An Approach to Computable Contracts with Verifiable Computation Outsourcing and Blockchain Transactions

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

[1] Independent Researcher, France, brunocarletta@gmail.com
[2] University of Warwick, UK, amit.chaudhary.3@warwick.ac.uk
[3] University of Pisa, Italy stefano.galatolo@unipi.it
[4] University of Trento, Italy massimiliano.sala@unitn.it

Abstract. In this short paper we present an 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, 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, where they are encoded as transactions, to achieve their decentralized handling and thus enforcing their correct execution via a combination of cryptographic techniques and economic security. Our delegation architecture allows for the execution of very complex collaborative tasks, such as the deployment of an AI marketplace.

Keywords: Verifiable Computation · AI Marketplace · 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), mostly (decentralized) solutions use smart-contracts (e.g., in Ethereum [3]) as main bricks and then adds other components to achieve different goals, such as verifiability with ad hoc systems (e.g., TrueBit [11]), or scalability with oracles and/or bridges and/or second-layers 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, where they are encoded as transactions, to achieve their decentralized handling and thus enforcing 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 blockchain not only to store the agreed final version of the contract, but alto to keep track of any subsequent contract's execution.

Crucially, 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 between 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.

In this paper we briefly introduce the main actors considered in our solution, describe their role and their goal. We sketch how the contract is defined and how the contract is executed. We close our short paper by providing some considerations on security versus applicability, with an eye to possible extensions.

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 server $\mathbb{C}$ 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 (expect those which are necessarily public). Another necessary contract's component is $\mathbf{d}$, some 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 working because they just send/receive transactions. Users are necessary, since they provide the description and funding of contracts (including $\mathbf{d}$). Our contracts need an underlying blockchain with *Data Availability* features; 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 minimal Data Availability features, such as data storage retrieval;

→ **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.

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 contract 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 an ad-hoc 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 with $(\mathsf{Enc}, \mathsf{Dec})$ a public-key encryption scheme (or similarly a hybrid encryption scheme via $\mathsf{KEM}/\mathsf{DEM}$ paradigm [5]), $(\mathsf{Share}, \mathsf{Recon})$ is a secret sharing scheme [1, 10] where (t, n) are the required t threshold amount out of n parties, $(\mathsf{Sign}, \mathsf{Ver})$ is the signature scheme algorithms [6], e.g. 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 Sec. 4.

To allow the verifiable outsource of computation, we define a protocol between the previously described parties. Such protocol is described into 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 where economic agreements are fulfilled. These phases define a single transaction and, by combining multiple transactions which are clearly identified in the initial setup, the protocol can handle more complex contracts where multiple data exchanges are required.

We describe our communication protocol following the exchange of data from $\mathbb{U}_1$ to $\mathbb{U}_2$. 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 $\sigma_i \leftarrow \mathsf{Sign}(\mathsf{sk}_{\mathbb{M}_i}, \mathsf{tx}_0)$;
(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 end of the setup procedure, these parties have agreed on a contract.

After finalizing the contract, the protocol executes a sequence of transactions that culminate in some derived data which to be stored for later retrieval. To guarantee full decentralization, the miners use a Data Availability (DA) layer as a distributed storage solution which guarantees provable availability of the uploaded data. We omit details on the DA layer and point the interested reader to recent surveys [2, 7].

The second 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 some shares $(\mathbf{d}_1, \ldots, \mathbf{d}_n)$, each encrypted for the specific miner $\mathbb{M}_i$ precisely as $\mathsf{c}_i \leftarrow \mathsf{Enc}\left(\mathsf{pk}_{\mathbb{M}_i}, \mathbf{d}_i\right)$, according to 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, the signatures of all participating miners $\{\sigma_i\}_i$, and their encrypted shares $\{\mathsf{c}_i\}_i$.
(4) `(on-chain)` $\mathbb{U}_1$ finalizing 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 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), the volunteering miners sign 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 a consequent retrieval phase;
(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 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, the miners retrieve the encrypted shares $\{\mathsf{c}_i\}_i$, execute the share reconstruction of the decrypted share $\mathbf{d}_i = \mathsf{Dec}\left(\mathsf{sk}_{\mathbb{M}_i}, \mathsf{c}_i\right)$, and send the reconstructed datum $\mathbf{d}$ to the mempool;
(5) the datum in the mempool is verified by the miners. 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 transactions, each indicating one data exchange, which are coordinated by the specific contract execution agreed upon in the initialization phase. More precisely, the execution of a verifiable outsourced computation (Fig. 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 $\mathsf{f}(\mathbf{d})$ and verifies the correctness of the computation according to the contract's requirements;
(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.

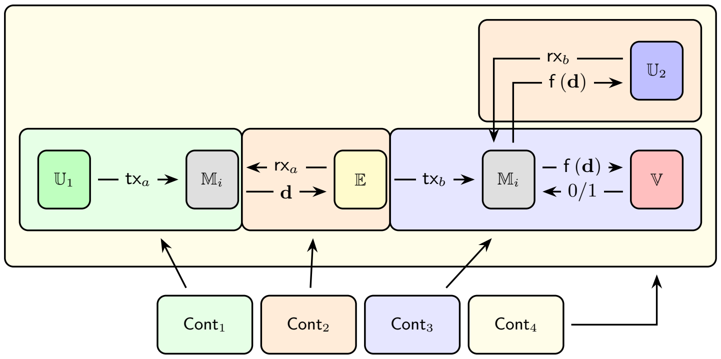

Fig. 1: High-level overview of a contract's transaction where a user $\mathbb{U}_1$ provides $\mathbf{d}$ to an evaluator $\mathbb{E}$ which computes f. The verifier $\mathbb{V}$ checks the correctness of $\mathsf{f}(\mathbf{d})$ and, if correct, $\mathbb{U}_2$ is allowed to retrieve $\mathsf{f}(\mathbf{d})$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

4 Discussion and Future Direction

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 or, if the requirement is mainly verifiability, surgical application of SNARK/STARK proofs aimed at only f's correctness proof and not at the whole contract's execution.

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, 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.

On the other hand, our protocol admits a simpler approach, highlighted in Fig. 2, where the contract contains a *challenge* that can only be solved if evaluator/retriever receives some sort of token 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).

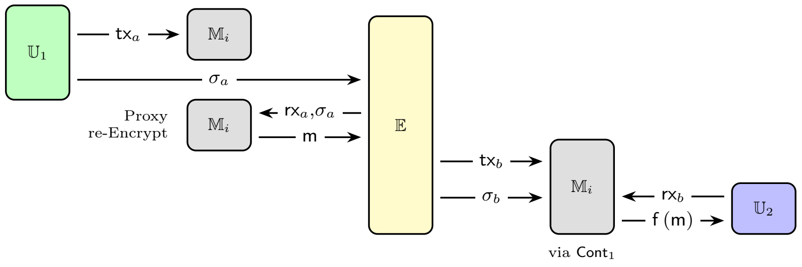

Fig. 2: Overview of proxy re-encryption and contract-based solutions for outsourced computations. $\mathbb{E}$ and $\mathbb{U}_2$ are unknown during contract's initialization.

Our contracts differ significantly from usual contracts that can be found in public blockchains, such as the smart-contracts in Ethereum [3], in at least two significant aspects:

- Our contracts enable the execution of arbitrary code on arbitrary input, without any constraints on the programming language/environment. Our contracts rely on human-readable descriptions, rather than any form of script: they must be proposed by humans and agreed upon by humans. This freedom in the contract formulation allows for an extreme flexibility, which is of paramount importance for real application.
- 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.

4.1 Security Arguments

We argue that our protocol provides an adaptable framework for verifiable outsourced computation, with different security requirements, from strict usage of

FHE and SNARK/STARK to more computing intensive by leveraging 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 Future Direction

Despite the security arguments provided above, our protocol would require a formal security proof to fully guarantee that no party can maliciously misbehave without being identified and punished. On this note, a thorough attention should be invested in providing security proofs for general technique of importance for real application, e.g. how to properly initialize the contract when there are unknown actors.

With the current migration to post-quantum cryptography (PQC), we are required to adapt our solution to allow crypto-agility and proper integration with PQC solutions. Similarly, our protocol would further benefit if instantiated using standardized protocol for secure multi-party computation, e.g. the miners are provided secret shares which related to threshold cryptography. In both cases, we are following the discussion of current standardization efforts [8, 9] to properly adapt our protocol.

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" [4].

References

1. 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)

2. Brunetta, C., Sala, M.: SoK: Modelling Data Storage and Availability. Financial Cryptography and Data Security. FC 2025 International Workshops (2025)
3. Buterin, V.: Ethereum: a next generation smart contract and decentralized application platform, `https://github.com/XXX:Buterin13/wiki/wiki/White-Paper` (2013).
4. 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`
5. 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)
6. 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`
7. Li, C., Xu, M., Zhang, J., Guo, H., Cheng, X.: SoK: Decentralized storage network. High-Confidence Computing **4**(3), 100239 (2024). `https://doi.org/https://doi.org/10.1016/j.hcc.2024.100239`
8. NIST, Multi-Party Threshold Cryptography (MPTC), `https://csrc.nist.gov/projects/threshold-cryptography` (visited on 08/19/2025).
9. NIST, Post-Quantum Cryptography (PQC), `https://csrc.nist.gov/projects/post-quantum-cryptography` (visited on 08/19/2025).
10. 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`
11. TrueBit, TrueBit: Don't just trust, verify, `https://truebit.io`.