Eagle: Efficient Privacy Preserving Smart Contracts

Carsten Baum[1(✉)], James Hsin-yu Chiang[1], Bernardo David[2], and Tore Kasper Frederiksen[3]

[1] Technical University of Denmark, Kongens Lyngby, Denmark
cabau@dtu.dk, jachiang@ucla.edu
[2] IT University of Copenhagen, Copenhagen, Denmark
bernardo@bmdavid.com
[3] Alexandra Institute, Aarhus, Denmark

Abstract. The proliferation of Decentralised Finance (DeFi) and Decentralised Autonomous Organisations (DAO), which in current form are exposed to front-running of token transactions and proposal voting, demonstrate the need to shield user inputs and internal state from the parties executing smart contracts. In this work we present "Eagle", an efficient UC-secure protocol which efficiently realises a notion of privacy preserving smart contracts where both the amounts of tokens and the auxiliary data given as input to a contract are kept private from all parties but the one providing the input. Prior proposals realizing privacy preserving smart contracts on public, permissionless blockchains generally offer a limited contract functionality or require a trusted third party to manage private inputs and state. We achieve our results through a combination of secure multi-party computation (MPC) and zero-knowledge proofs on Pedersen commitments. Although other approaches leverage MPC in this setting, these incur impractical computational overheads by requiring the computation of cryptographic primitives within MPC. Our solution achieves security without the need of any cryptographic primitives to be computed inside the MPC instance and only require a constant amount of exponentiations per client input.

Keywords: Blockchain · DeFi · MPC · Privacy

Carsten Baum: Part of the work was carried out while the author was visiting Copenhagen University and supported by Partisia. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of Partisia.
Bernardo David: The project was supported by the Concordium Foundation, by the Independent Research Fund Denmark (IRFD) grants number 9040-00399B (TrA^{2}C), 9131-00075B (PUMA) and 0165-00079B, and by Copenhagen Fintech.
Tore Kasper Frederiksen: The work was carried out while at the Alexandra Institute, supported by Copenhagen Fintech as part of as part of the "National Position of Strength programme for Finans & Fintech" funded by the Danish Ministry of Higher Education and Science.

F. Baldimtsi and C. Cachin (Eds.): FC 2023, LNCS 13950, pp. 270–288, 2024.
https://doi.org/10.1007/978-3-031-47754-6_16

1 Introduction

Ethereum introduced the first implementation of Turing-complete smart contracts for blockchains, widely adopted for financial and contracting applications since its introduction in 2015. Smart contracts offer auditability and correctness guarantees, and as a consequence expose both their state and any submitted inputs to all participants of the blockchain network. This lack of privacy not only leaks user data but also gives rise to concrete attacks. For example, current Decentralised Finance (DeFi) and Decentralised Autonomous Organisations (DAO) are vulnerable to front-running [23] in token transactions and proposal voting. This motivates the need to shield user inputs and internal contract state from the very parties who execute smart contracts in a decentralized environment.

Challenges. Hawk [37] introduced the first notion of general-purpose privacy preserving smart contracts, which required users to privately submit both input strings and confidential balances to a trusted contract manager. Upon evaluation of the contract over private inputs, the contract manager settles the confidential outputs to a confidential ledger, proving in zero knowledge that these outputs have been obtained according to the contract's instructions. Importantly, in order to accommodate real-world applications such as DeFi or DAO's, we must extend the Hawk notion of confidential contracts as follows:

1. Distribute the role of the trusted third party in an efficient manner, avoiding a single point of failure without significantly sacrificing performance.
2. Only require clients to be online during a short input phase; as in the standard client-blockchain interaction model, clients only broadcast signed inputs.
3. Allow privacy preserving smart contracts to be long-running applications over indefinite rounds, as is the case in standard, public smart contracts.

Our Contributions. In this work we present "Eagle", a Universally Composable [17] protocol for achieving efficient privacy preserving smart contracts, which handles all the three challenges explained above: (1) is achieved by evaluating the contract's instructions via an *outsourced* secure multi-party computation (MPC) protocol [31], where clients provide private inputs and servers execute the bulk of the protocol to compute a function on these inputs without learning them. We use a MPC protocol, known as *insured MPC*, which allows a public verifier to identify servers aborting at the output phase, so that cheating servers can be identified and financially punished, incentivizing fairness (*i.e.* if a server gets the output, all servers/clients also get it) [7]. That is, by combining outsourced and insured MPC we get a protocol where client computation and interaction is independent of the circuit computed in MPC and where *reliability* is incentivized and *security* is obtained as long as only a single MPC is honest. (2) is accomplished with a novel input protocol which pre-processes data necessary for the servers to generate private outputs (*e.g.* token amounts) that are posted directly to the public ledger but can only be read by specific clients. (2) facilitates (3), realized by a *reactive* version of our MPC protocol, which maintains a secret off-chain state over multiple rounds. Here, we contribute a model

of long-running, privacy preserving contracts, which at the onset of each round accepts new inputs from any subset of clients. At the end of each round, clients get public outputs and servers keep a secret internal state, allowing evaluation to take place in a continuous, *multi-round* fashion, even if clients are offline (2).

Applications. Several general applications for privacy preserving smart contracts have already been proposed. **Auctions:** can be realized securely on-chain with privacy preserving smart contracts, as auctions implemented without privacy are vulnerable to front-running (miners can trivially observe individual bids posted to the ledger). **Identity management:** Decentralized Identity (DID) management considers the setting where user-attributes are posted to a ledger, in a certified, yet hidden manner. DID implemented with privacy preserving smart contracts enables proofs and computations on private identity attributes, facilitating their integration with blockchain applications. **KYC Mixing:** We can construct a privacy preserving smart contract to realize a mixer that enforces Anti Money Laundering (AML) policies. For example, such a mixer could use DID to integrate Know Your Customer (KYC) information to either limit user permissions or the quantity of mixed tokens allowed per month. **Side-chains:** The MPC servers alone could be considered a privacy preserving side-chain. Multiple sets of MPC servers could work together with a single smart contract to realize a privacy preserving sharding scheme on any layer 1 chain with Turing complete smart contracts. **AMMs and DeFi via Cross-chain contracts:** Using ideas of P2DEX [9], we show that the MPC servers can interact with smart contracts on many different ledgers. Hence, privacy preserving smart contracts can work *across* multiple ledgers and different native tokens. This realizes cross-chain, front-running resistant automated market makers (AMMs) with strong privacy guarantees. We discuss these applications in more detail in the full version [6].

Our Techniques. We sketch our protocol in Fig. 1. This only considers execution of a *single* instance of a privacy preserving smart contract for simplicity. We discuss the multi-round setting in the full version [6]. where computations are executed continuously with different sets of clients. We assume a set of clients $\mathcal{C}$ and MPC servers $\mathcal{P}$, both interacting with a ledger functionality $\mathcal{F}_{\mathsf{Ledger}}$. The ledger hosts two deployed smart contract instances: $\mathcal{X}_{\underline{\mathsf{C}}\mathsf{Ledger}}$ maintains a confidential ledger and is extended with $\mathcal{X}_{\mathsf{Lock}}$, which locks and redistributes confidential balances, output and jointly signed, by the MPC servers. Concretely our protocol runs the following phases:

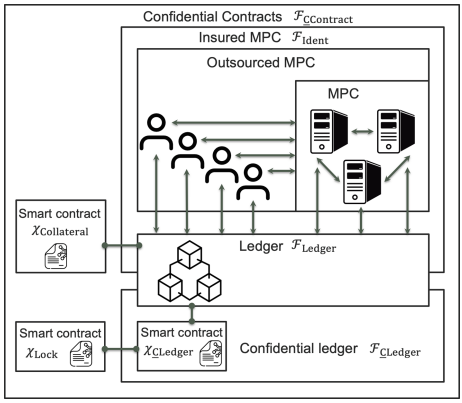


Fig. 1. Outline of our protocol for confidential contracts. The wrapping and interaction of functionalities are shown.

Init. Before any execution, the servers setup the system by sampling a threshold signature key pair and provide sufficient collateral for the insured MPC execution, and setup smart contact $\mathcal{X}_{\mathsf{Lock}}$, administered by the distributed signature key. We note that in the multi-round setting this only needs to be executed *once* for the specific set of MPC servers, and is thus independent of the clients and the amount of computations that will get carried out later.

Enroll. When a privacy preserving smart contract is to be executed, each client who wish to participate transfers confidential tokens to $\mathcal{X}_{\mathsf{CLedger}}$ which they wish to use as input to the confidential smart contract $\underline{\mathsf{C}}\mathsf{Contract}$. The client then gives any auxiliary input, along with the opening information to the commitment containing their confidential balance v, to the insured MPC functionality $\mathcal{F}_{\mathsf{Ident}}$ from the work of Baum *et al.* [7,9], extended with a secure client input interface to allow for outsourced MPC [31] and described in detail in the full version [6]. Each client constructs an appropriate amount of "mask" commitments; one for each round of confidential contract computation, for which they wish their input to be used. A masking commitment is simply a commitment to a random value.

Verify Input. The servers validate the input received from the clients using outsourced MPC, and ensure that $\mathcal{X}_{\mathsf{Lock}}$ has also received the appropriate confidential tokens. The servers and the clients also execute a proof to ensure that the opening information supplied by clients are indeed valid for the confidential token commitments. They do this following a standard Σ-protocol where each client commits to a random commitment a and servers select a random challenge γ and ask the client to open $\mathrm{com}(c) = \mathrm{com}(a) \oplus (\gamma \odot \mathrm{com}(v))$. Similarly the servers use MPC to securely open $[c] = [a] + \gamma \cdot [v]$ and check consistency[1].

Evaluate. After the checks are completed the servers evaluate the circuit expressing the private smart contract $\underline{\mathsf{C}}\mathsf{Contract}$, using insured MPC. For the clients who are supposed to get output from this round of computation, shares of messages and randomness for a new commitment for *each* client are computed, and blinded with the "masking" values the clients provided during *Enroll*. If this goes well, the servers distributedly sign a message saying that they have reached this stage and post it to $\mathcal{X}_{\mathsf{Lock}}$.

Open. For clients that receive output after this round of computation the servers open the masked output. They publish these values and sign them, as part of the transcript of the current round execution, and post this to $\mathcal{X}_{\mathsf{Lock}}$. Note that $\mathcal{X}_{\mathsf{Lock}}$ can generate the output coins in commitment form, due to the homomorphism of the commitments and since it obtained the mask commitments from the clients in *Enroll*. $\mathcal{X}_{\mathsf{Lock}}$ can then transfer the new confidential tokens back to the client's address. We show an extension to our protocol in the full version [6]) that ensures no token minting can occur even if all servers are corrupted.

Withdraw. Based on the masks they constructed, the clients who are supposed to receive outputs can compute the coin commitment openings from their masked outputs signed and posted to $\mathcal{X}_{\mathsf{Lock}}$ by servers during *Open*.

[1] In our full protocol we optimize this by batching client input checks.

Abort. In case a server stops responding or acts maliciously, an honest server can request the entering of an abort phase. Any server can do this, either by submitting a proof that the malicious server sent wrong information or by requesting missing information from the accused servers. At this point the accused malicious server has a constant amount of time left to prove to the smart contract that they did not abort, by submitting the message that the accusing server claims they didn't get. If they don't, they will have their collateral revoked and it will be shared among the honest servers and clients, and the contract state will roll back one round, i.e. to the contract state preceding *Evaluate*. Concretely $\mathcal{X}_{\mathsf{Lock}}$ will refund the clients their input funds, plus a compensation obtained from the cheating servers' collateral.

Related Works. A long line of work realizes notions of privacy preserving smart contracts that sacrifice privacy [21,32,35,37,41,46–48] or flexibility [14,15]. Zexe [14] extends the ZCash model of confidential transactions to enable Bitcoin Script-like stateless privacy preserving smart contracts supporting only very simple logic. Zether [15] implements confidential transactions on top of Ethereum, allowing for very simple privacy preserving applications (*e.g.* auctions). Zkay [47] allows for computing on encrypted private inputs by means of keeping data encryption on the blockchain, and using NIZKs to validate that any updates done to the encrypted is carried out correctly. Follow-up work, Zeestar [46] uses additively homomorphic encryption to allow for *limited* private computation on data from multiple owners, without them having to share their private data with each other. Secret Network [48] and Ekiden [21] implement general purpose contracts but rely on notoriously vulnerable trusted execution environments (*e.g.* Intel SGX [42]) for privacy and correctness. Arbitrum [32] relies on a full quorum of parties (the servers in our setting) being honest to achieve privacy for general purpose contracts. Finally, Kachina [35] subsumes these approaches with a framework based on state oracles [41] that yields privacy preserving smart contracts, where either flexibility is limited (*i.e.* contract state is only updated by one client's private input at a time) or privacy is compromised (*i.e.* a trusted third party must learn clients' private inputs in order to update the state). The ideal functionality of Kachina is designed to permit *input concurrency*, allowing honest inputs to be finalized on a global ledger in a different order as their generation; the Kachina protocol requires private inputs to be accompanied with NIZKs proving a valid update of the private state fragment. Here, the NIZKs are not bound to a specific, public contract state and thus remain valid even if the public contract state observed by the user was updated by another user input in the meantime.

Combining MPC with blockchain based cryptocurrencies and smart contracts has been investigated in a long line of works [1,2,7–12,22,27,36,38–40] aiming at achieving fairness in the dishonest majority setting via financial punishments. The core idea of these works is having all parties, who execute the MPC protocol,

provide a collateral deposit, which is taken from them in case they are caught cheating. Thus incentivizing honest behavior. However, this approach publicly reveals the amount of collateral deposited by each party, which falls short of achieving our notion of privacy preserving smart contracts, where both auxiliary data *and* the amount of tokens given as input to the contract must remain private. Notice that revealing the deposit amount is an issue in applications where this amount is directly related to the client's private input, *e.g.* in sealed-bid auctions, where the collateral deposit must be equal to at least the client's private bid. An auction protocol using collateral deposits with private amounts was proposed in [28] but it cannot be generalized to other tasks.

Hawk [37, App. G] does suggest to use MPC to achieve a decentralized confidential smart contracts on both token amount and auxiliary input. However, Hawk works in the ZCash model and thus their MPC solution would require the computation of SNARKs to realize the ZCash transactions, within the MPC circuit. Although it has been shown [33,43] that integrating NIZKs with MPC can be done without degrading performance too much, there is still a performance hit. Since the construction of a single ZCash transaction SNARK still takes a non-negligible amount of time plain, this would naturally be inefficient to realize in MPC, as MPC is orders of magnitude slower than regular computation. Furthermore, they need *all* users to take part in the MPC computation. zkHawk [4] improves upon this, by forgoing the need of doing SNARKs in MPC, but still require *all* users taking part in a confidential smart contract to facilitate an MPC computation which *must* compute Schnorr style ZKPs on Pedersen commitments to the bit-decomposition of the amount of coins each of them hold. While V-zkHawk [5] forgoes the need of proofs of the bit-decomposed commitments, they replace it with the computation of commitments in a larger fields and a signature, in MPC instead. While more efficient, this approach would still require MPC over a large domain and contributes non-negligible overhead. In the full version [6]. we further discuss related works.

2 Preliminaries

Let $y \leftarrow\!\$\, F(x)$ denote running the randomized algorithm F with input x and implicit randomness, and obtaining output y. Similarly, $y \leftarrow F(x)$ is used for a deterministic algorithm. For a set $\mathcal{X}$, let $x \leftarrow\!\$\, \mathcal{X}$ denote x chosen uniformly at random from $\mathcal{X}$. s denotes the computational and κ the statistical security parameter. Let $[\![x]\!]$ denote secret x maintained in an MPC instance: we lift the $[\![\cdot]\!]$ notation to any object that can be encoded over secrets securely input to an MPC scheme, e.g. $[\![g]\!]$, where g is an arithmetic circuit

Table 1. Notation.

$\mathcal{P}$	The set of servers
$\mathcal{C}$	The set of clients
n	Number of servers $n = \lvert\mathcal{P}\rvert$
m	Number of clients; $m = \lvert\mathcal{C}\rvert$
l	Number of bits representing balances
z	Number of input/output per client
κ	Computational security parameter
s	Statistical security parameter
$\mathcal{F}$	An ideal functionality
Π	A protocol
$\mathcal{L}$	A ledger map indexed by vk
$\mathcal{X}$	A smart contract program
g	A smart contract in circuit form
vk	A public key for signature verification
x	A client input
y	A client output
$\bar{\mathbf{v}}$	A token balance
$\bar{v}_{\mathsf{max}}$	The maximum permitted balance
$\bar{\mathbf{v}}_{\mathsf{max}}$	A vector of the maximum permitted balance

over field $\mathbb{F}$. We use a group $\mathbb{G}$ where the discrete log problem is hard, and which is a source group of pairing scheme. For simplicity we assume $|\mathbb{G}| = |\mathbb{F}| = p$. Unless noted otherwise we use log to denote the logarithm to base 2, rounded up. We use $\bar{v}_{\mathsf{max}}$ to denote the maximum amount of tokens we want to represent and say $l = \log(\bar{v}_{\mathsf{max}})$. For simplicity, we assume $|\mathcal{C}| \cdot \bar{v}_{\mathsf{max}} < |\mathbb{G}|$, where $\mathcal{C}$ is the set of participating clients. We denote set $\{1, 2, \ldots, n\}$ by $[n]$ and vectors by bold faced Latin letters, e.g. $\mathbf{v}, \mathbf{w}$.

2.1 Security Model and Building Blocks

We analyse our results in the the (Global) Universal Composability or (G)UC framework [18,20]. We consider static malicious adversaries. Our protocols work in a synchronous communication setting, which is modeled by assuming parties have access to a global clock ideal functionality $\mathcal{F}_{\mathsf{Clock}}$ as seen in multiple works [3,34,36]. The core component of our protocols is publicly verifiable MPC with cheater identification in the output phase, which is modelled as an ideal functionality $\mathcal{F}_{\mathsf{Ident}}$, which can be realized as described by Baum *et al.* [7,9]. This functionality produces a proof that either a certain output was obtained after the MPC or that a certain party has misbehaved in the output phase, while cheating before the output phase causes an abort without cheater identification. We further extend this functionality to handle reactive computation [25,26] and an *outsourced* computation with inputs provided by clients and computation done by servers [24,31]. Moreover, we use Pedersen Commitments [44], digital signatures represented by an ideal functionality $\mathcal{F}_{\mathsf{Sig}}$ as in [19], threshold signatures represented by an ideal functionality $\mathcal{F}_{\mathsf{TSig}}$ as defined by Baum *et al.* [9] and non-interactive zero knowledge proofs represented by $\mathcal{F}_{\mathsf{NIZK}}$ as defined by Groth [30]. Further discussion on our security model and building blocks are presented in the full version [6].

2.2 Ledgers and Smart Contracts

We model a ledger functionality $\mathcal{F}_{\mathsf{Ledger}}$ in the full version [6]. featuring a smart contract virtual machine which is adapted from an authenticated, public bulletin board functionality, an approach adopted from the work of Baum *et al.* [7,9]. For this work, we emphasize accurate modelling of confidential balances, which are implemented on a public ledger, and omit the full consensus details in our UC model, similar to previous works [3,36].

Token Universe. $\mathcal{F}_{\mathsf{Ledger}}$ supports a token universe consisting of t token types: $\mathbb{T} = (\tau_1, ..., \tau_t)$. A ledger in $\mathcal{F}_{\mathsf{Ledger}}$ maintains a map from signature verification key to balances of each token type: $\mathcal{L} : \{0,1\}^* \rightarrow \mathbb{Z}^t$. We write $\bar{\mathbf{v}} = (v_1, ..., v_t)$ for a balance over all supported token types. In addition to balances associated to signature verification keys, $\mathcal{F}_{\mathsf{Ledger}}$ also maintains token balances for each deployed smart contract instance. The ledger functionality enforces the preservation of token supplies over $\mathbb{T}$.

Overview of Smart Contracts. In this work, we present smart contracts as human-readable programs and assume the presence of a compiler which translates program $\mathcal{X}$ to a valid circuit T and initial state γ_{init}. The following smart contract programs are deployed in the protocol which realizes the proposed confidential contract functionality $\mathcal{F}_{\underline{\mathsf{C}}\mathsf{Contract}}$.

- $\mathcal{X}_{\underline{\mathsf{C}}\mathsf{Ledger}}$ (described in the full version [6]) describes a smart contract which implements a confidential token wrapper for each token in $\mathbb{T}$ supported on the base ledger $\mathcal{F}_{\mathsf{Ledger}}$.
- $\mathcal{X}_{\mathsf{Lock}}$ (described in the full version [6]) is an extension to $\mathcal{X}_{\underline{\mathsf{C}}\mathsf{Ledger}}$. It permits the locking and redistribution of confidential balances authorized by verifying threshold signatures generated by the servers (via global functionality $\mathcal{F}_{\mathsf{TSig}}$).
- $\mathcal{X}_{\mathsf{Collateral}}$ (described in the full version [6]) accepts collateral deposits from servers, which upon being identified as cheating parties lose their collateral to clients.

2.3 Confidential Ledgers from $\mathcal{F}_{\mathsf{Ledger}}$

We briefly describe a confidential ledger functionality $\mathcal{F}_{\underline{\mathsf{C}}\mathsf{Ledger}}$, presented in full detail in the full version [6], that can be implemented from a hybrid $\mathcal{F}_{\mathsf{Ledger}}$ functionality, enabling both confidential balances and the confidential transfer of default tokens types $\mathbb{T}$ exposed by the underlying public ledger $\mathcal{F}_{\mathsf{Ledger}}$. This modeling choice maximizes the generality of our construction, as it can be implemented on any standard ledger and a basic smart contract machine.

Confidential Ledger. Confidential coins in $\mathcal{F}_{\underline{\mathsf{C}}\mathsf{Ledger}}$ are identifiable by a unique public id, and a confidential balance $\bar{\mathbf{v}}$ over $\mathbb{T}$, as in [45]. Each confidential token is publicly associated with an account verification key vk, owned by a party that generated it with GenAcct. A confidential transfer consumes two input coins $(\mathsf{id}_1, \mathsf{id}_2)$ with confidential balances $(\bar{\mathbf{v}}_1, \bar{\mathbf{v}}_2)$ and mints fresh output coins $(\mathsf{id}'_1, \mathsf{id}'_2)$ with confidential balances $(\bar{\mathbf{v}}'_1, \bar{\mathbf{v}}'_2)$, such that $(\bar{\mathbf{v}}_1 + \bar{\mathbf{v}}_2 = \bar{\mathbf{v}}'_1 + \bar{\mathbf{v}}'_2)$. Here, coin id'_1 is now held by the owner of the receiving account, who also learns the confidential amount $\bar{\mathbf{v}}'_1$.

Functionality $\mathcal{F}_{\underline{\mathsf{C}}\mathsf{Ledger}}$ exposes Mint and Redeem interfaces: a mint activation *locks* a public amount of tokens $\mathbb{T}$ and generates a fresh confidential token of the same balance. Conversely, a redeem activation will *release* the balance of a confidential coin back to the public ledger.

Realizing a Confidential Ledger. A confidential token is realized in protocol $\Pi_{\underline{\mathsf{C}}\mathsf{Ledger}}$ with Pedersen Commitments [44], described in full detail in the full version [6]. Let $g, g_1, ..., g_t, h$ denote generators of group $\mathbb{G}$ of safe prime order p, such that s_i in $g_i = g^{s_i}$ and w in $h = g^w$ are given by $\mathcal{F}_{\mathsf{Setup}}$ (parameterized with $g \in \mathbb{G}$) that publicly outputs $g_1, ..., g_t, h$. The commitment to a balance $\bar{\mathbf{v}} = (v_1, ..., v_t)$ over tokens $\mathbb{T}$ with blinding r is $\mathsf{com}(\bar{\mathbf{v}}, r) = \mathbf{g}^{\bar{\mathbf{v}}} h^r = g_1^{v_1} ... g_t^{v_t} h^r$. Pedersen commitments are additively homomorphic: $\mathsf{com}(\bar{\mathbf{v}}_1, r_1) \circ \mathsf{com}(\bar{\mathbf{v}}_2, r_2) = \mathsf{com}(\bar{\mathbf{v}}_1 + \bar{\mathbf{v}}_2, r_1 + r_2)$. Thus, during a confidential transfer, the sum equality between consumed input and freshly constructed output coin commitments holds

if total token balances are preserved and r'_1 and r'_2 are correlated such that $r_1 + r_2 = r'_1 + r'_2$.

$$\mathsf{com}(\bar{\mathbf{v}}_1, r_1) \circ \mathsf{com}(\bar{\mathbf{v}}_1, r_1) = \mathsf{com}(\bar{\mathbf{v}}'_1, r'_1) \circ \mathsf{com}(\bar{\mathbf{v}}'_2, r'_2) \tag{1}$$

However, since the equality above holds for any $\bar{\mathbf{v}}_1 + \bar{\mathbf{v}}_2 \equiv \bar{\mathbf{v}}'_1 + \bar{\mathbf{v}}'_2 \mod p$ and correlated r'_1, r'_2, an additional p units of each token in $\mathbb{T}$ can be minted: $\bar{v}_1 + \bar{v}_2 + p \equiv \bar{v}'_1 + \bar{v}'_2 \mod p$. Thus, each confidential token is associated with NIZK π which proves $\mathcal{R}(c; \bar{\mathbf{v}}, r) = \{c = \mathsf{com}(\bar{\mathbf{v}}, r) \wedge \bar{\mathbf{v}} \leq \bar{\mathbf{v}}_{\mathsf{max}} = 2^l - 1\}$, such that such wrap-around never occurs undetected.

We note that $\Pi_{\underline{\mathsf{C}}\mathsf{Ledger}}$ in itself affords a fully decentralized layer 2 confidential token transfer solution, since it is independent of the MPC servers. Thus allowing client's to send a receive confidential tokens in a peer-to-peer manner. This is needed to prevent leakage of exchange orders after-the-fact by analysing client's non-confidential tokens given as input and withdrawn as output from a privacy preserving smart contract execution. By allowing the privacy preserving smart contract executions to integrate in a greater payment ecosystem reasonably ensures that it is possible to hide token inputs and outputs from a privacy preserving smart contract execution by using them for confidential payment, similar to other confidential token systems.

We present a protocol $\Pi_{\underline{\mathsf{C}}\mathsf{Ledger}}$ which GUC-realizes $\mathcal{F}_{\underline{\mathsf{C}}\mathsf{Ledger}}$ in the full version [6], where we also prove the following statement:

Theorem 1. *Protocol $\Pi_{\underline{\mathsf{C}}\mathsf{Ledger}}$ GUC-realizes functionality $\mathcal{F}_{\underline{\mathsf{C}}\mathsf{Ledger}}$ in the $\mathcal{F}_{\mathsf{Clock}}$, $\mathcal{F}_{\mathsf{Ledger}}$, $\mathcal{F}_{\mathsf{NIZK}}$, $\mathcal{F}_{\mathsf{Setup}}$, $\mathcal{F}_{\mathsf{Sig}}$-hybrid model against any PPT-adversary corrupting any minority of committee $\mathcal{Q}$.*

3 Confidential Contracts

We present our formal model of confidential contracts. We assume m clients $\{C_1, \ldots, C_m\}$ and servers $\{P_1, \ldots, P_n\}$ that interact with $\mathcal{F}_{\underline{\mathsf{C}}\mathsf{Contract}}$, which extends $\mathcal{F}_{\underline{\mathsf{C}}\mathsf{Ledger}}$. For simplicity of presentation, we first present a single-round confidential contract functionality in Fig. 2, and subsequently illustrate how it is easily extended to a multi-round contract functionality where clients can selectively choose to participate in specific rounds.

The choice of modelling $\mathcal{F}_{\underline{\mathsf{C}}\mathsf{Contract}}$ as an extension of $\mathcal{F}_{\underline{\mathsf{C}}\mathsf{Ledger}}$ arises from the relation between underlying protocols: confidential coins in $\Pi_{\underline{\mathsf{C}}\mathsf{Ledger}}$ that are committed to a confidential contract evaluation must be *locked* and subsequently *replaced* by a new set of output coins reflecting a new distribution of balances, determined by $\Pi_{\underline{\mathsf{C}}\mathsf{Contract}}$. However, this requires verification operations over the homomorphic *commitment* representation of coins in $\Pi_{\underline{\mathsf{C}}\mathsf{Ledger}}$, which are not exposed by $\mathcal{F}_{\underline{\mathsf{C}}\mathsf{Ledger}}$.

We provide a brief sketch of the interface exposed by $\mathcal{F}_{\underline{\mathsf{C}}\mathsf{Ledger}}$. Upon initialization with an arithmetic circuit g encoding only the contract logic, users can enroll, specifying input string x and a confidential coin to input, identified by its

id. Upon a completed *Enroll*, the functionality is prompted by servers to evaluate circuit g on both client input strings, interpreted as numerical values, and input balances, with checks to ensure g does not mint tokens. $\mathcal{F}_{\underline{\mathsf{C}}\mathsf{Ledger}}$ permits clients to withdraw anytime to retrieve the private output string and output balance. $\mathcal{F}_{\underline{\mathsf{C}}\mathsf{Contract}}$ permits the simulator to abort and indicate cheating servers, which are then penalized by the functionality.

Functionality $\mathcal{F}_{\underline{\mathsf{C}}\mathsf{Contract}}$, extends $\mathcal{F}_{\underline{\mathsf{C}}\mathsf{Ledger}}$

$\mathcal{F}_{\underline{\mathsf{C}}\mathsf{Contract}}$ interacts with clients $\mathcal{C} = \{C_1, ..., C_m\}$ and servers $\mathcal{P} = \{P_1, ..., P_n\}$. The functionality exposes interfaces and and accesses the state of $\mathcal{F}_{\underline{\mathsf{C}}\mathsf{Ledger}}$. It is parameterized with max. circuit depth d_T, and collateral balance $\bar{\mathbf{v}}_{\mathsf{coll}}$.

Init: On (INIT, *sid*, g) from $P_i \in \mathcal{P}$ forward messages to $\mathcal{S}$. If $\mathcal{S}$ continues.
1. Run **GenAcct** and **Init** procedures on $\mathcal{F}_{\underline{\mathsf{C}}\mathsf{Ledger}}$.
2. Assert that g is a circuit and that $\mathsf{depth}(g) \leq d_T$, store g.
3. Assert $\mathsf{vk} \in \mathcal{K}[P_i]$ and $\mathcal{L}[\mathsf{vk}] \geq \bar{\mathbf{v}}_{\mathsf{coll}}$.
4. Set $\mathcal{L}[\mathsf{vk}] \leftarrow \mathcal{L}[\mathsf{vk}] - \bar{\mathbf{v}}_{\mathsf{coll}}$.
   - If all servers have successfully called **Init**, set state to `enroll`, tick $\mathcal{F}_{\mathsf{Clock}}$.

Enroll: Upon input (ENROLL, *sid*, x, id, vk) from client $C_j \in \mathcal{C}$,
1. Assert $\mathsf{vk} \in \mathcal{K}[C_j]$ and $\langle \mathsf{id}, \bar{\mathbf{v}} \rangle \in \mathcal{L}[\mathsf{vk}]$.
2. Forward (ENROLL, *sid*, id, vk) to $\mathcal{S}$, if $\mathcal{S}$ aborts, run **Abort**. Otherwise, continue.
3. Assert state is in `enroll` and $\exists \langle \mathsf{id}, \bar{\mathbf{v}} \rangle \in \mathcal{L}_{\mathsf{Conf}}[\mathsf{vk}]$: then remove $\langle \mathsf{id}, \bar{\mathbf{v}} \rangle$.
4. Store input $(x_j, \bar{\mathbf{v}}_j)$.
   - If all clients have successfully called **Enroll**, tick $\mathcal{F}_{\mathsf{Clock}}$.

Execute: Upon input (EXECUTE, *sid*) from $P_i \in \mathcal{P}$,
1. If EXECUTE received from all $\mathcal{P}$ and $\mathcal{F}_{\mathsf{Clock}}$ ticked since state update to `enroll`, forward (EXECUTE, *sid*) to $\mathcal{S}$ and wait for *Ok* or *Abort*. If *Ok*, continue.
   a. Evaluate circuit g over current user inputs $\{(x_j, \bar{\mathbf{v}}_j)\}_{j \in [m]}$ and client state.
   c. Store client states $\{(y_j, \bar{\mathbf{w}}_j)\}_{j \in [m]}$ read from output gates of g.
      - Assert $\sum_{j \in [m]} \bar{\mathbf{v}}_j = \sum_{j \in [m]} \bar{\mathbf{w}}_j$. Tick $\mathcal{F}_{\mathsf{Clock}}$.
2. Forward (EVALUATE, *sid*) to $\mathcal{S}$ and wait for *Ok* or *Abort*.
   - If *Ok* returned, set state to `evaluated` and tick $\mathcal{F}_{\mathsf{Clock}}$.
3. Send (OUTPUT, *sid*, $\{y_j, \bar{\mathbf{w}}_j\}_{j \in [m]}$) to $\mathcal{S}$ and wait for *Ok* or *Abort*.
   - If $\mathcal{S}$ aborts, it provides cheating server set $\mathcal{J}$, run **Abort** with $\mathcal{J}$.
4. For $j \in [m]$, get a unique id'_j from $\mathcal{S}$, and set $\mathcal{L}_{\mathsf{Conf}}[\mathsf{vk}_j] \leftarrow \mathcal{L}_{\mathsf{Conf}}[\mathsf{vk}_j] \cup \{\langle \mathsf{id}'_j, \bar{\mathbf{w}}_j \rangle\}$.
5. Set state to `enroll` and tick $\mathcal{F}_{\mathsf{Clock}}$.

Withdraw: Upon (WITHDRAW, *sid*) from $C_j \in \mathcal{C}$, obtain newly stored outputs since last *Withdraw* by $C_j \in \mathcal{C}$. Return $((y_{j,1}, \langle \mathsf{id}'_{j,1}, \bar{\mathbf{w}}_{j,1} \rangle), ..., (y_{j,l}, \langle \mathsf{id}'_{j,l}, \bar{\mathbf{w}}_{j,l} \rangle))$.

Abort: Tick $\mathcal{F}_{\mathsf{Clock}}$,
a. If state is `enroll`, return server and client funds: update $\mathcal{L}$, $\mathcal{L}_{\mathsf{Conf}}$.
b. Else if state in `evaluated`, obtain cheating servers $\mathcal{J}$ from $\mathcal{S}$:
   - If $\mathcal{J} \neq \emptyset$, reimburse clients $\mathcal{C}$ and honest servers $\mathcal{P} \backslash \mathcal{J}$, then distribute $\mathcal{J}$'s collateral amongst $\mathcal{C}$: update $\mathcal{L}$, $\mathcal{L}_{\mathsf{Conf}}$ accordingly.
   - Else if $\mathcal{J} = \emptyset$, obtain $\{(y_j, \bar{\mathbf{v}}_j)\}_{j \in [m]}$ from last evaluation of circuit g.
     - For $C_j \in \mathcal{C}'$, sample $\mathsf{id}_j \leftarrow\$ \mathbb{F}$ and set $\mathcal{L}_{\mathsf{Conf}}[\mathsf{vk}_j] \leftarrow \mathcal{L}_{\mathsf{Conf}}[\mathsf{vk}_j] \cup \{\langle \mathsf{id}_j, \bar{\mathbf{v}}_j \rangle\}$.
     - Return collateral for all $\mathcal{P}$: update $\mathcal{L}$ accordingly.
c. Terminate.

Fig. 2. Functionality for Confidential Contracts

Model of Confidential Contracts. Unlike public smart contracts deployed to $\mathcal{F}_{\mathsf{Ledger}}$, an instance of $\mathcal{F}_{\mathsf{Ident}}$ permits the computation of any arithmetic circuit on both private and public inputs. We model a confidential contract as an arithmetic circuit over a field $\mathbb{F}_p$ consistent with the domain that $\mathcal{F}_{\mathsf{Ident}}$ is realized with. A well-formed confidential contract permits the writing of both *numerical* and *financial* inputs from each client to its input gates. Further, we enforce a maximum circuit depth d_T prior to the circuit evaluation to bound the rounds of interaction in the MPC instance.

$$\big(([\,y_1\,],[\,\bar{\mathbf{w}}_1\,]),...,([\,y_m\,],[\,\bar{\mathbf{w}}_m\,])\big) \leftarrow \mathsf{eval}_g\big(([\,x_1\,],[\,\bar{\mathbf{v}}_1\,])....,([\,x_m\,],[\,\bar{\mathbf{v}}_m\,])\big)$$

Upon confidential evaluation of a contract circuit g with well-formed depth and gates, the following assertion must be performed at each run-time over confidential inputs and outputs of evaluated g: namely, that token supplies have been preserved.

$$\sum_{i\in[m]} [\,\bar{\mathbf{v}}_i\,] =^{?} \sum_{i\in[m]} [\,\bar{\mathbf{w}}_i\,] \tag{2}$$

One-Round Client-Server Interaction. Upon providing inputs to a confidential contract execution, clients can go off-line and retrieve confidential outputs with *Withdraw* at any later point in time.

Collateral. Our need for collateral follows the same logic as in Insured MPC [7]. The collateral contract incentivizes the servers to continue to participate in the privacy preserving smart contract computation, and behave honestly as they would otherwise suffer a financial loss. While the underlying maliciously secure MPC system will ensure that a server acting maliciously will cause an abort except with negligible probability, such an abort the adversary might have learned the output of the computation. This can in some situations have high value. Thus we require each server to give as collateral, strictly *more* than the maximum value they could gain from learning the output of a privacy preserving computation.

3.1 Realizing the Confidential Contract Functionality

Overview of Protocol. Having provided a high-level overview of the protocol phase in Sect. 1, we now proceed to detail the individual protocol phases for the single-round privacy preserving smart contract execution and refer to the full version [6] for the full protocol description and UC-security proof.

Setup of Contracts. Servers deploy instances of $\mathcal{X}_{\mathsf{Lock}}[\mathcal{X}_{\underline{\subseteq}\mathsf{Ledger}}]$, $\mathcal{X}_{\mathsf{Collateral}}$ on $\mathcal{F}_{\mathsf{Ledger}}$. Since wrapper $\mathcal{X}_{\mathsf{Lock}}$ extends $\mathcal{X}_{\underline{\subseteq}\mathsf{Ledger}}$, both are deployed and initialized as a single contract instance on $\mathcal{F}_{\mathsf{Ledger}}$ with shared contract id ($\mathsf{cn}_{\mathsf{Lock}}$) and shared state such as the confidential ledger ($\mathcal{L}_{\mathsf{Conf}}$). Here, the function of $\mathcal{X}_{\mathsf{Lock}}$ is to lock the confidential coins of clients input to the confidential contract evaluation, and to *replace* these with a new confidential distribution according to result of the contract evaluation. Further, $\mathcal{X}_{\mathsf{Lock}}$ is initialized with a threshold signature

verification key $\mathsf{vk}_{\mathsf{TSig}}$, jointly generated by all servers via $\mathcal{F}_{\mathsf{TSig}}$: whenever servers agree on a new status of the contract evaluation in $\mathcal{F}_{\mathsf{Ident}}$, this agreement can be settled in $\mathcal{X}_{\mathsf{Lock}}$ with a threshold signature jointly generated via global functionality $\mathcal{F}_{\mathsf{TSig}}$. $\mathcal{X}_{\mathsf{Collateral}}$ is parameterized by $\mathsf{cn}_{\mathsf{Lock}}$ and is activated each time $\mathcal{F}_{\mathsf{Clock}}$ progresses: it obtains collateral from all participating servers. It observes any recorded cheating servers $\mathcal{J}$ stored in the state of contract instance $\mathsf{cn}_{\mathsf{Lock}}$ and enforces penalties accordingly.

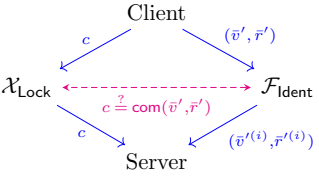
Client Enrollment. Clients interact with $\mathcal{X}_{\mathsf{Lock}}$ to enroll a confidential coin it controls to the contract evaluation, and send both the coin commitment opening and numerical input x to an instance of $\mathcal{F}_{\mathsf{Ident}}$. Enrolled coins are removed from the confidential ledger $\mathcal{L}_{\mathsf{Conf}}$ maintained by $\mathcal{X}^{\underline{\mathsf{CLedger}}}$ and moved to a dedicated ledger $\mathcal{L}_{\mathsf{Lock}}$ for funds committed to a pending MPC computation in $\mathcal{F}_{\mathsf{Ident}}$.

Clients must also commit to a *output mask* during enrollment, which enables the subsequent redistribution of confidential coins without client interaction in the output phase of the contract evaluation. Here each client with confidential coin input c and numerical input x performs the following:

- Samples $\hat{y} \leftarrow\$ \mathbb{F}$ as a numerical output mask and sends to $\mathcal{F}_{\mathsf{Ident}}$.
- Samples $\hat{\mathbf{w}} \leftarrow\$ \mathbb{F}^{|\mathbb{T}|}$, $\hat{s} \leftarrow\$ \mathbb{F}$, and computes mask commitment $\hat{c} \leftarrow \mathsf{com}(\hat{\mathbf{w}}, \hat{s})$.
- Sends mask commitment $\hat{c}$ to $\mathcal{X}_{\mathsf{Lock}}$ on $\mathcal{F}_{\mathsf{Ledger}}$.
- Sends mask commitment openings $(\hat{\mathbf{w}}, \hat{s})$ of $\hat{c}$ to $\mathcal{F}_{\mathsf{Ident}}$.

Here clients can also give any auxiliary input, x, needed for the privacy preserving smart contract computation.

Input Verification. Upon enrollment of clients, servers must verify that the confidential coin c and mask commitment $\hat{c}$ sent to $\mathcal{X}_{\mathsf{Lock}}$ are consistent with their respective openings $(\bar{\mathbf{v}}, \bar{r})$ and $(\hat{\mathbf{v}}, \hat{r})$ sent to $\mathcal{F}_{\mathsf{Ident}}$ during enrollment. For simplicity of presentation, we illustrate the batched input verification of input confidential coins and their openings assuming a token universe size of $|\mathbb{T}| = 1$, such that $c = g^{\bar{v}} h^{\bar{r}}$. Input verification for output masks $\hat{c}$ and their openings submitted to $\mathcal{F}_{\mathsf{Ident}}$ follow similarly.



Each server obtains both confidential coin c from $\mathcal{X}_{\mathsf{Lock}}$ and *additive shares* of submitted openings thereof from $\mathcal{F}_{\mathsf{Ident}}$, namely $(\bar{v}'^{(i)}, \bar{r}'^{(i)})$. We write $\bar{v}'^{(i)} = (\bar{v} + \epsilon)^{(i)}$ and similarly for $\bar{r}'^{(i)}$, where the ϵ denotes the error or discrepancy that the adversary can introduce to $\bar{v}$. We employ a standard technique of evaluating a random linear combination over client inputs to verify consistency.

1. Servers jointly sample $\gamma, \alpha, \beta \leftarrow\$ \mathbb{F}$ and open γ.
2. Each server locally computes the following on the inputs from m clients.
   - $\bar{v}'^{(i)}_{\mathsf{lin}} = \alpha^{(i)} + \gamma\, \bar{v}'^{(i)}_1 + \ldots + \gamma^m\, \bar{v}'^{(i)}_m$ and $r'^{(i)}_{\mathsf{lin}} = \beta^{(i)} + \gamma\, r'^{(i)}_1 + \ldots + \gamma^m\, r'^{(i)}_m$
   - Subsequently, it sends $\bar{v}'^{(i)}_{\mathsf{lin}}$ and $\bar{v}'^{(i)}_{\mathsf{lin}}$ to all other servers.
3. Each server locally reconstructs $\bar{v}'_{\mathsf{lin}} = \prod_{i \in [n]} \bar{v}'^{(i)}_{\mathsf{lin}}$ and $r'_{\mathsf{lin}} = \prod_{i \in [n]} r'^{(i)}_{\mathsf{lin}}$

4. Servers locally verify: $\prod_{i\in[n]} g^{\alpha^{(i)}} h^{\beta^{(i)}} \prod_{j\in[m]} c_j^{\gamma^j} \stackrel{?}{=} g^{\bar{v}'_{\mathsf{lin}}} h^{r'_{\mathsf{lin}}}$

Note that $\bar{v}'^{(i)}_{\mathsf{lin}}$ and $r'^{(i)}_{\mathsf{lin}}$ are shares held by servers and do not reveal the values of user inputs. We write $\bar{v}'_{\mathsf{lin}} = \alpha + \gamma\,(\bar{v}_1 + \epsilon_{\bar{v}_1}) + \ldots + \gamma^m\,(\bar{v}_m + \epsilon_{\bar{v}_m})$ and similarly for r'_{lin} to expose ϵ's introduced by the adversary. If ϵ values are committed to by the adversary before α, β, γ are sampled, we can interpret $\bar{v}'_{\mathsf{lin}} - \bar{v}_{\mathsf{lin}} = 0$ and $r'_{\mathsf{lin}} - r_{\mathsf{lin}} = 0$ as m - degree polynomials with coefficients chosen by the adversary that are later evaluated at some random coordinate γ: since verification step (4) implies exactly these assertions, the probability for an undetected non-zero error is therefore $m/|\mathbb{G}|$, where m is the number of polynomial roots, by the Schwartz-Zippel Lemma.

Execute. Servers call the **Evaluate** interface on $\mathcal{F}_{\mathsf{Ident}}$ to evaluate circuit g with input gates set to client inputs.

$$([x_1], [\hat{y}_1], [\bar{\mathbf{v}}_1], [r_1], [\hat{\mathbf{w}}_1], [\hat{s}_1]), ..., ([x_m], [\hat{y}_m], [\bar{\mathbf{v}}_m], [r_m], [\hat{\mathbf{w}}_m], [\hat{s}_m])$$

Upon secure evaluation, outputs in form of numerical values and balances are written to the output gates of g: $\big(([y_1], [\bar{\mathbf{w}}_1]), ..., ([y_m], [\bar{\mathbf{w}}_m])\big)$. Before masking these for opening, the servers then perform a confidential consistency check to ensure the preservation of tokens as shown in Eq. (2).

Masked output values are obtained by applying the masking values input by users, $[y'_j] = [y_j] + [\hat{y}_j]$ and similarly for balances, $[\bar{\mathbf{w}}'_j] = [\bar{\mathbf{w}}_j] + [\hat{\mathbf{w}}_j]$ and generating a joint signature $\sigma_{\mathsf{vk}_{\mathsf{TSig}}}(\mathsf{evaled})$ via $\mathcal{F}_{\mathsf{TSig}}$, that is sent to $\mathcal{X}_{\mathsf{Lock}}$ on $\mathcal{F}_{\mathsf{Ledger}}$. Upon verification, the $\mathcal{X}_{\mathsf{Lock}}$ contract updates the state of protocol execution, reflecting completion of the *Execute* phase.

Open. Servers run **Optimistic Reveal** in $\mathcal{F}_{\mathsf{Ident}}$ to open masked numerical outputs and balances $\big((y'_1, \bar{\mathbf{w}}'_1), ..., (y'_m, \bar{\mathbf{w}}'_m)\big)$. Should all servers agree on the successful completion of the contract evaluation, they jointly sign all *masked* outputs and send these to $\mathcal{X}_{\mathsf{Lock}}$ (on $\mathcal{F}_{\mathsf{Ledger}}$), which then computes the *unmasked* confidential coins for clients with the newly computed distribution as follows. Given the masked output balance $\bar{\mathbf{w}}'$ from $\mathcal{F}_{\mathsf{Ident}}$ and the coin mask $\hat{c}$ sampled by a client in **Enroll**, contract $\mathcal{X}_{\mathsf{Lock}}$ computes

(a) The masked confidential coin: $c^{\mathsf{out}\,\prime} \leftarrow \mathbf{g}^{\bar{\mathbf{w}}'} h^0$
(b) The unmasked confidential coin: $c^{\mathsf{out}} \leftarrow c^{\mathsf{out}\,\prime} \cdot \hat{c}^{-1}$

We rewrite (b) as $c^{\mathsf{out}} = \mathbf{g}^{\bar{\mathbf{w}}' - \hat{\mathbf{w}}} h^{-\hat{s}} = \mathsf{com}(\bar{\mathbf{w}}, -\hat{s})$ to expose the unmasking of the output coin without any knowledge of the final balance. $\mathcal{X}_{\mathsf{Lock}}$ subsequently stores unmasked output coin c^{out} in the confidential ledger in $\mathcal{X}_{\underline{\mathsf{C}}\mathsf{Ledger}}$, thereby settling the output balance distribution read from output gates of contract circuit g. Should $\mathcal{X}_{\mathsf{Lock}}$ successfully verify the signed outputs, $\mathcal{X}_{\mathsf{Collateral}}$ will infer from the state of $\mathcal{X}_{\mathsf{Lock}}$ the completion of a successful round and return the deposited collateral to the servers.

Withdraw. Upon a successful **Open**, the output of the confidential contract evaluation has completed. Each client can obtain their masked output $(y', \bar{\mathbf{w}}')$

from $\mathcal{X}_{\mathsf{Lock}}$ and newly minted c_{out} from $\mathcal{X}_{\underline{\mathsf{C}}\mathsf{Ledger}}$ anytime following a successful execution of a contract evaluation. Let $\hat{y}$ and $(\hat{\mathbf{w}}, \hat{s})$ be the output masks generated by the client in **Enroll**. The withdrawing client obtains

(a) The numerical output: $y \leftarrow y' - \hat{y}$
(b) The opening of the output coin: $(\bar{\mathbf{w}}, s) \leftarrow (\bar{\mathbf{w}}' - \hat{\mathbf{w}}, -\hat{s})$

Thus, their the tokens are still confidential and that clients can transfer or redeem these using $\Pi_{\underline{\mathsf{C}}\mathsf{Ledger}}$ described in the full version [6].

Abort. If the protocol aborts prior to the completion of the **Execute** phase, client funds are simply returned by $\mathcal{X}_{\mathsf{Lock}}$ and collateral deposited to $\mathcal{X}_{\mathsf{Collateral}}$ is returned. If servers have agreed upon the completion of **Execute**, honest servers can interact with $\mathcal{F}_{\mathsf{Ident}}$ to either (a) obtain shares that are verifiable and enable reconstruction of the output or (b) identify cheating servers (functionality described in the full version [6]). Thus, $\mathcal{X}_{\mathsf{Lock}}$ as a registered public verifier, can identify cheating servers by either verifying shares with $\mathcal{F}_{\mathsf{Ident}}$, or obtaining the identities of servers $\mathcal{J}$ that refuse to participate in revealing their shares and allowing their verification. Cheating servers lose their collateral held by $\mathcal{X}_{\mathsf{Collateral}}$ which is redistributed to clients.

We present the full protocol $\Pi_{\underline{\mathsf{C}}\mathsf{Contract}}$ which GUC-realizes $\mathcal{F}_{\underline{\mathsf{C}}\mathsf{Contract}}$ in the full version [6] and prove the following statement.

Theorem 2. *$\Pi_{\underline{\mathsf{C}}\mathsf{Contract}}[\Pi_{\underline{\mathsf{C}}\mathsf{Ledger}}]$ realizes $\mathcal{F}_{\underline{\mathsf{C}}\mathsf{Contract}}[\mathcal{F}_{\underline{\mathsf{C}}\mathsf{Ledger}}]$ in the $\mathcal{F}_{\mathsf{Clock}}$, $\mathcal{F}_{\mathsf{Ident}}$, $\mathcal{F}_{\mathsf{Ledger}}$, $\mathcal{F}_{\mathsf{NIZK}}$, $\mathcal{F}_{\mathsf{Setup}}$, $\mathcal{F}_{\mathsf{Sig}}$, $\mathcal{F}_{\mathsf{TSig}}$-hybrid model against any PPT-adversary corrupting at most $n-1$ of the n servers $\mathcal{P}$ statically and any minority of $\mathcal{Q}$.*

4 Efficiency

We note that since previous works focus on using zero knowledge proofs and a trusted contract manager, we refrain from directly comparing our efficiency to their works. The closest previous works to ours is the Hawk family [4,5,37]. Unfortunately neither of the works provide an efficiency analysis, making it hard to provide a meaningful comparison. However, we note they all require computation of cryptographic primitives (commitments and ZKPs) in MPC. Thus requiring strictly more MPC computation, *along* with a larger (and hence) slower field of computation, as this field is needed to facilitate *computational* security of the cryptographic primitives they compute in MPC. In the following analysis, we assume Bulletproofs for range proofs and standard Fiat-Shamir Schnorr proofs of knowledge of exponents using elliptic curves. Although neither of these are UC-secure since knowledge extraction requires rewinding, there is evidence [29] that these techniques can be made non-malleable in the algebraic group model. Hence, for the purpose of efficiency we believe it is reasonable to forgo the formal UC security in this section. We use BLS threshold signatures and for simplicity we assume the size of the group used for BLS and commitments is the same, although it will in practice be slightly larger for BLS.

Table 2. Complexity of our protocol when executing one $\underline{\mathsf{C}}\mathsf{Contract}$, *excluding* the computation of contract circuit g in MPC. We assume $|\mathcal{C}|z > s$ for statistical security parameter s, where z is the amount of input/output for each client in the set of clients $\mathcal{C}$, including the hidden token amount. $n = |\mathcal{P}|$ is the amount of servers and **mult** denotes the number of multiplications in MPC.

		Init	Execution	Abort
User	**exp**	2	2	0
Server	**exp**	$2 + 2(n-1)$	$6\|\mathcal{C}\| + 2$	0
	pair	0	$n - 1$	0
	mult	0	$z\|\mathcal{C}\|$	0
SC comp.	**exp**	0	$2\|\mathcal{C}\|z$	$\|\mathcal{C}\|$
	pair	0	2	0
SC call space	$\#\mathbb{G}$ elem.	3	$\|\mathcal{C}\|z$	$O(n\|\mathcal{C}\|z)$
Comm	$\#\mathbb{G}$ elem.	$O(n)$	$O(n^2 \cdot z \cdot \|\mathcal{C}\|)$	$O(n^2 \cdot z \cdot \|\mathcal{C}\|)$

We outline the amount of heavy computations needed for our core protocol in Table 2, *except* what is needed by the underlying MPC computation computing the contract circuit g, reflecting the privacy preserving smart contract $\underline{\mathsf{C}}\mathsf{Contract}$. Concretely we count the amount of group exponentiations when assuming that the Pedersen commitments are realized using elliptic curves, along with pairings assuming BLS [13] has been used for realizing distributed signatures. The table only contains the complexity of executing one instance of $\underline{\mathsf{C}}\mathsf{Contract}$, but we note that execution of multiple contracts is slightly sublinear in the complexity of a single execution. The *Abort* column illustrates the *additional* overhead associated with a cheating party.

Table 3. Complexity of $\underline{\mathsf{C}}\mathsf{Ledger}$ in group exponentiation and amount of group elements stored, when $\bar{v}_{\mathsf{max}}$ is the maximum amount of allowed tokens (Recall $|\mathcal{C}| \cdot \bar{v}_{\mathsf{max}} < |\mathbb{G}|$).

	Mint	ConfTransfer	Redeem
User	4	$O(\log(\bar{v}_{\mathsf{max}}) \cdot \log(\log(\bar{v}_{\mathsf{max}})))$	3
SC comp	3	$O(\log(\bar{v}_{\mathsf{max}}))$	3
SC space	3	$2\log(\bar{v}_{\mathsf{max}}) + 10$	4

When it comes to our confidential token layer, we outline the complexity in Table 3. We note that the constant in the complexity of *Confidential Transfer* reflects two range proofs over $\log(|\mathbb{G}|/2)$, under the assumption that BulletProofs are used [16]. Although if the domain of the token amounts is further limited from $\mathbb{G}$ to $\bar{v}_{\mathsf{max}} < |\mathbb{G}|/|\mathcal{C}|$ then they can be reduced to range proofs of $[0; \bar{v}_{\mathsf{max}} - 1]$ and thus complexity $O(\bar{v}_{\mathsf{max}} \cdot \log(\bar{v}_{\mathsf{max}}))$.

In both tables the amount of smart contract space is only what needs to be submitted. The persistent space use needed is only $3+3|\mathcal{C}|$ group elements, if we assume that the storage used when posting to $\mathcal{X}_{\mathsf{Lock}}$ in *evaluate* and *open* gets overwritten the next time the servers call these methods.

The round complexity for all steps of both the confidential token layer protocols and our core protocol is constant, assuming g has constant multiplicative depth. Otherwise, the computation of g dominates the round complexity.

References

1. Andrychowicz, M., Dziembowski, S., Malinowski, D., Mazurek, Ł: Fair two-party computations via bitcoin deposits. In: Böhme, R., Brenner, M., Moore, T., Smith, M. (eds.) FC 2014. LNCS, vol. 8438, pp. 105–121. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-44774-1_8
2. Andrychowicz, M., Dziembowski, S., Malinowski, D., Mazurek, L.: Secure multiparty computations on bitcoin. In: 2014 IEEE Symposium on Security and Privacy, pp. 443–458. IEEE Computer Society Press (2014). https://doi.org/10.1109/SP.2014.35
3. Badertscher, C., Maurer, U., Tschudi, D., Zikas, V.: Bitcoin as a transaction ledger: a composable treatment. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017. LNCS, vol. 10401, pp. 324–356. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63688-7_11
4. Banerjee, A., Clear, M., Tewari, H.: zkhawk: practical private smart contracts from mpc-based hawk. In: 2021 3rd Conference on Blockchain Research & Applications for Innovative Networks and Services (BRAINS), pp. 245–248. IEEE (2021). https://doi.org/10.1109/BRAINS52497.2021.9569822
5. Banerjee, A., Tewari, H.: Multiverse of HawkNess: A Universally-Composable MPC-based Hawk Variant. Cryptology ePrint Archive (2022). https://eprint.iacr.org/2022/421
6. Baum, C., yu Chiang, J.H., David, B., Frederiksen, T.K.: Eagle: efficient privacy preserving smart contracts. Cryptology ePrint Archive, Paper 2022/1435 (2022). https://eprint.iacr.org/2022/1435,
7. Baum, C., David, B., Dowsley, R.: Insured MPC: efficient secure computation with financial penalties. In: Bonneau, J., Heninger, N. (eds.) FC 2020. LNCS, vol. 12059, pp. 404–420. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-51280-4_22
8. Baum, C., David, B., Dowsley, R., Nielsen, J.B., Oechsner, S.: CRAFT: composable randomness and almost fairness from time. Cryptology ePrint Archive, Report 2020/784 (2020). https://eprint.iacr.org/2020/784
9. Baum, C., David, B., Frederiksen, T.K.: P2DEX: privacy-preserving decentralized cryptocurrency exchange. In: Sako, K., Tippenhauer, N.O. (eds.) ACNS 2021. LNCS, vol. 12726, pp. 163–194. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-78372-3_7
10. Benhamouda, F., Halevi, S., Halevi, T.: Supporting private data on hyperledger fabric with secure multiparty computation. IBM J. Res. Dev. **63**(2/3), 1–3 (2019). https://doi.org/10.1147/JRD.2019.2913621
11. Bentov, I., Kumaresan, R.: How to use bitcoin to design fair protocols. In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014. LNCS, vol. 8617, pp. 421–439. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-44381-1_24

12. Bentov, I., Kumaresan, R., Miller, A.: Instantaneous decentralized poker. In: Takagi, T., Peyrin, T. (eds.) ASIACRYPT 2017. LNCS, vol. 10625, pp. 410–440. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70697-9_15
13. Boneh, D., Lynn, B., Shacham, H.: Short signatures from the Weil pairing. J. Cryptol. **17**(4), 297–319 (2004). https://doi.org/10.1007/s00145-004-0314-9
14. Bowe, S., Chiesa, A., Green, M., Miers, I., Mishra, P., Wu, H.: ZEXE: enabling decentralized private computation. In: 2020 IEEE Symposium on Security and Privacy, pp. 947–964. IEEE Computer Society Press (2020). https://doi.org/10.1109/SP40000.2020.00050
15. Bünz, B., Agrawal, S., Zamani, M., Boneh, D.: Zether: towards privacy in a smart contract world. In: Bonneau, J., Heninger, N. (eds.) FC 2020. LNCS, vol. 12059, pp. 423–443. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-51280-4_23
16. Bünz, B., Bootle, J., Boneh, D., Poelstra, A., Wuille, P., Maxwell, G.: Bulletproofs: short proofs for confidential transactions and more. In: 2018 IEEE Symposium on Security and Privacy, pp. 315–334. IEEE Computer Society Press (2018). https://doi.org/10.1109/SP.2018.00020
17. Canetti, R.: Universally composable security: a new paradigm for cryptographic protocols. In: 42nd FOCS, pp. 136–145. IEEE Computer Society Press (2001). https://doi.org/10.1109/SFCS.2001.959888
18. Canetti, R.: Universally composable security: a new paradigm for cryptographic protocols. In: Proceedings 42nd IEEE Symposium on Foundations of Computer Science, pp. 136–145. IEEE (2001), https://doi.org/10.1109/SFCS.2001.959888
19. Canetti, R.: Universally composable signature, certification, and authentication. In: 17th IEEE Computer Security Foundations Workshop, (CSFW-17 2004), 28–30 June 2004, Pacific Grove, CA, USA, p. 219. IEEE Computer Society (2004). https://doi.org/10.1109/CSFW.2004.24, http://doi.ieeecomputersociety.org/10.1109/CSFW.2004.24
20. Canetti, R., Dodis, Y., Pass, R., Walfish, S.: Universally composable security with global setup. In: Vadhan, S.P. (ed.) TCC 2007. LNCS, vol. 4392, pp. 61–85. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-70936-7_4
21. Cheng, R., et al.: Ekiden: a platform for confidentiality-preserving, trustworthy, and performant smart contracts. In: 2019 IEEE European Symposium on Security and Privacy (EuroS&P) (2019). https://doi.org/10.1109/EuroSP.2019.00023
22. Choudhuri, A.R., Green, M., Jain, A., Kaptchuk, G., Miers, I.: Fairness in an unfair world: fair multiparty computation from public bulletin boards. In: Thuraisingham, B.M., Evans, D., Malkin, T., Xu, D. (eds.) ACM CCS 2017. pp. 719–728. ACM Press (2017). https://doi.org/10.1145/3133956.3134092
23. Daian, P., et al.: Flash boys 2.0: frontrunning in decentralized exchanges, miner extractable value, and consensus instability. In: 2020 IEEE Symposium on Security and Privacy, pp. 910–927. IEEE Computer Society Press (2020). https://doi.org/10.1109/SP40000.2020.00040
24. Damgård, I., Damgård, K., Nielsen, K., Nordholt, P.S., Toft, T.: Confidential benchmarking based on multiparty computation. In: Grossklags, J., Preneel, B. (eds.) FC 2016. LNCS, vol. 9603, pp. 169–187. Springer, Heidelberg (Feb (2016). https://doi.org/10.1007/978-3-662-54970-4_10
25. Damgård, I., Keller, M., Larraia, E., Pastro, V., Scholl, P., Smart, N.P.: Practical covertly secure MPC for dishonest majority – or: breaking the SPDZ limits. In: Crampton, J., Jajodia, S., Mayes, K. (eds.) ESORICS 2013. LNCS, vol. 8134, pp. 1–18. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40203-6_1

26. Damgård, I., Pastro, V., Smart, N., Zakarias, S.: Multiparty computation from somewhat homomorphic encryption. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 643–662. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-32009-5_38
27. David, B., Dowsley, R., Larangeira, M.: Kaleidoscope: an efficient poker protocol with payment distribution and penalty enforcement. In: Meiklejohn, S., Sako, K. (eds.) FC 2018. LNCS, vol. 10957, pp. 500–519. Springer, Heidelberg (2018). https://doi.org/10.1007/978-3-662-58387-6_27
28. David, B., Gentile, L., Pourpouneh, M.: FAST: fair auctions via secret transactions. In: Ateniese, G., Venturi, D. (eds.) ACNS 2022. LNCS, vol. 13269, pp. 727–747. Springer, Heidelberg (Jun 2022). https://doi.org/10.1007/978-3-031-09234-3_36
29. Ganesh, C., Orlandi, C., Pancholi, M., Takahashi, A., Tschudi, D.: Fiat-shamir bulletproofs are non-malleable (in the algebraic group model). In: Dunkelman, O., Dziembowski, S. (eds.) EUROCRYPT 2022, Part II. LNCS, vol. 13276, pp. 397–426. Springer, Heidelberg (2022). https://doi.org/10.1007/978-3-031-07085-3_14
30. Groth, J., Ostrovsky, R., Sahai, A.: New techniques for noninteractive zero-knowledge. J. ACM (JACM) **59**(3), 1–35 (2012). https://doi.org/10.1145/2220357.2220358
31. Jakobsen, T.P., Nielsen, J.B., Orlandi, C.: A framework for outsourcing of secure computation. In: Ahn, G., Oprea, A., Safavi-Naini, R. (eds.) Proceedings of the 6th edition of the ACM Workshop on Cloud Computing Security, CCSW 2014, Scottsdale, Arizona, USA, 7 November 2014, pp. 81–92. ACM (2014). https://doi.org/10.1145/2664168.2664170
32. Kalodner, H.A., Goldfeder, S., Chen, X., Weinberg, S.M., Felten, E.W.: Arbitrum: scalable, private smart contracts. In: Enck, W., Felt, A.P. (eds.) USENIX Security 2018, pp. 1353–1370. USENIX Association (Aug 2018)
33. Kanjalkar, S., Zhang, Y., Gandlur, S., Miller, A.: Publicly auditable mpc-as-a-service with succinct verification and universal setup. In: IEEE European Symposium on Security and Privacy Workshops, EuroS&P 2021, Vienna, Austria, 6–10 September 2021, pp. 386–411. IEEE (2021). https://doi.org/10.1109/EuroSPW54576.2021.00048
34. Katz, J., Maurer, U., Tackmann, B., Zikas, V.: Universally composable synchronous computation. In: Sahai, A. (ed.) TCC 2013. LNCS, vol. 7785, pp. 477–498. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-36594-2_27
35. Kerber, T., Kiayias, A., Kohlweiss, M.: KACHINA - foundations of private smart contracts. In: Küsters, R., Naumann, D. (eds.) CSF 2021 Computer Security Foundations Symposium, pp. 1–16. IEEE Computer Society Press (2021). https://doi.org/10.1109/CSF51468.2021.00002
36. Kiayias, A., Zhou, H.-S., Zikas, V.: Fair and robust multi-party computation using a global transaction ledger. In: Fischlin, M., Coron, J.-S. (eds.) EUROCRYPT 2016. LNCS, vol. 9666, pp. 705–734. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49896-5_25
37. Kosba, A.E., Miller, A., Shi, E., Wen, Z., Papamanthou, C.: Hawk: the blockchain model of cryptography and privacy-preserving smart contracts. In: 2016 IEEE Symposium on Security and Privacy, pp. 839–858. IEEE Computer Society Press (May 2016). https://doi.org/10.1109/SP.2016.55
38. Kumaresan, R., Bentov, I.: Amortizing secure computation with penalties. In: Weippl, E.R., Katzenbeisser, S., Kruegel, C., Myers, A.C., Halevi, S. (eds.) ACM CCS 2016, pp. 418–429. ACM Press (2016). https://doi.org/10.1145/2976749.2978424

39. Kumaresan, R., Moran, T., Bentov, I.: How to use bitcoin to play decentralized poker. In: Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, pp. 195–206 (2015). https://doi.org/10.1145/2810103.2813712
40. Kumaresan, R., Vaikuntanathan, V., Vasudevan, P.N.: Improvements to secure computation with penalties. In: Weippl, E.R., Katzenbeisser, S., Kruegel, C., Myers, A.C., Halevi, S. (eds.) ACM CCS 2016, pp. 406–417. ACM Press (2016). https://doi.org/10.1145/2976749.2978421
41. Lee, J., Nikitin, K., Setty, S.T.V.: Replicated state machines without replicated execution. In: 2020 IEEE Symposium on Security and Privacy, pp. 119–134. IEEE Computer Society Press (2020). https://doi.org/10.1109/SP40000.2020.00068
42. Nilsson, A., Bideh, P.N., Brorsson, J.: A survey of published attacks on intel SGX. CoRR abs/ arXiv: 2006.13598 (2020)
43. Ozdemir, A., Boneh, D.: Experimenting with collaborative zk-SNARKs: Zero-knowledge proofs for distributed secrets. Cryptology ePrint Archive, Report 2021/1530 (2021). https://eprint.iacr.org/2021/1530
44. Pedersen, T.P.: Non-interactive and information-theoretic secure verifiable secret sharing. In: Feigenbaum, J. (ed.) CRYPTO 1991. LNCS, vol. 576, pp. 129–140. Springer, Heidelberg (1992). https://doi.org/10.1007/3-540-46766-1_9
45. Abe, M., Ohkubo, M., Suzuki, K.: 1-out-of-n signatures from a variety of keys. In: Zheng, Y. (ed.) ASIACRYPT 2002. LNCS, vol. 2501, pp. 415–432. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-36178-2_26
46. Steffen, S., Bichsel, B., Baumgartner, R., Vechev, M.: ZeeStar: private Smart Contracts by Homomorphic Encryption and Zero-knowledge Proofs. In: 2022 IEEE Symposium on Security and Privacy (SP), pp. 1543–1543. IEEE Computer Society (2022). https://files.sri.inf.ethz.ch/website/papers/sp22-zeestar.pdf
47. Steffen, S., Bichsel, B., Gersbach, M., Melchior, N., Tsankov, P., Vechev, M.T.: zkay: specifying and enforcing data privacy in smart contracts. In: Cavallaro, L., Kinder, J., Wang, X., Katz, J. (eds.) ACM CCS 2019, pp. 1759–1776. ACM Press (2019). https://doi.org/10.1145/3319535.3363222
48. Team, T.S.N.: Secret network: a privacy-preserving secret contract & decentralized application platform (2022). https://scrt.network/graypaper