Work-in-Progress: an Approach to Computable Contracts with Verifiable Computation Outsourcing and Blockchain Transactions

Carlo Brunetta[1], Amit Chaudhary[2,3], Stefano Galatolo[4], and Massimiliano Sala[5,6]

[1] ShunyaXifra, cbrunetta@shunyax.net
[2] University of Warwick, UK, amit.chaudhary.3@warwick.ac.uk
[3] Palliora, amit@palliora.org
[4] University of Pisa, Italy stefano.galatolo@unipi.it
[5] University of Trento, Italy, massimiliano.sala@unitn.it,
[6] ShunyaXifra, msala@shunyax.net

Abstract. We present our work-in-progress approach to computable contracts, where all roles in a computation may be outsourced, from the servers performing computations, to those providing input, to those performing verifications (on input and on output), including all related communications. Varying levels of confidentiality can be chosen on both data and calculations. Although the largest part of the computational and communication effort is performed off-chain, our contracts require a specialized underlying blockchain where they are encoded as transactions. This enables decentralized handling and enforces correct execution through a combination of cryptographic techniques and economic security. Our delegation architecture allows for the execution of very complex collaborative tasks, such as decentralized AI.

Keywords: Verifiable Computation · Decentralized AI · Blockchain · Distributed Protocol · Cryptography

1 Introduction

In modern society there is a growing need for collaborative computations on a massive scale, albeit with a series of strict requirements that make this collaboration very challenging. A striking example is a sought-after AI marketplace, where all AI actors would request high confidentiality and privacy, as well as a fair reward for their contribution to the final result, including model providers, training-data providers, inference providers, verification providers (verification on the quality of input, verification on the quality of output, etc.) and many others. Apart from centralized solutions (with an unacceptable level of trust in one entity), most (decentralized) solutions use smart-contracts (e.g., in Ethereum [6]) as main bricks and then add other components to achieve different goals, such as verifiability with ad hoc systems (e.g. TrueBit [35]), or scalability with oracles [8, 27] and/or bridges [14, 33, 26] and/or second-layers [32, 29, 20], in order to delegate heavy computations and massive input/output handling.

Our approach is different. We present computable contracts, whose design natively regulates the outsourcing of all roles in a computation, from the servers performing computations, to those providing input, to those performing verifications (on input and on output), including all related communications. Varying levels of confidentiality can be chosen, both on data and calculations. While the largest part of the computational and communication effort is performed off-chain, our contracts require a *specialized* underlying blockchain to encode them via transactions. This allows us to achieve contracts' decentralized handling and enforce their correct execution via a combination of cryptographic techniques and economic security. More precisely, the contract is encoded as a series of on-chain transactions, allowing the underlying blockchain not only to store the agreed final version of the contract, but also to track all contract's execution.

Our contracts fully describe the (verifiable) computational interaction, rather than focussing on the computation itself. Indeed, the major complexity in our design lies in the coordination among actors that do not trust (fully) each other, while fairly rewarding them for their data/services. We build our solution over a blockchain to inherit a natural payment method and an immutable public ledger, simultaneously facilitating the negotiation phase. Palliora's contracts [24] are an instantiation of our approach, focussed on decentralized AI.

1.1 Our contributiom

The notion of *computable contract* is already considered in legal/insurance scenarios [13] where formal-computations can be used to force contractual terms. To achieve a realistic outcome, we design our computable contracts assuming that they can avail of an underlying blockchain with specific properties. From a blockchain point of view, they can thus be viewed as a hybrid object with characteristics in between simple transactions and smart-contracts. In other words, computable contracts allow interaction via transactions, yet their content is a predefined form of data-fields which does not include the code to be executed. This approach significantly reduces contractual-computational costs with respect to smart-contracts, since the bulk of computations is done off-chain.

The remainder of this paper is organized as follows. Section 2 introduces the main actors considered in our solution, describes their assumptions, their role and their goal. In Section 3, we first sketch how the contract is defined and then we explain how the contract is executed. Section 4 provides some final considerations on the contracts' properties, on their security and applicability, as well as on their possible extensions and other interesting topics.

1.2 Related Works

Goodenough *et al.* [13] introduce the notion of *computable contract* as a legally-binding agreement between parties in the legal/insurance scenario. Their idea is to introduce special clauses into contracts that can be *formally* evaluated, e.g. a payment might be released only when an agreed-equation has output in a desired

interval. Notably, their contracts are expressed in natural-language to differentiate them from *automated* computable contracts, which are instead described entirely as code. Our computable contracts might be seen as an instantiation of their notion within a blockchain environment. With a totally different philosophy, smart-contracts (e.g. Ethereum [6], Solana [31], Tezos [34]) have been proposed as a powerful decentralized methodology to define and evaluate contracts written as code (i.e., "automated computable contracts" in Goodenough *et al.* [13]'s terminology). Smart contracts are defined by their code, and transactions provide the means to interact with their execution (and to trigger it). However, fully executing smart contracts on-chain in a naïve manner leads to substantial computational overhead and high economic costs.

Currently, there are two existing approaches to reduce the huge cost of (naive) smart-contract execution: scaling solutions and (partial) computation delegation. As regards scaling solutions, we cite bridges [14, 33, 26] and second layers [32, 29, 20, 1]. These solutions represent a more general research line to tackle the issue of efficiency and cost of running a public blockchain, but they cannot compete with solutions such as ours, which is entirely focussed on shrinking the execution costs to a minimum.

Regarding computation delegations, we cite oracles [8, 27] (though they were invented for other reasons) and challenge-based solutions, e.g. Truebit [35]. The difference between these and our approach cannot be found at an efficiency/cost reduction level. Indeed, (blockchain) computation delegation is meant as something to be used inside a smart contract, or called by a smart contract, or in any other interaction with the virtual machine of a blockchain, while our computable contracts are meant to do the opposite, that is, they use a (specifically designed) blockchain to perform computations which, a priori, have nothing to do with what happens in a target blockchain or its virtual machine. While our computable contracts can indeed be employed to offload computation from congested or economically expensive blockchains, this application constitutes just one of many possible use cases. The actual limitation of our approach is its own freedom: since our contracts live outside the target blockchain environment, they cannot natively create its transactions and therefore cannot easily interact with each other. For ease of presentation, we propose a more detailed comparison with Truebit [35]. To have a fair comparison, we limit ourselves to computable contracts whose output has a meaning for a target blockchain (not to be confused with the underlying blockchain), since Truebit is only used in these situations. Truebit system works through a *verification game*, that is an interactive challenge-based protocol, where computation tasks are performed off-chain and disputes are resolved on-chain (in the target blockchain) through a multi-round binary search over the computation steps. The framework assumes honest participation of all the actors and resorts to heavy on-chain resources only when a fraud test is suspected. Truebit employs an incentive layer to reward challengers for their auditing efforts. In contrast to Truebit, our computable contract extends computation delegation by including all the interactions between actors and allowing computations to be executed under arbitrary security requirements. Since

we do not have disputes, we never need to go on the target blockchain to verify correctness and thus we avoid any significant on-chain computation. To ensure correct execution, our contracts rely on verifiers, with a complex mechanism which is described later on in this paper. It may be noteworthy that the general setting of our computable contracts would make an interaction with Truebit fruitful, since it could serve as an external tool referenced by our contracts, providing verifiable execution results while our framework coordinates the broader interactions between all actors.

2 The Actors of the Contract

The contract's computation can be represented as an algorithm f. The contract does not contain f, rather it contains sufficient information to identify it (e.g., pointers and digests) for the actor that would compute it. The same holds for f's input data and f's output data, as well as other ancillary computations (e.g., verification algorithms) and contract conditions (except those which are necessarily public). Another important contract's component is $\mathbf{d}$, which encodes all additional data needed for the correct computation of f. The easiest scenario (which we keep in this paper as an example) is when f's input is not referenced in the contract and so $\mathbf{d}$ must contain f's input. In this simplified case, a user $\mathbb{U}$ requires the evaluation of f with input $\mathbf{d}$ and the verification of f's output via a verification algorithm g by an external verifier $\mathbb{V}$.

Our protocol considers several actors, of which only **users** are not involved in the blockchain's working, because they just send/receive transactions. Users are necessary because they provide the description (including $\mathbf{d}$) and funding of contracts. Our contracts need an underlying blockchain with *data availability* features (see [5]). More precisely, we request the following minimal properties from the underlying blockchain:

- the blockchain (as a network) has mechanisms to store and retrieve data when requested,
- the blockchain can provide a proof-of-availability for any stored data (whenever requested).

While we may call the nodes of this network generically "miners", we have specific names according to their actual role:

→ **standard miner**, denoted by $\mathbb{M}_i$, acts jointly with other miners to guarantee both the liveliness of the blockchain and the above-mentioned (minimal) data availability features;
→ **an I/O verifier**, denoted as $\mathbb{V}$, is a specialized miner that validates the correctness of data in input/output to f, using g;
→ **a calculator**, denoted as $\mathbb{C}$, which actually computes f.

There are two important aspects in every non-trivial contract:

- ▷ **data handling**, data access shall be restricted to only the authorized parties, unequivocally decided by the involved parties, while computational correctness and data integrity must be guaranteed at any step of data handling;
- ▷ **fair rewarding**, all parties deserve an economical reward for their participation according to their role. For example, an evaluator buying and reselling agglomerated data from different users should pay for the data and network fees. Another example concerns all miners, who require their computational/responsibility efforts to be awarded when their efforts are requested.

2.1 Assumptions and Threat Model

We assume the use of an ad hoc blockchain with data availability features. This allows for the management of contractual transactions and facilitates interaction with decentralized data. This infrastructure requires a hybrid threat model that combines cryptographic and economic security. For this reason, we suppose that all "miners" are untrusted in the *'blockchain sense'*, i.e. miners participate in the network's liveliness by following the protocol code, and any deviation from correct execution results in their work being ignored and punishment by the network, economically or in terms of loss of reputation (or both).

Our protocol aims to guarantee the following properties:

1. **correctness**: if all actors involved are honest, the output returned by the calculator $\mathbb{C}$ corresponds exactly to the result of the specified algorithm on the agreed-upon inputs;
2. **data confidentiality**: sensitive data must remain inaccessible to unauthorized parties. For example, during data exchange between two users, they must be the only ones able to access the data;
3. **fairness**: economic rewards and reputation scores are applied according to the protocol rules. In particular, actors are penalized if they misbehave.

Using cryptographic and economic security models, our approach balances robustness and practicality. In fact, cryptographic techniques are used to protect data confidentiality and privacy, while economic incentives ensure protocol compliance and discourage misconduct, as malicious actors can be detected and economically penalized. We also consider a *reputation system* (e.g. [16, 28, 15]) in which the trustworthiness of actors can be measured on the basis of their actions. This is possible because all contract executions are publicly recorded on the blockchain, allowing anyone to extract actors' participation in the protocol, historical participation, and reliability.

Finally, we wish to emphasize that we do not consider attacks that require the violation of cryptographic assumptions, the exploitation of implementation bugs, or the compromise of blockchain consensus beyond its expected fault tolerance.

3 Computable Contracts over Blockchain Transactions

Computable contracts (contracts from now on) have a precisely defined structure that allows the correct and verifiable outsourcing of computation. Any con-

tract describes: *(i)* the actors and their role in detail, *(ii)* the input data, or a pointer/reference to it, *(iii)* which actor(s) evaluates the correctness of the input data and how, *(iv)* which computation is expected on the input, including the confidentiality level of the computation itself, *(v)* how the computation is verified, *(vi)* how rewards are computed for all involved actors.

The contract is fully encoded as a transaction (for the underlying blockchain) allowing miners $\mathbb{M}_i$ to verify the contained fields and store them into blocks. The contract's execution is achieved by publishing authenticated transactions following the contained instructions, i.e. actors provide correctly signed transaction containing relevant evaluation input/output and/or auxiliary information (as per the contract itself) required to obtain access to the input/output. Such a choice enables public traceability of the contract's execution over the blockchain. The rewards promised in the contract are effectively released during execution.

More in detail, a contract is divided into four sections:

- ▷ Cont_1 contains information on how to handle the contract, such as actors, cipher-suite, crypto-parameters, etc.;
- ▷ Cont_2 identifies the data access rules, i.e. who can access the data and under which conditions (e.g. specific time, by providing auxiliary information);
- ▷ Cont_3 denotes information related to the outsourced computation and its verification, e.g. where to find the algorithms f, g, data m and the confidentiality level of the computation;
- ▷ Cont_4 describes the contract's execution in terms of transactions.

3.1 Communication Protocol

Notation. We denote by $(\mathsf{Enc}, \mathsf{Dec})$ a public-key encryption scheme (or similarly a hybrid encryption scheme [10] via $\mathsf{KEM}/\mathsf{DEM}$ paradigm), $(\mathsf{Share}, \mathsf{Recon})$ is a secret sharing scheme [2, 30] where (t, n) denotes the threshold t out of n parties, and finally $(\mathsf{Sign}, \mathsf{Ver})$ is a signature scheme algorithm [17], for example the one used by the underlying blockchain network. We define the key-pair for the entity $\mathbb{X}$ as $(\mathsf{sk}_{\mathbb{X}}, \mathsf{pk}_{\mathbb{X}})$. All the cryptographic primitives used must be secure according to the applications requirements, more information can be found in Section 4.

To allow the verifiable outsource of computation, we define a protocol between the previously-described parties. Such a protocol is divided into three phases: an *initial setup* that creates the contract and identifies all the required actors, a *sending phase* where data is effectively provided and a *retrieving phase* where data is retrieved and economic agreements are fulfilled. These phases define a single data-transaction and, by combining multiple data-transactions which are clearly identified in the initial setup, the protocol can handle more complex contracts where multiple data exchanges/evaluations are required.

We describe our communication protocol following the exchange of data from user $\mathbb{U}_1$ to user $\mathbb{U}_2$. The initial (off-chain) setup procedure is executed as follows:

(1) $\mathbb{U}_1$ prepares the preliminary transaction tx_0 with all the relevant encoded fields $(\mathsf{Cont}_1, \mathsf{Cont}_2, \mathsf{Cont}_3, \mathsf{Cont}_4)$, especially those necessary for the miners to evaluate the economical reward in providing their service.
(2) $\mathbb{U}_1$ shares the transaction tx_0 to the miners network. A set of miners $\{\mathbb{M}_i\}_i$ matching the conditions provided in the contract, checks tx_0, accepts to handle the contract and signals their willingness by individually signing tx_0, indicatively $\sigma_i \leftarrow \mathsf{Sign}\left(\mathsf{sk}_{\mathbb{M}_i}, \mathsf{tx}_0\right)$;
(3) $\mathbb{U}_1$ collects and verifies the signatures $\{\sigma_i\}_i$ and, after obtaining enough participants, moves to the following phase. If not enough miners are providing their signatures, tx_0 remains idle in the mempool until its lifetime expires.

At the setup procedure's end, the users and miners have agreed on a contract.

After finalizing the contract, the protocol executes a sequence of transactions that culminate in some derived data which have to be stored for later retrieval. As mentioned, the miners use a data availability (DA) layer as a distributed storage solution which guarantees provable availability of the uploaded data[7].

The sending phase is executed as follows:

(1) `(off-chain)` $\mathbb{U}_1$ prepares the original data m to be provided in the transaction according to the contract's specification. We denote as *datum* $\mathbf{d}$ the pre-processed data m: to maintain confidentiality, m is effectively encrypted with the appropriate scheme and according to the required application, e.g. $\mathbf{d}$ can be computed as the encryption $\mathbf{d} \leftarrow \mathsf{Enc}\left(\mathsf{pk}_{\mathbb{U}_2}, \mathsf{m}\right)$ of a public key asymmetric scheme using $\mathbb{U}_2$'s public key $\mathsf{pk}_{\mathbb{U}_2}$;
(2) `(off-chain)` $\mathbb{U}_1$ computes the secret sharing of the datum $\mathbf{d}$, obtaining shares $(\mathbf{d}_1, \ldots, \mathbf{d}_n)$ for the miners $\{\mathbb{M}_i\}$ that agreed to participate in the first phase, encrypting each share for the specific miner $\mathbb{M}_i$ as $\mathsf{c}_i \leftarrow \mathsf{Enc}\left(\mathsf{pk}_{\mathbb{M}_i}, \mathbf{d}_i\right)$, according to the contract's specifications;
(3) `(off-chain)` $\mathbb{U}_1$ creates a new transaction tx_1 containing the previous transaction tx_0, some additional relevant modifications (e.g. additional metadata not provided during the initial setup), the signatures of all participating miners $\{\sigma_i\}_i$, and their encrypted shares $\{\mathsf{c}_i\}_i$.
(4) `(on-chain)` $\mathbb{U}_1$ finalizes tx_1 into a complete transaction tx_2, while also adding the handling fee which is willing to spend, by signing it and sending it to the public mempool;
(5) `(on-chain)` after the appropriate verification of the transaction's validity (together with possible additional verification on the datum specified by the contract), all participants miners sign[8] tx_2 obtaining the final transaction tx_3 which is sent to the DA's mempool for storing on the DA layer. The miners store their shares and any relevant transaction information for the consequent retrieval phase;

[7] We omit details on the DA layer and point readers to recent surveys [5, 18].
[8] Each miner signs independently of the others; if one signature is missing, the procedure is aborted.

(6) as soon as the transaction is contained in a block, $\mathbb{U}_1$ obtains an identifier for the mined transaction that can be provided to $\mathbb{U}_2$ to follow-up on the next contract's step.

After tx_3 is stored on the DA layer, the contract can follow up by releasing the reconstructed datum $\mathbf{d}$ to the appropriate entity $\mathbb{U}_2$, being it a retriever or a calculator. More formally, the retrieval phase is defined as:

(1) $\mathbb{U}_2$ creates a request transaction rx_0 indicating, e.g. *(a)* $\mathbb{U}_2$'s identifier; *(b)* the datum it wants to retrieve, e.g. by indicating the identifier of the transaction tx_3; *(c)* additional information as described in the contract; *(d)* the lifetime of the request. In rx_0, $\mathbb{U}_2$ adds also the participant miners, the data retrieval fees and the required costs, as specified in the contract;
(2) $\mathbb{U}_2$ signs rx_0 and sends it to the mempool;
(3) the miners which are alive check in their internal storage/system if the transaction identifier specifies their participation in the transaction's contract. If so, they verify that all the contract's logics are fulfilled for the request rx_0;
(4) if the checks are satisfied[9], the miners retrieve the encrypted shares $\{\mathsf{c}_i\}_i$, execute the share reconstruction of the decrypted share $\mathbf{d}_i = \mathsf{Dec}(\mathsf{sk}_{\mathbb{M}_i}, \mathsf{c}_i)$, and send the reconstructed datum $\mathbf{d}$ to the mempool;
(5) the datum in the mempool is verified by the miners (e.g., if the receiving miner can prove to be $\mathbb{U}_2$). The correct verification of $\mathbf{d}$ (according to the information provided in tx_3 and rx_0) triggers the execution of the contract's payment, which is recorded in the blockchain via further transactions.

The above-mentioned protocol permits only the exchange of data from one party to another. To perform computations, our protocol requires multiple data-transactions, each indicating one data exchange, which are coordinated by the specific contract-execution section Cont_4 which was agreed upon in the initial setup phase. More precisely, the execution of a verifiable outsourced computation (Figure 1) would be described as:

(1) a transaction tx_a where the data m is pre-processed as datum $\mathbf{d}$ and provided to a calculator that is the receiver for this transaction. Observe that the eventual receiver $\mathbb{U}_2$ of the computation's output is listed in the contract;
(2) a request transaction rx_a, made by the calculator, to retrieve $\mathbf{d}$;
(3) after the calculations are executed, the calculator inserts the output $\mathsf{f}(\mathbf{d})$ into the transaction tx_b for the receiver $\mathbb{U}_2$ and the verifier $\mathbb{V}$;
(4) $\mathbb{V}$ obtains $\mathsf{f}(\mathbf{d})$ (directly or via specific transaction) and verifies the correctness of the computation according to the contract's requirements and provides its output as a transaction tx_c;
(5) the retriever $\mathbb{U}_2$ requests the output $\mathsf{f}(\mathbf{d})$ (via the transaction rx_b), which is received if $\mathbb{V}$ guarantees for the correct computation.

[9] The contract will specify whether all original miners are actually needed to follow up or if a subset is enough; this specification will be linked to the way (threshold?) d can be reconstructed; to simplify the presentation, we will assume from now on that all original miners are present and willing to execute fully the contract.

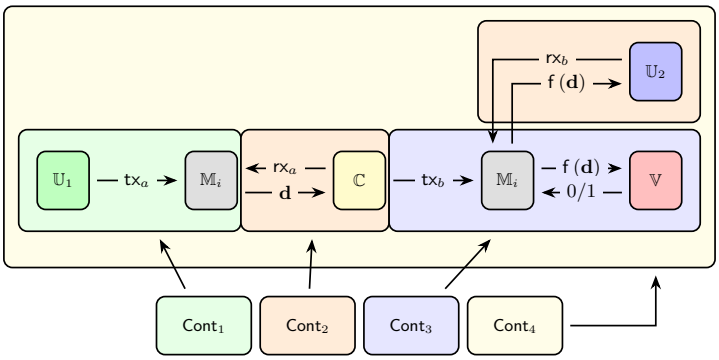

Fig. 1. High-level overview of a contract's transaction where a user $\mathbb{U}_1$ provides $\mathbf{d}$ to a calculator $\mathbb{C}$ which computes f. The verifier $\mathbb{V}$ checks the correctness of $\mathsf{f}(\mathbf{d})$ and, if correct, $\mathbb{U}_2$ is allowed to retrieve $\mathsf{f}(\mathbf{d})$ enabling the rewarding of $\mathbb{C}$, $\mathbb{V}$ and $\mathbb{M}_i$. Transactions are denoted as $(\mathsf{tx}, \mathsf{rx})$ or as (encrypted) content $(\mathbf{d}, \mathsf{f}(\mathbf{d}))$.

3.2 Special Case: Unknown Actors

An interesting situation arises when the identity of some actors is unknown at the time of contract's creation, being revealed only at a later time. A naive solution would require a proxy re-encryption scheme [36, 7], i.e. an encryption scheme that let miners transform a ciphertext c encrypted by $\mathbb{U}_1$ for a public key pk into a different ciphertext $\widehat{\mathsf{c}}$ (with the same plaintext) for a *different* public key $\mathsf{pk}_{\mathbb{U}_2}$. This approach introduces unnecessary complexity.

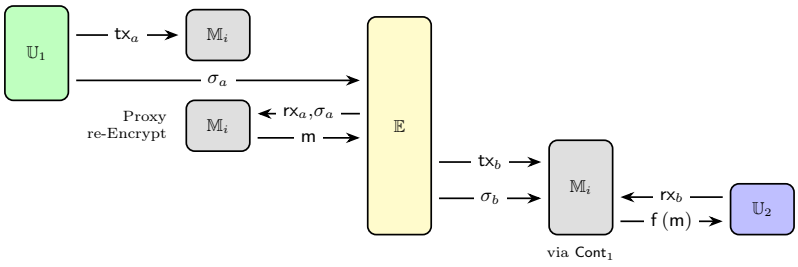

Fig. 2. Overview of proxy re-encryption and contract-based solutions for outsourced computations. $\mathbb{E}$ and $\mathbb{U}_2$ are unknown during contract's initialization.

On the other hand, our protocol admits a simpler approach, highlighted in Figure 2, where the contract contains a *challenge* that can only be solved if evaluator/retriever receives some sort of *confirmation* σ (to be provided while requesting to access the data). This is possible because the miners must follow the contract directions which, in this case, indicates that σ is required to access

the data. Again, this approach assumes that $\mathbb{U}_1$ honestly provides the correct encryption $\mathsf{Enc}\left(\mathsf{pk}_{\mathbb{U}_2}, \mathsf{sk}\right)$. If fully untrusted computation must be guaranteed, additional proof of correct encryption must be provided (e.g. via SNARK/STARK).

3.3 Protocol's Implementation

Palliora [24] is a framework focussed on decentralized AI applications. A key aspect is that Palliora enables fair compensation between actors involved into the AI execution-flow, i.e. data must be provided by data-provider to train AI-model, models must be trained by calculators that provide computational power and so on.To achieve fair compensation, Palliora builds a contract's instantiation which follows our approach with the appropriate ad-hoc modification to fit the specific decentralized AI necessities.

At the time of writing, all the DA-layer and underlying transactions between actors are consistently executed as described in our approach, thus allowing the contract's correct execution. The full computable contract potentiality denoting complicated execution-flows (e.g. usage of advanced cryptographic protocol such as FHE, conditional execution-flow) are still under development.

4 Discussion and Future Direction

We conclude our work-in-progress paper by providing some additional discussion points related to informal security arguments, general properties of contracts and future research directions.

4.1 Security Arguments

The contract's execution is designed to coordinate the communication exchange between the actors, as some sort of communication layer. Cryptographic primitives and protocols are applied to guarantee data confidentiality, computations verifiability and specific contract application's requirements. For example, to provide enhanced confidentiality for evaluation, the protocol can introduce cryptographic secure evaluation via, e.g, fully homomorphic encryption [21, 11, 3] or, if the requirement is mainly verifiability, surgical application of SNARK/STARK [12, 19] aimed at only f's correctness proof and not at the whole contract's execution.

We argue that our protocol provides an adaptable framework for verifiable outsourced computation, with different security requirements, from strict usage of fully homomorphic encryption (FHE) and SNARK/STARK to techniques that leverage the protocol structure and the penalties for misbehaving.

Data security is obtained via surgical usage of appropriate cryptographic tools. In particular, the security of datum $\mathbf{d}$ is always maintained since it is effectively a ciphertext of the original data m for some designated receiver. This is reconstructed only at the end of the protocol (when required) while, during

the sending phase, $\mathbf{d}$ is split into secret shares $(\mathbf{d}_1, \ldots, \mathbf{d}_n)$ which are encrypted for each miner. Thus, each miner obtains only its own share.

On the other hand, if a malicious $\mathbb{V}$ falsely claims some verification to be negative, then the relevant parties will be not rewarded. To prevent this, our contract's design requires the initial user to select $\mathbb{V}$ based on the perceived reputation. Moreover, the contract may require a formal proof for the verifiers' honest behaviour, e.g. by forcing the usage of secure cryptographic primitives, or the proof might be constructed by reaching a verifiers' majority (or even more complex quorum situations) where many verifiers are required. In any case, a correctly instantiated contract would provide a mechanism to identify such misbehaviour, thus forcing an economic penalty to the malicious $\mathbb{V}$.

4.2 Further Comments

Now that we have described in details our contracts, we can easily identify two important aspects where they differ significantly from smart-contracts:

- our contracts enable the execution of arbitrary code on arbitrary input, without any constraints on the programming language/environment. Differently from blockchain platforms using general-purpose execution environments[10], our contracts rely on human-readable descriptions, rather than any form of script: they must be proposed by humans and agreed upon by humans[11]. This freedom in the contract formulation allows for an extreme flexibility, which is of paramount importance for real applications;
- our contracts do not adopt a gas-based payment system (e.g. as done by Ethereum), rather they leave the determination of the contract's fees to the free negotiation among parties. This is especially useful for contracts aiming at training an AI model, which can be a hugely-expensive computational task. Thus, the fee for the computation cannot be determined with simple formulas, but depends on the evaluator capabilities. Similarly, the evaluation effort is expected in some cases to be much higher than that for the verifiers, and the relationship between the corresponding fees cannot be standardized in all situations.

Computable contracts are *an approach* to verifiable computation outsourcing, meaning that, intuitively and with the appropriate considerations, they can be implemented as some sort of data-struct over a layer-2 and be fully handled via smart-contracts. However:

- such choice would require a revision of our execution protocol into smart-contracts which might allow, e.g., some sort of roll-up technique to reduce the amount of contractual-transactions required but, at the same time,

[10] For example, blockchains using web assembly (WASM) like Polkadot [25].

[11] When we say "humans" we also include AI agents, or other programmes, that can read and process instructions in natural-language form structure. In other words, we only affirm that the contracts are easy to understand by humans. If humans trust a software to read contract and accept them, this is their decision.

might require the execution of a computationally intensive algorithm (proof-verification) on the smart contract;
- these different design choices might lead to more complex data-handling and less predictable costs, which are of perpendicular interest compared to the sole approach presented.

Therefore, we consider computable contracts as layer-1 citizens with their ad hoc blockchain, obtaining a simpler protocol with more predictable costs.

We further list several discussion points of general interest that requires clarification to avoid misunderstanding our computable contracts usage:

- A computable contract defines the functions to be evaluated in Cont_3 where a "function location" must be provided to allow the retrieval of the function/algorithm, similarly for the verification evaluation. Such location can be implemented with a simple web-link or an appropriate oracle call, clearly in relation to the contract's security requirements. Our approach allows for any scenario, leaving the choice to the implementation.
  At the same time, the provided algorithms might be deterministic or probabilistic in nature, since such a choice does not modify our protocol's execution in any way. We only require computational termination, which can be obtained with a bounded evaluation time. This highlights the importance of the contract's setup phase, where actors must decide 'what to compute' and highlights 'where' algorithms are effectively stored thus requiring additional attention on the functions agreed upon. Depending on the application, functions might be publicly version-traced or limited to a certified set, e.g. computable functions might be selected as training or evaluate inference from a pre-defined list of AI model.
- Apart from the requirements described in Section 2, we do not assume anything more on the underlying blockchain. Therefore, details such as its block structure, its consensus algorithm, its fees (for handling transactions and DA properties), are irrelevant to define our contracts. Obviously, these details are essential to obtain an instantiation of our contracts that satisfy the requirements of a the specific use case under consideration.
- It is the user funding the contracts that decides the verifiers, and the verdict of the verifiers is final, but two aspects need to put this draconian description into the right perspective. First, no calculator is forced to accept the user's contract (which crucially contains also the identity of the verifiers), so the user's apparently-absolute freedom must match the equally-absolute freedom of the calculator. They will find a middle point in their own off-chain discussion, or the contract will not be finalized. Second, the contract specifies also the method used by the verifiers to reach their verdict. It could be anything: one sole verifier trusted by all actors thanks to its reputation, a set of commonly-trusted verifiers taking a decision with a quorum system, an external protocol based on challenge-response, etc. Again, whatever the method, it is only *proposed* by the user, since it must be accepted by all actors or the contract will not execute.

- When we say that the user *funds* the contract, we mean that the user commits a quantity of the native cryptocurrency used by the underlying blockchain (the same coins used to pay for the DA fees). There is no automatic determination of the reward of the contract's actors. Indeed, in theory the user proposes their reward and how it is computed, while the actors accept or reject it. Clearly, in practice there will be off-chain negotiations to reach an agreement before even writing tx_0.

4.3 Future Direction

The next step in our work in progress is to write a formal security proof to fully guarantee that no party can maliciously misbehave without being identified and punished. The tricky part of the proof design is to model scenarios of importance for real application, e.g. how to properly initialize the contract when there are unknown actors.

Finally, we would like to discuss two upcoming global events in cryptography that will likely impact the instantiations of our contracts (but not their general structure). The first event is the current worldwide migration to post-quantum cryptography (PQC): we have to ensure that our instantiations allow for a significant degree of crypto-agility and proper integration with PQC solutions. The second event is the future creation of standards for secure multi-party computation (SMPC). Indeed, our protocol relies heavily on SMPC algorithms, used especially by miners, which need to handle secret shares. In both cases, we are following the discussion of current standardization efforts [22, 23] to properly adapt our protocol instantiations and, if needed, the protocol itself.

Acknowledgments. This paper has been written during the first research phase for the Palliora project (`https://www.palliora.org`), which incorporates our computable contracts and has been running extensive testing. Preliminary discussion and test reporting were presented at the conference CIFRIS24 in September 2024 *"HASHTA AI - Share and compute securely your data"* [9], a more recent draft was presented at a workshop in CIFRIS25 [4] and at a XRPL Workshop *"Cryptography at Work for Blockchain"* in Paris in October 2025. We like to thank Andrea Gangemi, Massimo Caccia, and the anonymous reviewers for the insightful comments and suggestions.

References

1. Arbitrum, Arbitrum: Welcome to the future of Ethereum, `https://arbitrum.foundation`.
2. Benaloh, J.C.: Secret sharing homomorphisms: Keeping shares of a secret secret. In: Conference on the Theory and Application of Cryptographic Techniques. Springer-Verlag, Berlin (1987)
3. Bos, J., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schanck, J.M., Schwabe, P., Seiler, G., Stehlé, D.: CRYSTALS-Kyber: a CCA-secure module-lattice-based KEM. In: 2018 IEEE European Symposium on Security and Privacy (EuroS&P), pp. 353–367 (2018)

4. Brunetta, C., Galatolo, S.: Verifiable Computation Outsourcing via Contracts over Blockchain Transactions. In: CIFRIS25 ACTA, pp. 84–85. De Cifris Press (2025). https://doi.org/10.69091/koine/vol-7-W04
5. Brunetta, C., Sala, M.: SoK: Modelling Data Storage and Availability. Financial Cryptography and Data Security. FC 2025 International Workshops (2025)
6. Buterin, V.: Ethereum: a next generation smart contract and decentralized application platform, https://github.com/XXX:Buterin13/wiki/wiki/White-Paper (2013).
7. Canetti, R., Hohenberger, S.: Chosen-ciphertext secure proxy re-encryption. In: Ning, P., De Capitani di Vimercati, S., Syverson, P.F. (eds.) ACM CCS 2007, pp. 185–194. ACM Press (2007). https://doi.org/10.1145/1315245.1315269
8. ChainLink, Unlock onchain finance at scale, https://chain.link.
9. Chaudhary, A.: Work in progress: HASHTA AI - Share and compute securely your data. In: CIFRIS24 ACTA, pp. 105–108. De Cifris Press (2025). https://doi.org/10.69091/koine/vol-5-W16
10. Dixit, P., Gupta, A.K., Trivedi, M.C., Yadav, V.K.: Traditional and hybrid encryption techniques: a survey. In: Networking Communication and Data Knowledge Engineering: Volume 2, pp. 239–248 (2018)
11. Doan, T.V.T., Messai, M.-L., Gavin, G., Darmont, J.: A survey on implementations of homomorphic encryption schemes. The Journal of Supercomputing **79**(13), 15098–15139 (2023)
12. Gabizon, A., Williamson, Z.J., Ciobotaru, O.: PLONK: Permutations over Lagrange-bases for Oecumenical Noninteractive arguments of Knowledge, Cryptology ePrint Archive, Paper 2019/953 (2019). https://eprint.iacr.org/2019/953.
13. Goodenough, O., Salkind, S.: Computable Contracts and Insurance: An Introduction. MIT Computational Law Report (2022)
14. Gravity Bridge, Blockchain Unlocked, https://www.gravitybridge.net.
15. Gurtler, S., Goldberg, I.: SoK: Privacy-Preserving Reputation Systems. Proceedings on Privacy Enhancing Technologies **2021**(1), 107–127 (2021). https://doi.org/10.2478/popets-2021-0007
16. Hendrikx, F., Bubendorfer, K., Chard, R.: Reputation systems: A survey and taxonomy. Journal of Parallel and Distributed Computing **75**, 184–197 (2015). https://doi.org/10.1016/j.jpdc.2014.08.004
17. Johnson, D., Menezes, A., Vanstone, S.: The elliptic curve digital signature algorithm (ECDSA). Int. J. Inf. Sec. **1**, 36–63 (2001). https://doi.org/10.1007/s102070100002
18. Li, C., Xu, M., Zhang, J., Guo, H., Cheng, X.: SoK: Decentralized storage network. High-Confidence Computing **4**(3), 100239 (2024). https://doi.org/10.1016/j.hcc.2024.100239
19. Liang, J., Hu, D., Wu, P., Yang, Y., Shen, Q., Wu, Z.: SoK: Understanding zk-SNARKs: The Gap Between Research and Practice, Cryptology ePrint Archive, Paper 2025/172 (2025). https://eprint.iacr.org/2025/172.
20. Linea, The L2 Where Ethereum Wins, https://linea.build.
21. Marcolla, C., Sucasas, V., Manzano, M., Bassoli, R., Fitzek, F.H., Aaraj, N.: Survey on fully homomorphic encryption, theory, and applications. Proceedings of the IEEE **110**(10), 1572–1609 (2022)
22. NIST, Multi-Party Threshold Cryptography (MPTC), https://csrc.nist.gov/projects/threshold-cryptography (visited on 08/19/2025).
23. NIST, Post-Quantum Cryptography (PQC), https://csrc.nist.gov/projects/post-quantum-cryptography (visited on 08/19/2025).

24. Palliora, Superagency for Human-AI Coordination, https://www.palliora.org.
25. Polkadot, Polkadot: Instruments for Individuality. https://polkadot.com.
26. Polygon Portal, Your Polygon Journey Starts Here, https://portal.polygon.technology.
27. Provable, Provable: Truly Private, https://provable.com.
28. Quaglia, E.A.: DeCifrisTrends Lecture 9 "Protocols for Peer Rating Systems", (2023). https://www.youtube.com/watch?v=HWp-KfHRvKQ.
29. Scroll, Network for the Open Economy, https://scroll.io.
30. Shamir, A.: How to Share a Secret. Communications of the Association for Computing Machinery **22**(11), 612–613 (1979). https://doi.org/10.1145/359168.359176
31. Solana, Solana: The capital market for every asset on earth. https://solana.com.
32. Starknet, The Bitcoin DeFi Layer, https://www.starknet.io.
33. Synapse, Secure cross-chain communication, https://www.synapseprotocol.com.
34. Tezos, Tezos: Blockchain designed to evolve. https://tezos.com.
35. TrueBit, TrueBit: Don't just trust, verify, https://truebit.io.
36. Zhou, Y., Liu, S., Han, S., Zhang, H.: Fine-Grained Proxy Re-encryption: Definitions and Constructions from LWE. In: Guo, J., Steinfeld, R. (eds.) ASIACRYPT 2023, Part VI. LNCS, pp. 199–231. Springer, Singapore (2023). https://doi.org/10.1007/978-981-99-8736-8_7