize a network *DN*. Then, other $\mathscr{E}$ must be attested by one of $\boldsymbol{\mathcal{E}}$ in *DN* to obtain the network account and join *DN*.

We express the program of an MPT f in Solidity 0.8.10 [42] and port EVM [43] into SGX. $\mathscr{P}$ is expressed in JSON. It specifies the settlement condition of the MPT and identifies the parameters, states to read and write, and return values of f, which is for TEE to know the I/O of the MPT. The hash of both f and $\mathscr{P}$ are registered and updated on *BC*, while their codes are provided by the MPT's developers/initiators and cached by $\boldsymbol{\mathcal{E}}$. Admittedly, $\mathscr{P}$ restricts the I/O of f to be statically identified. However, this problem can be solved by hooking EVM's `sstore` and `sload` instructions [44], and we leave it for future work.

VII. Security Analysis

A. *Protocol Security*

Here we formalize and prove our system goals. Recall $\boldsymbol{P}$, $\boldsymbol{E}$, and $\boldsymbol{\mathcal{E}}$ denote the party array, executor array and TEE array of an MPT, respectively. We have $|\boldsymbol{P}| = n$ and $|\boldsymbol{E}| = |\boldsymbol{\mathcal{E}}| = m$. We define $\boldsymbol{P}_H$ and $\boldsymbol{E}_H$ as the honest entities in $\boldsymbol{P}$ and $\boldsymbol{E}$, respectively. $\boldsymbol{P}_M$ and $\boldsymbol{E}_M$ denote the malicious entities in $\boldsymbol{P}$ and $\boldsymbol{E}$, *i.e.*, $\boldsymbol{P}_M \leftarrow \boldsymbol{P} \backslash \boldsymbol{P}_H$, $\boldsymbol{E}_M \leftarrow \boldsymbol{E} \backslash \boldsymbol{E}_H$. For convenience, we also define $\boldsymbol{P}^+ \leftarrow \boldsymbol{P} \cup \boldsymbol{E}$ and $\boldsymbol{P}_M^+ \leftarrow \boldsymbol{P}_M \cup \boldsymbol{E}_M$. Recall our adversary model in Section IV, the Byzantine adversary $\mathscr{A}$ can corrupt all $P_i \in \boldsymbol{P}_M$ and $E_i \in \boldsymbol{E}_M$ where $|\boldsymbol{P}_M| < n$ and $|\boldsymbol{E}_M| < m$.

For DeCloak protocol π_{DeCloak}, or simply π, we classically define the execution of π under the adversary $\mathscr{A}$ as $REAL_{\pi,\mathscr{A}}$. And it's formalized as follows. The inputs of an execution include an n-party MPT f and its policy $\mathscr{P}$, a parameter array $\boldsymbol{x}$, a parameter key array $\boldsymbol{k}_x$, an old state commitment array $\boldsymbol{c}_s$, a party array $\boldsymbol{P}$, a TEE array $\boldsymbol{\mathcal{E}}$, a deposit array $\boldsymbol{q}$ and a coin balance array $\boldsymbol{Q}$ ($|\boldsymbol{Q}| = n+m$. $Q_i|_{i<n}$ denotes the coin balance of $P_i \in \boldsymbol{P}$ pre-deposited to $ad_{\boldsymbol{\mathcal{E}}}$. $Q_{n+i}|_{i<m}$ denotes the pre-deposited coin balance of $\mathscr{E}_i \in \boldsymbol{\mathcal{E}}$). The outputs of π include a new coin balance array $\boldsymbol{Q}'$ after the execution, new state array $\boldsymbol{s}'$, new state key array $\boldsymbol{k}_{s'}$, return value array $\boldsymbol{r}$, return value key array $\boldsymbol{k}_r$, and the commitment array of new states $\boldsymbol{c}_{s'}$, return values $\boldsymbol{c}_{s'}$, and parameters $\boldsymbol{c}_x$, and $proof$ of the MPT-caused state transition.

Algorithm 1: DeCloak contract ($\mathscr{V}$)

```
// This contract is constructed by the network
   config $pk_{\boldsymbol{\mathcal{E}}}, ad_{\boldsymbol{\mathcal{E}}}$ and a TEE list $\boldsymbol{\mathcal{E}}$. $ad_{\boldsymbol{\mathcal{E}}}$ is the
   network account for managing coins deposited
   by parties. For simplicity, we ignore the
   register and deposit functions here.
Function challengeTEE($p$)
    // called by $TX_{chaT}$ from one of parties
    $id_p \leftarrow \mathtt{hash}(p)$
    require($prsls[id_p] = \emptyset$)
    $prsls[id_p].\{q, h_{neg}, \tau_{com}, \mathscr{E}\} \leftarrow p.\{q, h_{neg}\}, \tau_{com}, \boldsymbol{\mathcal{E}}[0]$
    $prsls[id_p].sta \leftarrow$ PROPOSED
Function acknowledge($id_p, \mathtt{enc}_{k_{\boldsymbol{\mathcal{E}}}}(ack_i)$)
    // called by $TX_{ack_i}$ from parties
    require($BC.\mathtt{getHeight}() < h_{neg}$)
Function failNegotiation($id_p$)
    // called by $TX_{fneg}$ from the specified TEE
    require($msg.sender = prsls[id_p].\mathscr{E}$)
    $prsls[id_p].sta \leftarrow$ NEGOFAILED
Function challengeParties($id_p, \boldsymbol{P}_M^*$)
    // called by $TX_{chaP}$ from the specified TEE
Function partyResponse($id_p, \mathtt{enc}_{k_{\boldsymbol{\mathcal{E}}}}(in)$)
    // called by $TX_{resP_i}$ from parties
    require($BC.\mathtt{getHeight}() < h_{neg} + \tau_{resP}$)
Function punishParties($id_p, \boldsymbol{P}_M$)
    // called by $TX_{pnsP}$ from the specified TEE
    require($msg.sender = prsls[id_p].\mathscr{E}$)
    // update coins for punishment
    for $P_i \in \boldsymbol{P}_M$ do
        $coins[P_i] \leftarrow coins[P_i] - q$
    $prsls[id_p].sta \leftarrow$ ABORTED
Function commit($id_p, proof, \boldsymbol{c}_x, \boldsymbol{c}_{s'}^*, \boldsymbol{c}_r^*, e_k$)
    // called by $TX_{cmt}$ from the specified TEE
    require($msg.sender = prsls[id_p].\mathscr{E}$)
    require(verify($proof$, $H_{\boldsymbol{c}_s}$)) // match old states
Function complete($id_p, [\mathtt{enc}_{k_{ie}}(k_{s'_i})|_{i\in[n]}], [\mathtt{enc}_{k_{ie}}(k_{r_i})|_{i\in[n]}]$)
    // called by $TX_{com}$ from any registered TEE
    require($msg.sender \in \boldsymbol{\mathcal{E}}$)
    $H_{\boldsymbol{c}_s} \leftarrow proof.H_{\boldsymbol{c}_{s'}}$ // set new states
    $prsls[id_p].sta \leftarrow$ COMPLETED
Function punishTEE($id_p$)
    // called by $TX_{pnsT}$ from anyone
    require($prsls[id_p] \neq \emptyset$ and $BC.\mathtt{getHeight}() > h_{neg} + \tau_{com}$)
    require($prsls[id_p].sta \notin$
      {NEGOFAILED,ABORTED,COMPLETED})
    $coins[prsls[id_p].\mathscr{E}] \leftarrow coins[prsls[id_p].\mathscr{E}] - q$
    $prsls[id_p].sta \leftarrow$ ABORTED
```

$$\boldsymbol{Q}', \boldsymbol{s}', \boldsymbol{k}_{s'}, \boldsymbol{r}, \boldsymbol{k}_r, \boldsymbol{c}_{s'}, \boldsymbol{c}_r, \boldsymbol{c}_x, proof, sta \leftarrow REAL_{\pi,\mathscr{A}}(\boldsymbol{Q}, f, \boldsymbol{x}, \boldsymbol{k}_x, \boldsymbol{c}_s, \boldsymbol{P}, \boldsymbol{\mathcal{E}}, \boldsymbol{q})$$

1) *Security definitions*

We formalize our security goals as follows.

Definition 1 (Correctness)**.** *There is a negligible function ε that for any output of $REAL_{\pi,\mathscr{A}}(\boldsymbol{Q}, f, \boldsymbol{x}, \boldsymbol{k}_x, \boldsymbol{c}_s, \boldsymbol{P}, \boldsymbol{\mathcal{E}}, \boldsymbol{q})$ where $sta =$ COMPLETED*

$$\left| Pr\left[(\boldsymbol{s}', \boldsymbol{r}, \boldsymbol{c}_{s'}, \boldsymbol{c}_r, \boldsymbol{c}_x, proof) = \mathtt{mpt}(f, \boldsymbol{x}, \boldsymbol{c}_s, \boldsymbol{P}) \right] - 1 \right| \leq \varepsilon$$

Note that `mpt` *in Algorithm 3 contains the correctness requirement of an MPT.*

Algorithm 2: DeCloak enclave program ($\mathscr{E}$)

```
// Each $\mathscr{E}$ has obtained the network config and
   cached its and parties' coin balances by
   synchronization. The config includes a
   secure parameter $\kappa$, a checkpoint $b_{cp}$ of $BC$,
   and the network account $(sk_{\boldsymbol{\mathcal{E}}}, pk_{\boldsymbol{\mathcal{E}}}, ad_{\boldsymbol{\mathcal{E}}})$.
Procedure generateIDp($p$)
    // check this is the specified TEE
    if $self \neq BC.\boldsymbol{\mathcal{E}}[0]$ or $cacheCoins.\texttt{lock}(self,\ q) \neq 1$ then abort
    $id_p \leftarrow \texttt{hash}(p)$
    return $(id_p, p)$
Procedure negotiate($id_p$,$\boldsymbol{ack}$)
    if $status = \texttt{NEGOTIATED}$ then return $(id_p, p')$
    if $status \neq \emptyset$ or $\texttt{conform}(\boldsymbol{ack}, \mathscr{P}) \neq 1$
       or $cacheCoins.\texttt{lock}(\boldsymbol{P},\ q) \neq 1$ then abort
    $p', status \leftarrow (p.\{H_f, H_{\mathscr{P}}, q, h_{neg}\})$, NEGOTIATED
    return $(id_p, p')$
Procedure failNegotiation($id_p$,$TX_{chaT}$,$PoP_{chaT}$)
    if $status \neq \emptyset$ or $\texttt{veriPoP}(b_{cp}, PoP_{chaT}, TX_{chaT}) \neq 1$ then abort
    if $PoP_{chaT}.\texttt{getComfHeight}() > p.h_{neg}$ then
        $\boldsymbol{TX}_{ack} \leftarrow$ all $PoP_{chaT}.TX_{ack_i}$ before $p.h_{neg}$
        $\boldsymbol{ack} \leftarrow \boldsymbol{ack} \cup \boldsymbol{TX}_{ack}.\boldsymbol{ack}$
    if $\texttt{conform}(\boldsymbol{ack}, \mathscr{P}) = 1$ then abort
    $cacheCoins.\texttt{unlock}(self,\ q)$
    return $\boldsymbol{TX}_{fneg}(id_p)$
Procedure execute($id_p$,$\boldsymbol{in}$,$PoP_s$)
    if $status \neq \texttt{NEGOTIATED}$ then abort
    $\boldsymbol{P}^*_M \leftarrow \boldsymbol{P}$
    for $x_i, k_{x_i}$ in $\boldsymbol{in}.\{\boldsymbol{x},\ \boldsymbol{k}_x\}$
        $\boldsymbol{P}^*_M \leftarrow \boldsymbol{P}^*_M \backslash \{P_i\}$
    if $|\boldsymbol{P}^*_M| > 0$ then return $(id_p, \boldsymbol{P}^*_M)$
    // evaluates $f(x)$ on states $s$
    $\boldsymbol{s}', \boldsymbol{r} \leftarrow f(PoP_s.\boldsymbol{s}, \boldsymbol{x})$
    $b_{cp} \leftarrow PoP_s.\texttt{getLastComfBlock}()$
    $status \leftarrow \texttt{EXECUTED}$
Procedure commit($id_p$)
    if $status \neq \texttt{EXECUTED}$ then abort
    $\boldsymbol{k}_{s'}, \boldsymbol{k}_r \leftarrow Gen(1^\kappa)$
    $e_k \leftarrow \texttt{enc}_{k_{\boldsymbol{\mathcal{E}}}}(\boldsymbol{k}_{s'}, \boldsymbol{k}_r)$
    $proof \leftarrow [H_{PoP_s.\boldsymbol{c}_s}]$
    $c_{x_i} \leftarrow [\texttt{enc}_{k_{x_i}}(x_i), \texttt{enc}_{k_{ie}}(k_{x_i}), P_i]$
    $c^*_{s'_i}, c^*_{r_i} \leftarrow [\texttt{enc}_{k_{s'_i}}(s'_i), 0, P_i],\ [\texttt{enc}_{k_{r_i}}(r_i), 0, P_i]$
    return $TX_{cmt}(id_p, proof, \boldsymbol{c}_x, \boldsymbol{c}^*_{s'}, \boldsymbol{c}^*_r, e_k)$
Procedure challengeParties($\boldsymbol{P}^*_M$)
    if $status \neq \texttt{NEGOTIATED}$ then abort
    if $|\boldsymbol{P}^*_M| > 0$ then
        return $\boldsymbol{TX}_{chaP}(id_p, \boldsymbol{P}^*_M)$
Procedure punishParties($\boldsymbol{TX}_{chaP}$,$\boldsymbol{TX}_{resP}$,$PoP_{resP}$)
    if $status \neq \texttt{NEGOTIATED}$ or
     $\texttt{veriPoP}(b_{cp}, \boldsymbol{TX}_{chaP}, PoP_{resP}) \neq 1$ then abort
    $\boldsymbol{P}_M \leftarrow \boldsymbol{P}^*_M$
    for $P_i \in \boldsymbol{P}^*_M$ do
        if $x_i, k_{x_i} \leftarrow TX_{resP_i}.\{x_i, k_{x_i}\}$ then
            $\boldsymbol{P}_M \leftarrow \boldsymbol{P}_M \backslash \{P_i\}$
    if $|\boldsymbol{P}_M| > 0$ then
        return $TX_{pnsP}(id_p, \boldsymbol{P}_M)$
Procedure complete($TX_{cmt}$,$PoP_{cmt}$)
    if $status \neq \texttt{NEGOTIATED}$ or $\texttt{veriPoP}(b_{cp}, TX_{cmt}, PoP_{cmt}) \neq 1$
     then abort
    $status \leftarrow \texttt{COMPLETED}$
    $cacheCoins.\texttt{unlock}(\boldsymbol{P} \cup self,\ q)$
    return $TX_{com}(id_p, [\texttt{enc}_{k_{ie}}(k_{s'_i})|_{i \in [n]}],\ [\texttt{enc}_{k_{ie}}(k_{r_i})|_{i \in [n]}])$
```

Algorithm 3: MPT evaluation function

```
Input: An $n$-party MPT $f$, a parameter array $\boldsymbol{x}$, an old state
       commitment array $\boldsymbol{c}_s$, and a party array $\boldsymbol{P}$.
Output: A new state array $\boldsymbol{s}'$, return value array $\boldsymbol{r}$, new state
        commitment array $\boldsymbol{c}_{s'}$, return value commitment
        array $\boldsymbol{c}_r$, parameter commitment array $\boldsymbol{c}_x$, and a
        $proof$.
Function mpt($f$,$\boldsymbol{x}$,$\boldsymbol{c}_s$,$\boldsymbol{P}$)
    foreach $c_{s_i}$ in $\boldsymbol{c}_s$
        assert $c_{s_i} = [\texttt{enc}_{k_{s_i}}(s_i), \texttt{enc}_{k_{ie}}(k_{s_i}), P_i]$
    $\boldsymbol{s}', \boldsymbol{r} \leftarrow f(\boldsymbol{s}, \boldsymbol{x})$
    $\boldsymbol{k}_{s'}, \boldsymbol{k}_r \leftarrow Gen(1^\kappa)$
    $c_{s'_i} \leftarrow [\texttt{enc}_{k_{s'_i}}(s'_i), \texttt{enc}_{k_{ie}}(k_{s'_i}), P_i]$
    $c_{r_i} \leftarrow [\texttt{enc}_{k_{r_i}}(r_i), \texttt{enc}_{k_{ie}}(k_{r_i}), P_i]$
    $c_{x_i} \leftarrow [\texttt{enc}_{k_{x_i}}(x_i), \texttt{enc}_{k_{ie}}(k_{x_i}), P_i]$
    $proof \leftarrow [H_{\boldsymbol{c}_s}]$
    return $(\boldsymbol{s}', \boldsymbol{r}, \boldsymbol{c}_{s'}, \boldsymbol{c}_r, \boldsymbol{c}_x, proof)$
```

Definition 2 (Confidentiality). *For any adversary $\mathscr{A}$ corrupting $\boldsymbol{P}^+_M$, the output of any $REAL_{\pi,\mathscr{A}}(\boldsymbol{Q}, f, \boldsymbol{x}, \boldsymbol{k}_x, \boldsymbol{c}_s, \boldsymbol{P}, \boldsymbol{\mathcal{E}}, \boldsymbol{q})$ is such that: There is a negligible function ε ensuring that $\forall x^*_1, s^*_1, s'^*_1, r^*_1, x^*_2, s^*_2, s'^*_2, r^*_2$ and $\forall P_i \in \boldsymbol{P}_H$:*

$$|Pr[x_i, s_i = x^*_1, s^*_1] - Pr[x_i, s_i = x^*_2, s^*_2]| \ \leq \ \varepsilon$$

$$and$$

$$\left| Pr[s'_i, r_i = s'^*_1, r^*_1] - Pr[s'_i, r_i = s'^*_2, r^*_2] \right| \ \leq \ \varepsilon$$

Definition 3 (Data availability). *For any adversary $\mathscr{A}$ corrupting $\boldsymbol{P}^+_M$, the output of any $REAL_{\pi,\mathscr{A}}(\boldsymbol{Q}, f, \boldsymbol{x}, \boldsymbol{k}_x, \boldsymbol{c}_s, \boldsymbol{P}, \boldsymbol{\mathcal{E}}, \boldsymbol{q})$ is such that: There is a negligible function ε satisfies that if $sta = \texttt{COMPLETED}$, all following statements must be true.*

(a) $\forall E_i \in \boldsymbol{E}_H$, there is a polynomial function $f_{\mathscr{E}_i}$ for its TEE that $s'_i = f_{\mathscr{E}_i}(sk_{\boldsymbol{\mathcal{E}}}, P_i, c_{s'_i})$

(b) $\forall P_i \in \boldsymbol{P}_H$, there is a polynomial function f_{P_i} that $s'_i = f_{P_i}(sk_i, pk_{\boldsymbol{\mathcal{E}}}, c_{s'_i})$

Definition 4 (Financial fairness). *For any adversary $\mathscr{A}$ corrupting entities from $\boldsymbol{P}^+_M$ in which $\boldsymbol{P}_M \subsetneqq \boldsymbol{P}$, the output of any $REAL_{\pi,\mathscr{A}}(\boldsymbol{Q}, f, \boldsymbol{x}, \boldsymbol{k}_x, \boldsymbol{c}_s, \boldsymbol{P}, \boldsymbol{\mathcal{E}}, \boldsymbol{q})$ is such that one of the following statements must be true:*

(i) $sta \in \{\texttt{NEGOFAILED}, \texttt{COMPLETED}\}$, $\forall P_i \in \boldsymbol{P}^+ : Q'_i \geq Q_i$

(ii) $sta = \texttt{ABORTED}$, $\forall P_i \in \boldsymbol{P}^+_H :\ Q'_i \geq Q_i$ and

$$\sum_{j \in \boldsymbol{P}^+_M} Q'_j < \sum_{j \in \boldsymbol{P}^+_M} Q_j$$

Definition 5 (Delivery fairness). *For any adversary $\mathscr{A}$ corrupting $\boldsymbol{P}^+_M$ where $\boldsymbol{E}_M \subsetneqq \boldsymbol{E}$, there is a negligible function ε that for any $REAL_{\pi,\mathscr{A}}(\boldsymbol{Q}, f, \boldsymbol{x}, \boldsymbol{k}_x, \boldsymbol{c}_s, \boldsymbol{P}, \boldsymbol{\mathcal{E}}, \boldsymbol{q})$, one of the following statements must be true:*

(i) $\boldsymbol{s}', \boldsymbol{r} = \emptyset, \emptyset$

(ii) $\boldsymbol{s}', \boldsymbol{r}, \neq \emptyset, \emptyset$, and the following two hold simultaneously :

(a) $\forall P_i \in \boldsymbol{P}_H : \left| t_{s'_i} - t_{r_i} \right| \leq \Delta$

(b) $\forall P_i, P_j \in \boldsymbol{P}_H : \left| t_{s'_i} - t_{s'_j} \right| \leq \Delta$ and $\left| t_{r_i} - t_{r_j} \right| \leq \Delta$

Definition 6 (Delivery atomicity). *For any adversary $\mathscr{A}$ corrupting $\boldsymbol{P}^+_M$ where $\boldsymbol{E}_M \subsetneqq \boldsymbol{E}$, there is a negligible function ε that for any $REAL_{\pi,\mathscr{A}}(\boldsymbol{Q}, f, \boldsymbol{x}, \boldsymbol{k}_x, \boldsymbol{c}_s, \boldsymbol{P}, \boldsymbol{\mathcal{E}}, \boldsymbol{q})$, one of the following statements must be true:*

(i) $sta \in \{\texttt{NEGOFAILED}, \texttt{ABORTED}\}$, and $\boldsymbol{c}_{s'}, \boldsymbol{s}', \boldsymbol{r} = \emptyset, \emptyset, \emptyset$

(ii) $sta = \texttt{COMPLETED}$, and $\boldsymbol{c}_{s'} \neq \emptyset$, $\boldsymbol{s}' \neq \emptyset$, and $\boldsymbol{r} \neq \emptyset$

2) *Security proof*

We claim that the following theorem holds. As the proof of correctness, confidentiality, and public verifiability are native and intuitive, here we seek to prove data availability, financial fairness, delivery fairness, and delivery atomicity.

Theorem 1 (Formal statement)**.** *Assume a EUF-CMA secure signature scheme, a IND-CCA2 encryption scheme, and a hash function that is collision-resistant, preimage and second-preimage resistant, a TEE emulating the TEE ideal functionality and a blockchain satisfies common prefix, chain quality and chain growth, and has a PoP scheme, π_{DeCloak} holds* ***correctness****,* ***confidentiality****,* ***public verifiability****,* ***data availability****,* ***financial fairness****,* ***delivery fairness****, and* ***delivery atomicity****.*

Proof of data availability. According to Algorithm 1, when $sta = \texttt{COMPLETED}$, the new state commitments $\boldsymbol{c}_{s'}$ must be published on BC. Recall the commitment subprotocol $Proc_{dcmt}$ where each new state commitment is as follows. We construct a polynomial function in Algorithm 4. With the function, any $\mathscr{E}$ of $E \in \boldsymbol{E}_H$ can restore $c_{s'_i}$ by $\texttt{restoreStates}(sk_{\boldsymbol{\mathcal{E}}}, pk_i, c_{s'_i})$, hence satisfying (a), and any $P_i \in \boldsymbol{P}$ can construct $c_{s'_i}$ by $\texttt{restoreStates}(sk_i, pk_{\boldsymbol{\mathcal{E}}}, c_{s'_i})$, satisfying (b). Therefore, the data availability holds.

$$c_{s'_i} \leftarrow [\texttt{enc}_{k_{s'_i}}(s'_i), \texttt{enc}_{k_{ie}}(k_{s'_i}), P_i]$$

Algorithm 4: States restoration function

```
Function restoreStates(sk, pk, c_{s'_i})
    $k_{ie} \leftarrow \texttt{ecdh}(sk, pk)$
    $k_{s'_i} \leftarrow \texttt{dec}_{k_{ie}}(c_{s'_i}[1])$
    $s'_i \leftarrow \texttt{dec}_{k_{s'_i}}(c_{s'_i}[0])$
    return $s'_i$
```

Proof of financial fairness. Here we prove that in all possible final sta, the financial fairness of π_{DeCloak} holds. First, we consider the *Negotiation phase*.

Lemma 2. *If $sta = \texttt{NEGOFAILED}$, the statement (i) of the financial fairness property holds.*

Proof: There is only one case that $sta = \texttt{NEGOFAILED}$: TX_{fneg} is confirmed on BC after $Proc_{\text{nneg}}$. This happens when the collected $\boldsymbol{ack}$ from both on-chain and off-chain channels cannot satisfy the settlement condition of MPT proposal or $\exists P_i \in \boldsymbol{P} \cup \boldsymbol{\mathcal{E}}[0]$ holds that $\boldsymbol{Q}_i < q$. No matter what reasons cause the TX_{fneg}, the TX_{fneg} accepted by $\mathscr{V}$ will be irrevocable. As we assume that BC satisfies the common prefix, chain quality and chain growth, the confirmation of TX_{fneg} is also irrevocable. Consequently, since both $\boldsymbol{\mathcal{E}}[0]$ and TX_{fneg} do not change $\boldsymbol{Q}$, *i.e.*, the (i) $Q'_i \geq Q_i$ holds.

Lemma 3. *If $sta = \texttt{COMPLETED}$, then the statement (i) of the financial fairness property holds.*

Proof: According to Algorithm 1, the protocol outputs $sta = \texttt{COMPLETED}$ iff a transaction TX_{com} is contained on BC before the $h_{neg} + \tau_{com}$-th block. As TX_{com} does not update $\boldsymbol{Q}$, $\forall P_i \in \boldsymbol{P}^+$ the $Q'_i = Q_i \geq Q_i$ holds.

Next, we show that financial fairness also holds even if an MPT fails by $\texttt{ABORTED}$ after a successful *Negotiation phase*.

Lemma 4. *If $sta = \texttt{ABORTED}$, then the statement (ii) of the financial fairness property holds.*

Proof: There are two cases when $sta = \texttt{ABORTED}$:

- (i) Before the $h_{neg} + \tau_{com}$-th block, $TX_{pnsP}(id_p, \boldsymbol{P}_M)$ is published on BC, and later been confirmed.
- (ii) After the $h_{neg} + \tau_{com}$-th block, $TX_{pnsT}(id_p)$ is published on BC, and later been confirmed.

We first consider the case (i) where $\exists P_j \in \boldsymbol{P}$ does not provide inputs in_j after the negotiation succeeded. According to Algorithm 2, the $\mathscr{E}^*$ releases a transaction $TX_{pnsP}(id_p, \boldsymbol{P}_M)$ iff E^* calls the $\mathscr{E}^*$*.punishParties* with a PoP_{resP} which proves that $P_j \in \boldsymbol{P}_M|_{\boldsymbol{P}_M \neq \emptyset}$ did not provide their inputs even though they were challenged by a TX_{chaP}. The TX_{pnsP} deducts coins of $P_i \in \boldsymbol{P}_M$ by the collateral q_i. In other word, for $P_i \in \boldsymbol{P}_M$, it holds that $Q'_i = Q_i - q_i$. Since $Q_i > q_i$, which has been ensured by $Proc_{\text{nneg}}$, $\boldsymbol{P}_M \neq \emptyset$, and $\boldsymbol{P}_M \in \boldsymbol{P}_M^+$, it holds that $\sum_{j \in \boldsymbol{P}_M^+} Q'_j < \sum_{j \in \boldsymbol{P}_M^+} Q_j$. Notably, no malicious entities earned coins in this case.

Second, we consider the case (ii) which indicates that TX_{com} fails to be contained before the $h_{neg} + \tau_{com}$-th block. When the timeout transaction TX_{pnsT} is posted on BC, p' will be marked as $\texttt{ABORTED}$, and E^* will be punished by collateral q. As and $E^* \in \boldsymbol{P}_M^+$, it holds that $\sum_{j \in \boldsymbol{P}_M^+} Q'_j < \sum_{j \in \boldsymbol{P}_M^+} Q_j$.

Lemma 5. *When π_{DeCloak} terminates, it must hold $sta \in \{\texttt{NEGOFAILED}, \texttt{ABORTED}, \texttt{COMPLETED}\}$.*

Proof: As we assume $\mathscr{A}$ corrupts $\boldsymbol{P}_M^+$ where $\boldsymbol{P}_M \subsetneqq \boldsymbol{P}$, there must be $\boldsymbol{P}_H \neq \emptyset$. Say a $P^* \in \boldsymbol{P}$ sending a proposal p to $\mathscr{E}^*$. If both P^* and E^* are malicious, the E^* may not feed p into its $\mathscr{E}$ or not broadcast (id_p, p) to $\boldsymbol{P}_H$ after the Step ***1.1***, and P^* does not challenge it on BC. In this case, $P_i \in \boldsymbol{P}_H$ or BC is unaware of the existence of p, nor being involved, thus meaningless. If P^* or E^* is malicious, the honest E^* will broadcast p to all $P_i \in \boldsymbol{P}_H$ or the honest P^* will challenge with p on-chain. Either case above ensures that $\exists P_i \in \boldsymbol{P}_H$ will be aware of the p to start with π_{DeCloak}. Then, if all parties and executors are honest in the following phases, the MPT corresponding p must succeed, and a TX_{com} will be sent, which leads to $sta \leftarrow \texttt{COMPLETED}$. Otherwise, if any reason fails the TX_{com}, there must be $\exists P_i \in \boldsymbol{P}_H$ challenge $\mathscr{E}^*$ with p by TX_{chaT}, making BC also aware of p, *i.e.*, $sta \leftarrow \texttt{PROPOSED}$. Then, if the adversary stops misbehaving after TX_{chaT}, there must be a TX_{com} being released, leading to $sta \leftarrow \texttt{COMPLETED}$. Otherwise, as $\mathscr{V}$ authorizes anyone to punish E^* after the $h_{neg} + \tau_{com}$-th block, E^* must prove its honesty to avoid being punished. Specifically, E^* either feeds $\mathscr{E}^*$ the BC view to prove the failure of negotiation to release TX_{fneg} and set

$sta \leftarrow$ NEGOFAILED, or feeds $\mathscr{E}^*$ the BC view to prove that parties being challenged fail to respond to release TX_{pnsP} and set $sta \leftarrow$ ABORTED. If none of TX_{fneg}, TX_{pnsP}, or TX_{com} happens, there must be $\exists P_i \in \boldsymbol{P}_H$ set $sta \leftarrow$ ABORTED by TX_{pnsT}. Therefore, it holds $sta \in \{\texttt{NEGOFAILED}, \texttt{ABORTED}, \texttt{COMPLETED}\}$.

Proof of delivery fairness. As Lemma 5 holds, we prove that the *delivery fairness* holds in all three final values of sta.

Lemma 6. *If $sta =$ NEGOFAILED, then the statement (i) of the delivery fairness holds.*

Proof: As proved in Lemma 2, $sta =$ NEGOFAILED only when there is a TX_{fneg} being successfully confirmed on the BC after the *negotiation phase*. Consequently, $\mathscr{E}^*$ never proceeds with the *Execution phase*. Therefore, $\forall P_i \in \boldsymbol{P}$ obtain no outputs, *i.e.*, $\boldsymbol{s}', \boldsymbol{r} = \emptyset, \emptyset$.

Lemma 7. *If $sta =$ ABORTED, then the statement (i) of the delivery fairness holds.*

Proof: Recall $\mathscr{A}$ corrupts $\boldsymbol{P}_M^+$ where $\boldsymbol{E}_M \subsetneq \boldsymbol{E}$. If TX_{cmt} is released, there must be that $\exists \mathscr{E}_j \in \boldsymbol{\mathcal{E}}$ releases a TX_{com} which sets $sta =$ COMPLETED. However, according to the proof of Lemma 4, if $sta =$ ABORTED is true, π_{DECLOAK} is terminated by either TX_{pnsT} or TX_{pnsP}, meaning that TX_{cmt} is impossible to be released while releasing TX_{cmt} is the condition of releasing outputs. Therefore, it holds that $\boldsymbol{s}', \boldsymbol{r} = \emptyset, \emptyset$.

Lemma 8. *If $sta =$ COMPLETED, then statement (ii) of the delivery fairness holds.*

Proof: $sta =$ COMPLETED only when TX_{com} is released and accepted by BC, which requires that TX_{cmt} has been confirmed on BC. Recall that we assume $\mathscr{A}$ corrupts $\boldsymbol{P}_M^+$ where $\boldsymbol{E}_M \subsetneq \boldsymbol{E}$ and BC is ideally accessible. Say TX_{cmt} is confirmed on BC in a wall-time t_{com}, then the time of all honest entities in $\boldsymbol{P}^+$ knowing that TX_{cmt} has been confirmed is also t_{com}, *i.e.*, $t_i \leftarrow t_{com}|_{t_i \in \boldsymbol{t}^+_{com}}$. There must be $\exists \mathscr{E} \in \boldsymbol{\mathcal{E}}$ independently validate PoP_{cmt} of TX_{cmt} and read $\boldsymbol{k}_{s'}, \boldsymbol{k}_r$ from TX_{cmt} to construct and release a TX_{com}. As $P_i \in \boldsymbol{P}_H$ undisturbedly and independently obtains TX_{com} broadcasted by $E \in \boldsymbol{E}$ from BC network within the network latency Δ, then we conclude that $\boldsymbol{t}_{s'} = \boldsymbol{t}_r$ and $\forall P_i, P_j \in \boldsymbol{P}_H : \left|t_{s'_i} - t_{s'_j}\right| \leq \Delta$ and $\left|t_{r_i} - t_{r_j}\right| \leq \Delta$, *i.e.*, the (a) and (b) of (ii) are satisfied.

Proof of delivery atomicity. Recall that Lemma 5 enumerates all possible three final statuses of sta. Lemma 6 and 7 prove that the first two statuses hold $\boldsymbol{s}', \boldsymbol{r} = \emptyset, \emptyset$, making releasing TX_{com} that sets $\boldsymbol{c}_{s'}$ impossible, thus conforming to the (i) of delivery atomicity. Similarly, Lemma 8 proves the (ii) where $\boldsymbol{s}', \boldsymbol{r}, \boldsymbol{c}_{s'}$ must be released and confirmed if $\boldsymbol{E}_M \subsetneq \boldsymbol{E}$. Therefore, the delivery atomicity holds.

B. *Architecture Security*

Here, we analyse the architecture security of DECLOAK by considering its implementation.

We note that parties $\boldsymbol{P}$ are only required to send and receive transactions from blockchain BC and exchange protocol messages with multiple TEEs $\boldsymbol{\mathcal{E}}$ through their executors $\boldsymbol{E}$ in DN. Parties in $\boldsymbol{P}$ can implement the client using entirely different code bases, possibly using memory-safe languages like Rust and Go. Hence, we focus on $\boldsymbol{E}$ in the following.

Although DECLOAK protocol specifies $\mathscr{E}^* \leftarrow \boldsymbol{\mathcal{E}}[0]$ to execute MPTs for simplicity. The executor can be verifiably and randomly selected by other methods, such as verifiable random functions [3], [45], which is an orthogonal problem. However, we stress that the executor's selection method does not impair our system's availability, as parties can always challenge the selected executor on-chain by $Proc_{\text{rcha}}$. Recall that DECLOAK involves multiple TEE executors $\boldsymbol{E}$ and their TEEs $\boldsymbol{\mathcal{E}}$. If the selected executor is unavailable, it will be immediately challenged by $\boldsymbol{P}_H$, punished by BC and replaced by the next selected $\mathscr{E}$ in $\boldsymbol{\mathcal{E}}$ to avoid single point failure. Nevertheless, industrial TEE executors usually have robust error-handling and DDoS-resistant mechanisms. $Proc_{\text{rcha}}$ should rarely happen in practice. Meanwhile, since BC plays as a trusted anchor to maintain $\boldsymbol{\mathcal{E}}$'s consistency, the goal of a $\boldsymbol{E}_M$ reduces to exploit the enclave program $\mathscr{E}$ at runtime, *i.e.*, launching attacks by using well-defined interfaces of $\mathscr{E}$. Specifically, while we assume that TEE's confidentiality and integrity hold, interface-based attacks against TEEs, *e.g.*, memory-corruption attacks and side-channel attacks against SGX, keep coming out [46]–[48]. However, our design is TEE-agnostic and language-agnostic. It's easy to mitigate the above attacks by hardware-based countermeasures [49], [50] including more advanced TEE design [51] and software-based [52]–[54] including memory-safety languages, *e.g.*, Rust.

VIII. EVALUATION

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

- **Q1**: What is its cost of enabling MPTs on a blockchain?
- **Q2**: Can it serve MPTs in various scenarios?
- **Q3**: How it compares to related work in evaluating MPTs?

The experiment is based on a server with Ubuntu 18.04, 32G memory, and 2.2GHz Intel(R) Xeon(R) Silver 4114 CPU. The memory used by TEE is set up to 200M.

Answering Q1. It costs 4.9M gas to deploy $\mathscr{V}$ to enable DECLOAK on a blockchain. This cost is only once paid by DECLOAK service provider, thereby is irrelevant. In the global setup phase, a party pays 4.2k gas to *deposit* coins, which happens once for each party, thus being acceptable.

Answering Q2. We evaluate DECLOAK on 5 contracts with 10 MPTs in different scenarios. They are all in Solidity, and the number of involved parties varies from 2-11.

`SupplyChain` is a supply chain contract with 39 LOC. It has one 2-party MPT, which allows suppliers to negotiate with confidential bids off-chain, select a winner, update balances and commit their new balances with a proof on-chain.

`Scores` is a contract in the education domain. It has 95 LOC and contains one 2-party MPT that allows students to submit confidential answers, get their scores, and commit the evaluation on-chain.

`ERC20Token` is an ERC20 contract in the DeFi domain modified for multi-party transfer. It has 55 LOC and contains two 2-party and one 3-party MPTs. These MPTs allow accounts to jointly approve, transfer or transferFrom without revealing their secrets off-chain, respectively, and commit on-chain.

`YunDou` is a real-world industrial digital asset contract, technically an ERC20 token contract fine tuned for a novel co-managed account. The contract has 105 LOC and contains a 3-party, 2-party and 11-party MPTs, where we highlight the 11-party MPT allows account managers self-selectly vote to transfer tokens without revealing their votes.

`Oracle` is a real-world Oracle contract customized for a verifiable random number generation service. It has 60 LOC and contains a 2-party and an 11-party MPTs, providing functions for parties to negotiate and generate random numbers in a joint and verifiable manner.

Table III shows the gas cost of all MPTs, which proves that DeCloak can evaluate MPTs, including which in real-world contracts, in the cost comparable to classic DeFi or NFT transactions on Ethereum.

Table III
AVERAGE ON-CHAIN COST OF MPTS ENDING AT DIFFERENT STATUSES. FOR $Proc_{\text{RCHA}}$ OF AN MPT, WE ASSUME ALL PARTIES INVOLVED ARE CHALLENGED

Status	TX	Gas	Sum
-	*acknowledge* (TX_{ack_i})	26999	
	partyResponse (TX_{resP_i})	34313	
COMPLETED	*commit* (TX_{cmt})	104568	215138
	complete (TX_{com})	110570	
NEGOFAILED	*challengeTEE* (TX_{chaT})	131762	162325
	failNegotiation (TX_{fneg})	30563	
ABORT	*challengeTEE* (TX_{chaT})	131762	211066
	challengeParties (TX_{chaP})	33786	
	punishParties (TX_{pnsP})	45518	
ABORT	*challengeTEE* (TX_{chaT})	131762	185016
	punishTEE (TX_{pnsT})	53254	
	DeFi: ERC20: Transfer	65000	
	DeFi: Uniswap V3: Swap	184523	
	DeFi: Balancer: Swap	196625	
	NFT: OpenSea: Sale	71645	
	NFT: LooksRare: Sale	326897	

Answering Q3. We analyze the gas and off-chain cost for evaluating each MPT, respectively. We especially compare the gas cost of DeCloak with the most related two works, Fastkitten [2] and Cloak [5].

On-chain cost of MPTs. Figure 3 shows the gas cost of each MPT. Overall, DeCloak reduces gas by 72.5% against Fastkitten. Specifically, for six 2-party MPTs, DeCloak costs 0.27-0.46X gas. For two 3-party and two 10/11-party MPTs, the gas significantly reduces to 0.22-0.25X and 0.09-0.11X, respectively. For Cloak, the cost of DeCloak decreases by 65.6% in average. Specifically, DeCloak costs 0.27-0.56X gas against Cloak in 2/3-party MPTs, while just 0.17-0.22X gas in 10/11-party MPTs. Therefore, DeCloak enables a more secure MPTs with lower on-chain cost. The on-chain cost not only surpasses Cloak, but is comparable to typical DeFi/NFT transactions on Ethereum, *e.g.*, NFT sale and ERC20 swap. Moreover, as the number of parties grows, the cost superiority of DeCloak improves.

Off-chain cost of MPTs. All 10 MPTs complete in constant 2 transactions. Specifically, the negotiation, execution, and delivery phases cost 0.32-1.71s, 0.29-0.81s, and 0.27-1.27s, respectively. As Cloak's negotiation phase takes 0.1-0.39s and its execution and distribution phases take 0.26-0.71s, the off-chain time cost of DeCloak surpasses Cloak since we offload more workload to the off-chain TEE. However, we stress that both costs are ignorable in practically multi-party scenarios.

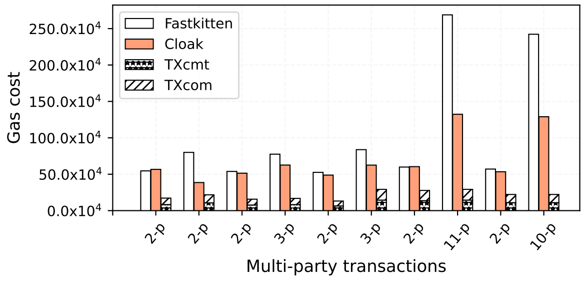

Figure 3. **The gas cost of DeCloak**. "Fastkitten" refers to the gas cost sum of $n+1$ transactions for each MPT. Here we adapt the protocol of Faskkitten to Ethereum. "Cloak" refers to the gas cost sum of its 2 transactions for each MPT. "TXcmt" and "TXcom" refers to gas cost of TX_{cmt}, TX_{com} in π_{DeCloak}, respectively.

On-chain cost of MPTs with adversaries presented. $Proc_{\text{rcha}}$ terminates at `ABORT` in two scenarios, punishing parties or the specified TEE. As shown in Table III, the costs of these two scenarios are 211066 and 185016, respectively, which reduce the cost of Cloak by 71.5% and 67.6%, respectively, or 69.6% on average. The main reason is that Cloak requires the TEE to commit parties' inputs and lock their collateral after negotiation, which is expensive to 471494 on average. Instead, DeCloak offloads most collateral operations to the TEE and reduces the inevitable commitment cost using `calldata`.

IX. Optimization and fine-tuning

A. *Improve the Scalability of* DeCloak

1) *Reduce gas cost in optimistic cases*

Recall that serving a n-party MPT in optimistic scenarios only involves 2 transactions, TX_{cmt} and TX_{com}, leading to $O(1)$ transactions. We can adopt the following measures to further reduce the optimistic cost of DeCloak.

Reduce the pessimistic cost by batch processing. In the pessimistic scenarios, each MPT will trigger a *challenge-response submission* ($Proc_{rcha}$) protocol. Each party being challenged on-chain has to respond with their inputs independently. We can involve an off-chain third-party service to collect parties' responses and publish an aggregated TX_{resP} to the blockchain. In this way, even though a $Proc_{rcha}$ is triggered, the on-chain transaction complexity is still $O(1)$. And combining with the batch processing technique of MPT, the complexity of $Proc_{rcha}$ can furthermore reduce to $O(1/m)$, where m is the number of MPTs in a batch.

Trading system goals for efficiency and cheapness. By intentionally sacrificing some system goals, DeCloak can further reduce its on-chain cost. First, we can drop *data* *availability* to delete the last transaction TX_{com}. Specifically, in the delivery phase, TEEs will first send TX_{cmt} to commit outputs on-chain. If the $proof$ in TX_{cmt} passes, $\mathscr{V}$ will accept the state transition immediately. Then, upon TX_{cmt} being accepted and confirmed, TEEs will release the keys of the output ciphertext in TX_{cmt} to parties by off-chain channels, rather than sending a TX_{com}. Consequently, the required transactions of DECLOAK reduce to only 1, *i.e.*, TX_{cmt}. However, in this variant, parties need to keep all received keys to access their plaintext states. Second, we can furthermore drop *delivery atomicity* and *delivery fairness* to delete TX_{cmt}, meaning that no transactions are required in the optimistic case. Specifically, MPT involves reading on-chain inputs. If we delete TX_{cmt}, when the specified TEE obtains outputs, the blockchain needs not to ensure the old states have not been mutated. This way, the MPT outputs that TEE regard as valid cannot be accepted by the blockchain, breaking the atomicity. Moreover, as we cannot utilize the TX_{cmt} to ensure that output ciphertext can be ideally delivered to all TEEs, *delivery fairness* is broken.

2) Reduce gas cost in pessimistic cases

In pessimistic scenarios, the *challenge-response subprotocol* ($Proc_{\text{rcha}}$) is triggered. Each party being challenged on-chain has to respond with their acknowledgements or inputs independently. We can introduce an off-chain third-party service to collect parties' responses and publish an aggregated TX_{resP} to the blockchain. In this way, even though a $Proc_{\text{rcha}}$ is triggered, the on-chain transaction complexity is still $O(1)$. And combining with the batch processing technique of MPT, the complexity of $Proc_{\text{rcha}}$ can furthermore reduce to $O(1/m)$, where m is the number of MPTs in a batch.

3) Reduce storage cost

To minimize the trust of off-chain TEE network, DECLOAK stores parties' data commitments on BC and ensures the corresponding plaintext are still accessible to parties even without DECLOAK. This sounds indicating a high storage cost. However, as demonstrated in Section VIII, the storage cost is acceptable by using `calldata`. Actually, `calldata` as storage has been well-adopted in Ethereum Rollup projects [16], [17]. Moreover, reducing the storage cost is also a main issue of Ethereum 2.0. Specifically, Ethereum proposes reducing the gas cost of `calldata` from 16 to 3, meaning an 81% decrease [15]. It also introduces `blob` [18], a new storage mechanism which allows different Ethereum Layer-2 projects to cheaply store all their transactions and states on the Beacon chain. Therefore, the design of DECLOAK strongly matches the need and tendency of Ethereum.

B. Improve the Availability of DECLOAK

Nevertheless, we can also improve the availability of DECLOAK further. For example, DECLOAK can adopt a similar availability enhancement mechanism as in POSE [3]. Specifically, every time the specified TEE executor changes its local state, it should synchronize the state updates to all other registered TEEs and collect their signatures in off-chain channels to carry on the next state transfer. If the specified TEE is not available off-chain, parties can publicly challenge it on-chain. If the unavailability of the specified TEE is because that other TEE executors do not respond with signatures as expected, the specified TEE can publicly challenge these unavailable TEEs on the blockchain. Finally, if the on-chain challenge-response mechanism finally punishes the specified TEE, it will be kicked out, and the next TEE in the registered list will be specified to serve MPTs. As a result, in an optimistic scenario, *i.e.*, all other TEEs honestly respond with their signatures, DECLOAK will not lose its off-chain states if at least one TEE is available. In a word, we stress that improving the availability of TEE network is an orthogonal field with DECLOAK, and DECLOAK can combine with related work [3] to further improve its availability.

X. CONCLUSION

In this paper, we develop a novel framework, DECLOAK, which can support MPT-enabled off-chain contract execution on legacy blockchains by using a TEE network. DECLOAK features maximising the security of MPT and minimising the gas cost and the network's trust. Compared with the SOTA, Cloak, DECLOAK not only realizes all security properties the SOTA claimed but also achieves data availability, delivery fairness, and delivery atomicity. To our knowledge, DECLOAK achieves the most general and secure MPT. Meanwhile, it assumes at least one party and executor are honest, which is also one of the weakest assumptions compared to related work. Moreover, according to our evaluation, DECLOAK reduces the gas cost of the SOTA by 65.6%, and the superiority of DECLOAK increases as the number of parties grows.

REFERENCES

[1] R. Cheng, F. Zhang, J. Kos, W. He, N. Hynes, N. Johnson, A. Juels, A. Miller, and D. Song, "Ekiden: A Platform for Confidentiality-Preserving, Trustworthy, and Performant Smart Contracts," *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*, vol. 00, pp. 185–200, 2019.

[2] P. Das, L. Eckey, T. Frassetto, D. Gens, K. Hostáková, P. Jauernig, S. Faust, and A.-R. Sadeghi, "Fastkitten: Practical smart contracts on bitcoin," in *28th USENIX Security Symposium (USENIX Security 19)*. Santa Clara, CA: USENIX Association, Aug. 2019, pp. 801–818. [Online]. Available: https://www.usenix.org/conference/usenixsecurity19/presentation/das

[3] T. Frassetto, P. Jauernig, D. Koisser, D. Kretzler, B. Schlosser, S. Faust, and A.-R. Sadeghi, "Pose: Practical off-chain smart contract execution," in *Proceedings of the 2022 Network and Distributed System Security Symposium*, vol. abs/2210.07110, 2022.

[4] Q. Ren, H. Liu, Y. Li, and H. Lei, "Demo: Cloak: A framework for development of confidential blockchain smart contracts," in *2021 IEEE 41st International Conference on Distributed Computing Systems (ICDCS)*, 2021, pp. 1102–1105.

[5] Q. Ren, Y. Wu, H. Liu, Y. Li, A. Victor, H. Lei, L. Wang, and B. Chen, "Cloak: Transitioning states on legacy blockchains using secure and publicly verifiable off-chain multi-party computation," in *Proceedings of the 38th Annual Computer Security Applications Conference*, 2022, pp. 117–131.

[6] A. Kosba, A. Miller, E. Shi, Z. Wen, and C. Papamanthou, "Hawk: The Blockchain Model of Cryptography and Privacy-Preserving Smart Contracts," *2016 IEEE Symposium on Security and Privacy (SP)*, pp. 839–858, 2016.

[7] R. Sinha, "Luciditee: A tee-blockchain system for policy-compliant multiparty computation with fairness," 2020.

[8] K. Govindarajan, D. Vinayagamurthy, P. Jayachandran, and C. Rebeiro, "Privacy-preserving decentralized exchange marketplaces," in *2022 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, 2022, pp. 1–9.

[9] F. Massacci, C. N. Ngo, J. Nie, D. Venturi, and J. Williams, "Futuresmex: Secure, distributed futures market exchange," in *2018 IEEE Symposium on Security and Privacy (SP)*, 2018, pp. 335–353.

[10] C. Baum, I. Damgård, and C. Orlandi, "Publicly auditable secure multi-party computation," in *Security and Cryptography for Networks*, M. Abdalla and R. De Prisco, Eds. Cham: Springer International Publishing, 2014, pp. 175–196.

[11] D. Boneh, E. Boyle, H. Corrigan-Gibbs, N. Gilboa, and Y. Ishai, "Zero-knowledge proofs on secret-shared data via fully linear pcps," Cryptology ePrint Archive, Paper 2019/188, 2019, https://eprint.iacr.or g/2019/188. [Online]. Available: https://eprint.iacr.org/2019/188

[12] H. Cui, K. Zhang, Y. Chen, Z. Liu, and Y. Yu, "Mpc-in-multi-heads: A multi-prover zero-knowledge proof system," in *European Symposium on Research in Computer Security*. Springer, 2021, pp. 332–351.

[13] S. Steffen, B. Bichsel, R. Baumgartner, and M. Vechev, "Zeestar: Private smart contracts by homomorphic encryption and zero-knowledge proofs," in *2022 IEEE Symposium on Security and Privacy (SP)*, 2022, pp. 179–197.

[14] EthHub, "Data availability," May 2023. [Online]. Available: https: //ethereum.org/en/developers/docs/data-availability

[15] V. Buterin and A. Dietrichs, "Eip-4488: Transaction calldata gas cost reduction with total calldata limit," Nov 2021. [Online]. Available: https://eips.ethereum.org/EIPS/eip-4488

[16] EthHub, "Optimistic rollups," July 2022. [Online]. Available: https: //docs.ethhub.io/ethereum-roadmap/layer-2-scaling/optimistic_rollups

[17] ——, "Zk-rollups," July 2022. [Online]. Available: https://docs.ethhub. io/ethereum-roadmap/layer-2-scaling/zk-rollups

[18] V. Buterin, D. L. Dankrad Feist, G. Kadianakis, M. Garnett, and A. Dietrichs, "Eip-4844: Shard blob transactions," Feb 2022. [Online]. Available: https://eips.ethereum.org/EIPS/eip-4844

[19] Ethereum, "Cancun network upgrade specification," May 2023. [Online]. Available: https://github.com/ethereum/execution-specs/blob /master/network-upgrades/mainnet-upgrades/cancun.md#included-eips

[20] K. Qin, L. Zhou, and A. Gervais, "Quantifying blockchain extractable value: How dark is the forest?" in *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2022, pp. 198–214.

[21] G. Wood *et al.*, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum project yellow paper*, 2014.

[22] D. Lee, D. Kohlbrenner, S. Shinde, K. Asanović, and D. Song, "Keystone: An open framework for architecting trusted execution environments," in *Proceedings of the Fifteenth European Conference on Computer Systems*, ser. EuroSys '20. New York, NY, USA: Association for Computing Machinery, 2020. [Online]. Available: https://doi.org/10.1145/3342195.3387532

[23] V. Costan and S. Devadas, "Intel sgx explained." *IACR Cryptol. ePrint Arch.*, vol. 2016, no. 86, pp. 1–118, 2016.

[24] AMD, "Amd sev-snp: Strengthening vm isolation with integrity protection and more," https://www.amd.com/system/files/TechDocs/SEV-SN P-strengthening-vm-isolation-with-integrity-protection-and-more.pdf, Jan 2020.

[25] B. Ngabonziza, D. Martin, A. Bailey, H. Cho, and S. Martin, "Trustzone explained: Architectural features and use cases," in *2016 IEEE 2nd International Conference on Collaboration and Internet Computing (CIC)*, 2016, pp. 445–451.

[26] R. Pass, E. Shi, and F. Tramèr, "Formal abstractions for attested execution secure processors," in *Advances in Cryptology – EUROCRYPT 2017*, J.-S. Coron and J. B. Nielsen, Eds. Cham: Springer International Publishing, 2017, pp. 260–289.

[27] Wikipedia, "Elliptic-curve diffie–hellman," June 2023. [Online]. Available: https://en.wikipedia.org/wiki/Elliptic-curve_Diffie%E2% 80%93Hellman

[28] C. Lederer, R. Mader, M. Koschuch, J. Großschädl, A. Szekely, and S. Tillich, "Energy-efficient implementation of ecdh key exchange for wireless sensor networks," in *Information Security Theory and Practice. Smart Devices, Pervasive Systems, and Ubiquitous Networks*, O. Markowitch, A. Bilas, J.-H. Hoepman, C. J. Mitchell, and J.-J. Quisquater, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 112–127.

[29] E. Mbadu, "Why elliptic curve diffie-hellman (ecdh) is replacing diffie-hellman (dh)," *Journal of Computing Sciences in Colleges*, vol. 35, no. 3, pp. 218–218, 2019.

[30] E. B. et al., "Recommendation for pair-wise key-establishment schemes using discrete logarithm cryptography," Apr. 2018. [Online]. Available: https://csrc.nist.gov/pubs/sp/800/56/a/r3/final

[31] D. Maier, R. Pottinger, A. Doan, W.-C. Tan, A. Alawini, H. Q. Ngo, Y. Yan, C. Wei, X. Guo, X. Lu, X. Zheng, Q. Liu, C. Zhou, X. Song, B. Zhao, H. Zhang, and G. Jiang, "Confidentiality Support over Financial Grade Consortium Blockchain," 2020, pp. 2227–2240.

[32] S. Bowe, A. Chiesa, M. Green, I. Miers, P. Mishra, and H. Wu, "ZEXE: Enabling Decentralized Private Computation," *2020 IEEE Symposium on Security and Privacy*, 2020.

[33] A. R. Choudhuri, M. Green, A. Jain, G. Kaptchuk, and I. Miers, "Fairness in an Unfair World: Fair Multiparty Computation from Public Bulletin Boards," ser. Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, 2017, pp. 719–728.

[34] M. Russinovich, E. Ashton, C. Avanessians, M. Castro, A. Chamayou, S. Clebsch, and et al., "Ccf: A framework for building confidential verifiable replicated services," Microsoft Research and Microsoft Azure, Tech. Rep., Apr. 2019.

[35] L. Cavallaro, J. Kinder, X. Wang, J. Katz, I. Bentov, Y. Ji, F. Zhang, L. Breidenbach, P. Daian, and A. Juels, "Tesseract: Real-Time Cryptocurrency Exchange Using Trusted Hardware," ser. Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, 2019, pp. 1521–1538.

[36] E. Weippl, S. Katzenbeisser, C. Kruegel, A. Myers, S. Halevi, R. Kumaresan, and I. Bentov, "Amortizing Secure Computation with Penalties," *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pp. 418–429, 2016.

[37] E. Weippl, S. Katzenbeisser, C. Kruegel, A. Myers, S. Halevi, R. Kumaresan, V. Vaikuntanathan, and P. N. Vasudevan, "Improvements to Secure Computation with Penalties," *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pp. 406–417, 2016.

[38] I. Ray, N. Li, C. Kruegel, R. Kumaresan, T. Moran, and I. Bentov, "How to use bitcoin to play decentralized poker," *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pp. 195–206, 2015.

[39] F. Baldimtsi, A. Kiayias, T. Zacharias, and B. Zhang, "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*. Berlin, Heidelberg: Springer-Verlag, 2020, p. 717–748. [Online]. Available: https://doi.org/10.1007/978-3-030-6484 0-4_24

[40] A. Ozdemir and D. Boneh, "Experimenting with collaborative zk-SNARKs: Zero-Knowledge proofs for distributed secrets," in *31st USENIX Security Symposium (USENIX Security 22)*. Boston, MA: USENIX Association, Aug. 2022, pp. 4291–4308. [Online]. Available: https://www.usenix.org/conference/usenixsecurity22/presentation/ozde mir

[41] J. Garay, A. Kiayias, and N. Leonardos, "The bitcoin backbone protocol: Analysis and applications," in *Annual international conference on the theory and applications of cryptographic techniques*. Springer, 2015, pp. 281–310.

[42] Ethereum, "Solc 0.8.10," July 2021. [Online]. Available: https: //github.com/ethereum/solidity/releases/tag/v0.8.10

[43] E. Foundation, "Ethereum virtual machine," Dec 2020. [Online]. Available: https://ethereum.org/en/developers/docs/evm/

[44] S. State and O. Labs, "Confidential Ethereum Smart Contracts," Tech. Rep., 12 2020.

[45] Binance, "Introducing binance oracle vrf: The next generation of verifiable randomness," Aug. 2023. [Online]. Available: https: //www.binance.com/en/blog/tech/introducing-binance-oracle-vrf-the-ne xt-generation-of-verifiable-randomness-114582038468709401

[46] A. Biondo, M. Conti, L. Davi, T. Frassetto, and A.-R. Sadeghi, "The guard's dilemma: Efficient Code-Reuse attacks against intel SGX," in *27th USENIX Security Symposium (USENIX Security 18)*. Baltimore, MD: USENIX Association, Aug. 2018, pp. 1213–1227. [Online]. Available: https://www.usenix.org/conference/usenixsecurity18/present ation/biondo

[47] J. V. Bulck, M. Minkin, O. Weisse, D. Genkin, B. Kasikci, F. Piessens, M. Silberstein, T. F. Wenisch, Y. Yarom, and R. Strackx, "Foreshadow: Extracting the keys to the intel SGX kingdom with transient Out-of-Order execution," in *27th USENIX Security Symposium (USENIX Security 18)*. Baltimore, MD: USENIX Association, Aug. 2018, p. 991–1008. [Online]. Available: https: //www.usenix.org/conference/usenixsecurity18/presentation/bulck

[48] Intel, "Resources and response to side channel l1 terminal fault," Dec 2021. [Online]. Available: https://www.intel.com/content/www/us/en/ar chitecture-and-technology/l1tf.html?wapkw=l1tf

[49] J. Noorman, P. Agten, W. Daniels, R. Strackx, A. Van Herrewege, C. Huygens, B. Preneel, I. Verbauwhede, and F. Piessens, "Sancus: Low-cost trustworthy extensible networked devices with a zero-software

trusted computing base," in *22nd USENIX Security Symposium (USENIX Security 13)*, 2013, pp. 479–498.

[50] V. Costan, I. Lebedev, and S. Devadas, "Sanctum: Minimal hardware extensions for strong software isolation," in *25th USENIX Security Symposium (USENIX Security 16)*, 2016, pp. 857–874.

[51] R. Bahmani, F. Brasser, G. Dessouky, P. Jauernig, M. Klimmek, A.-R. Sadeghi, and E. Stapf, "Cure: A security architecture with customizable and resilient enclaves," in *USENIX Security Symposium*, 2020. [Online]. Available: https://api.semanticscholar.org/CorpusID:226221896

[52] D. Gruss, J. Lettner, F. Schuster, O. Ohrimenko, I. Haller, and M. Costa, "Strong and efficient cache side-channel protection using hardware transactional memory," in *26th USENIX Security Symposium (USENIX Security 17)*, 2017, pp. 217–233.

[53] M.-W. Shih, S. Lee, T. Kim, and M. Peinado, "T-sgx: Eradicating controlled-channel attacks against enclave programs." in *NDSS*, 2017.

[54] F. Lang, W. Wang, L. Meng, J. Lin, Q. Wang, and L. Lu, "Mole: Mitigation of side-channel attacks against sgx via dynamic data location escape," *Proceedings of the 38th Annual Computer Security Applications Conference*, 2022.