SoK: Outsourced Computation Using Smart Contracts

1st Given Name Surname
dept. name of organization (of Aff.)
name of organization (of Aff.)
City, Country
email address or ORCID

2nd Given Name Surname
dept. name of organization (of Aff.)
name of organization (of Aff.)
City, Country
email address or ORCID

***Abstract*—Blockchain technology faces computational constraints related to programming languages and fees, especially when it comes to the implementation of smart contracts. Several solutions have been proposed to address these limitations, most of which involve Outsourced Computation (OC) to external servers, which sometimes are referred as *oracles*. This paper presents a comprehensive analysis of the existing literature on blockchain-based OC, with the aim of answering pivotal research questions. These questions delve into the similarities among various OC proposals, the security assurances provided by external servers, the nature of computations that can be outsourced, and the mechanisms ensuring payment in secure OC environments.**

Our study makes several contributions to the field. We introduce classification framework of blockchain-based OC, establish a connection between OC and the Fair Exchange problem, and highlight the common challenges faced by current OC protocols. Furthermore, we propose potential improvements to these challenges and introduce the concept of the OC-trilemma.

***Index Terms*—blockchain, smart contract, oracle, verifiable computation, ZKCP, replication**

I. Introduction

The computational capabilities of blockchain technology are currently constrained by several factors. One such limitation is the programming language used for the implementation of smart contracts on the blockchain, which is less versatile compared to languages used in general-purpose computing. Additionally, the high fees associated with complex smart contracts not only make them prohibitive for general computation, but also prevent the full development of the expressive capacity of smart contracts. This expressive capacity could be achieved if the fees were lower, despite the limitations of the language [1]. These constraints hinder the potential for widespread adoption and the development of practical use cases. Specifically, they isolate blockchains, disconnecting them from real-world applications. Consequently, numerous proposals have been put forth to bridge this gap.

Most of these solutions involve outsourcing computation to an external entity, such as a single server (e.g. [2]) or a cluster of IoT devices (e.g. [3], [4]). Sometimes these solutions are called *oracles*. For example, the so called 'decentralized oracle network' in Chainlink [5] together with the users creates an Outsourced Computation (OC) protocol using replication (see Section IV-B1c).

Despite the apparent similarities among these proposals, a structured or systematic approach to the topic has been lacking. This lack of a structural model complicates the understanding and contextualization of the subject, especially when trying to relate it to the body of literature on Fair Exchange [6] and Verifiable Computation [7] that was established before the emergence of blockchain technology.

To address this research gap, we have reviewed significant digital libraries and analyzed the literature on outsourced computation via smart contracts. Our analysis was guided by the following research questions:

RQ1: What are the commonalities (if any) between the various OC proposals?
RQ2: What security guarantees can an external server provide regarding the computation performed?
RQ3: What types of computation can be outsourced and what are the limitations?
RQ4: Assuming a secure OC exists in a certain attacker model, how can parties be certain of the payment?

In this paper, we aim to answer the aforementioned research questions and provide new insights into the topic of outsourced computation. Our goal is to offer a structured document that serves as a comprehensive reference for understanding this complex subject.

Contribution: The contributions of this paper can be summarized as follows:

- We provide an evaluation framework for OC, which answers RQ1;
- We link OC to the Fair Exchange problem, and prove that OC is impossible without a third party, which directly answers RQ4 and RQ2
- We highlight the common challenges for existing OC protocols, which directly answers RQ1
- We suggest improvements to some of these challenges and propose new research questions
- We present the OC-trilemma which directly answers RQ2 and RQ3

Organization: The remainder of this paper is organized as follows: in Section II we present the technical background needed to understand the different solutions, the theoretical framework and theorem we propose. In Section III we formally

define the concept of blockchain-based OC and present the impossibility theorem. We use these concepts in Section IV where we present an evaluation framework for the possible designs of blockchain-based OC proposals. After that, we analyze current OC Challenges in Section V and then we present the OC trilemma together with new research questions in Section VI. Then we present relevant Related Works in Section VII and we conclude in Section VIII.

II. Preliminaries

A. Ledgers

We use the terms *blockchain*, *distributed ledger* and *ledger* as synonyms. We consider a ledger $\mathbf{L}$ to be an append-only sequence of set of information and instructions, commonly called *blocks*. Each block $\mathbf{B}$ is committed to the following one, generally via the output of a hash function[1]: this information is stored in the *header* of the block among other pieces of information. Changes in the state of $\mathbf{L}$ follows a *consensus algorithm*, which specifies the validity of instructions (generally called *transactions*, denoted here as TX) and their rules for inclusion in the block.

So defined, blocks also define *time* in the ledger $\mathbf{L}$ as discrete rounds[2] of (valid) information and instruction appending.

B. Channels

In the context of information systems, the term *channel* is broadly used to denote any medium or mechanism that facilitates the flow of information. A channel $\mathbf{C}$ can be perceived as a conduit through which data travels from one point to another. Channels are intrinsic to the functionality of distributed systems, including blockchains, where they serve as the pipelines for data transmission and communication. If a ledger $\mathbf{L}$ is an append-only sequence of blocks, then a channel $\mathbf{C}$ is an open pathway, enabling the ongoing exchange of information.

Theoretically, a channel may also be a ledger, i.e. ledgers are a subset of channels. In this regard, we do not put any constrain on time on channels, although (for simplicity reasons) we assume a channel $\mathbf{C}$ has a continuous flow and exchange of data within and across systems when not specified.

C. Smart Contracts

In the context of blockchain systems, a *smart contract* is a set of deterministic instructions encoded as a script or program. These instructions may use either a Turing-complete language (e.g. in Ethereum and EVM compatible blockchains) or be expressed though more limited, stack-based, non-Turing-complete scripting language (e.g. in Bitcoin). The state and outputs of these smart contracts become a part of the ledger $\mathbf{L}$.

[1]In this sense, block $\mathbf{B}$ is *back-linked* to block $\mathbf{B}-1$. This is a *reveal-then-commit* scheme, different from normal commit-and-reveal schemes since in blockchains we want to preserve integrity of the past.

[2]Some blockchain proposals may be characterized differently, as for example the work by Eyal *et al.* [8], where the authors present a ledger with blocks and *sub-blocks*. For easiness of explanation and since we will deal mostly with Bitcoin-like and Ethereum-like systems we omit this differentiation.

A smart contract, in both systems, can be seen as a specialized type of transaction with the ability to execute logic beyond simple asset transfer.

Formally, a smart contract $\mathbf{Sc}$ on both platforms can be represented as a tuple:

$$\mathbf{Sc}^i := < code, state^i, trigger^i >$$

where:

- $code$ represents the immutable set of instructions or program code of the smart contract. In EVM-compatible systems, this can be any computation expressible in its scripting language, whereas in Bitcoin, it is limited to simpler, predetermined operations, commonly knows as *locking scripts*.
- $state^i$ denotes the state of the contract at i-th block $\mathbf{B}^i$. This state may evolve over time based on the execution of code.
- $trigger^i$ refers to the set of inputs that initiate the execution of code at block $\mathbf{B}^i$, stemming from incoming transactions $(\mathrm{TX}_1^i, \mathrm{TX}_2^i, \ldots, \mathrm{TX}_N^i)$. Note that $trigger^i \neq \emptyset, \forall i$ since every smart contract needs to be triggered to change its state and cannot do that automatically. In Bitcoin, these triggers are usually conditionals based on transaction inputs and outputs commonly knows as *unlocking scripts*, while in Ethereum, they can be more complex function calls and state changes.

In Bitcoin-like systems, smart contracts are more restricted due to the simplicity and limited scripting language of the Bitcoin blockchain, primarily focusing on conditions for spending outputs and requiring that the evaluation of the combination (stacking) of the unlocking an locking script results to `true`. On the other hand, in EVM-compatible systems, smart contracts are more complex and can embody a wide range of logic, from simple transactions to decentralized applications, thanks to the Turing-complete Ethereum Virtual Machine (EVM).

Each execution of a smart contract's code, resulting from a triggering transaction or event, leads to a state transition in the ledger $\mathbf{L}$, recorded as part of a block $\mathbf{B}$. The deterministic nature of these contracts ensures that all nodes on the network, following the consensus algorithm, will independently arrive at the same resulting state when executing the contract's code under the same conditions.

D. Systems

A *system* is defined as an entity that interacts with its environment through an interface, as described by Lamport [9]. We adopt the state-based approach to specify a system, as proposed by Pagnia [10]. In this approach, the state of the system fully describes the state of the interface at any given point in time. The system's state includes the input variables (written by the environment) and output variables (written by the system).

Systems can be categorized as either *open* or *closed*. A system is considered closed if it does not interact with the environment, while it's open if it does interact and modifies

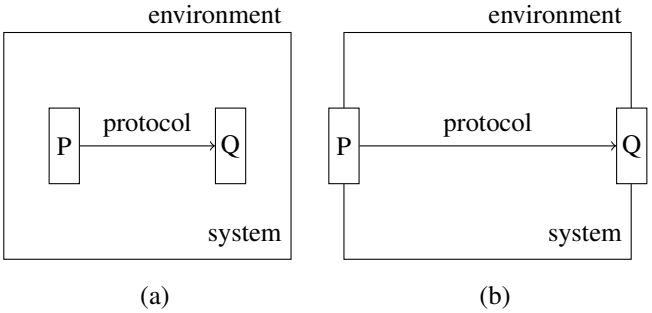


Fig. 1. System for fair exchange scenario: closed (a) and open (b), as explained in Pagnia [10].

the environment. Specifically, in a closed system, the parties interacting with each other through a protocol are considered to be within the system. Conversely, in an open system, the parties are outside the system and are therefore part of the environment. Figure 1 provides a visual representation of these concepts.

The key difference between open and closed systems in our context is that in an open system, parties can request other parties to perform actions for them, while in a closed system, the parties interact only with each other, the blockchain, and the channel.

Systems can also be classified as either *asynchronous* or *synchronous*, based on their handling of message transmission times. In asynchronous systems, the delay δ between the sending and receiving of a message is considered to be unbounded, potentially extending to infinity. This implies that there is no guaranteed timeframe within which a message must be delivered. In contrast, synchronous systems operate under the assumption that there is a defined upper limit to δ, ensuring that message transmission occurs within a predictable and constrained time window.

The distinction between open and closed systems and asynchronous and synchronous ones will be further explored in Theorem 1.

E. Third Parties

In the context of OC protocols, an increasing body of literature explores the use of distributed computing platforms, similar to those in blockchain technologies like Bitcoin and Ethereum, for facilitating computation-intensive tasks. These platforms employ distributed consensus mechanisms to validate the integrity and correctness of computations outsourced by clients. In OC, the integrity of the computation process is paramount, ensuring that the returned results are both accurate and trustworthy. This approach effectively transforms the distributed consensus mechanism into a proxy for a trusted third party (TTP).

In this context, if a majority of the nodes in the distributed network are operating honestly, the correctness and integrity of the outsourced computations are ensured. This setup mirrors the enforcement mechanism seen in fair exchange protocols, where the release of a resource (e.g., data, computation results) is contingent upon the fulfillment of certain conditions verified by the consensus network (we formally prove the connection between OC protocols and Fair exchange in Lemma 1). The role of the TTP, therefore, evolves from simply enforcing correct behavior to also ensuring the confidentiality and privacy of the involved parties.

Whether the TTP is a singular entity or a decentralized collective operating under a consensus algorithm (i.e. the set of validators or miners), the primary goal remains the same: to guarantee the correctness and integrity of outsourced computations. Granted, the resilience of such systems when a TTP is implemented as a distributed collective is inherently higher against individual node failures or malicious activities, but only if it is assumed that there is a *honest majority* [11] in the set of write-enabled nodes (blockchain miners or validators). Throughout the paper and especially in Section IV-B we use this assumption.

F. Fair Exchange

Loosely speaking, an exchange is *fair* if no participant can cheat. More formally, assume two parties P and Q have items i_P and i_Q respectively and they want to exchange them. Assume also that for each item $i_j, j = P, Q$ there is a description $\mathfrak{d}_{i_j}$. This description represents any knowledge deriving from having i_j. We can now state the definition of a fair exchange protocol as explained in Asokan's seminal work on fair exchange [6]:

Definition 1 (Fair Exchange Protocol)**.** *An exchange is* fair *if follows the properties:*

1) Effectiveness*: the protocol is effective if when participants P and Q behave correctly, at the end of the exchange party P has i_Q and Q has i_P.*
2) Strong Fairness*: when the protocol has completed, either P has i_Q and Q has i_P or neither party has any additional information about the other's item (e.g. P has neither i_Q nor $\mathfrak{d}_Q$).*
3) Timeliness*: the protocol is timeliness if it completes in a finite amount of time.*

One way to achieve a fair exchange is by having a "simultaneous exchange". Unfortunately, this can not be done on the Internet, since it is an asynchronous (or semi-synchronous) system, see Section II-D.

G. Zero Knowledge Proofs

Zero-knowledge proofs, denoted as ZKP, are a class of cryptographic protocols that allow a party, the prover $\mathcal{P}$, to demonstrate the truth of a statement to another party, the verifier $\mathcal{V}$, without revealing any information beyond the veracity of the statement itself [12]. The statement in question is referred to as S.

1) Components of Zero-Knowledge Proofs.: In the context of zero-knowledge proofs, the roles and elements are as follows:

- **Prover** ($\mathcal{P}$): This is the entity that possesses knowledge of a secret or a fact and wishes to convince the verifier of its truth without revealing the secret itself.
- **Verifier** ($\mathcal{V}$): The verifier is the party that needs to be convinced of the truth of the prover's statement. The verifier checks the evidence provided by the prover but gains no additional knowledge about the secret or fact itself.
- **Statement** (S): The statement is the specific fact or assertion that the prover wants to convince the verifier is true. This could be a claim of knowledge of a secret key, the correctness of a computation, or any other verifiable proposition.

2) Non-Interactive Zero-Knowledge Proofs (NIZKPs): There are tow kinds of ZKP: *interactive* and *non interactive*. In interactive ZKP,$\mathcal{P}$ and $\mathcal{V}$ engage in a multi round routine to reach a verdict on the truth of S. On the other hand, in a non-interactive ZKP, the interaction between $\mathcal{P}$ and $\mathcal{V}$ is reduced to a single message. Formally:

Definition 2. *A Non-Interactive Zero Knowledge Proof system can be semi-formally defined as a tuple* $(\mathrm{Setup}, \mathrm{Prove}, \mathrm{Verify})$ *such that:*

- $pp \leftarrow \mathrm{nizkp.Setup}(1^\lambda)$*: Given a security parameter* λ*, the* Setup *algorithm generates public parameters* pp *that are shared between* $\mathcal{P}$ *and* $\mathcal{V}$*.*
- $\pi \leftarrow \mathrm{nizkp.Prove}(S, w, pp)$*: Given an input statement* S*, a witness* w*, and the public parameters* pp*,* $\mathcal{P}$ *is able to produce a proof* π*, without revealing any information about the witness* w*.*
- $\{0,1\} \leftarrow \mathrm{nizkp.Verify}(x, \pi, pp)$*: Given* S*, a proof* π*, and the public parameters* pp*,* $\mathcal{V}$ *determines whether* π *is valid (*1*) or invalid (*0*), without learning any information about the witness* w*.*

3) Simulation-Based Nature of ZKPs: A fundamental aspect of zero-knowledge proofs is their reliance on simulation. In ZKPs, it is not feasible for $\mathcal{P}$ to convince a third party, different from the designated verifier, of the validity of a proof. This is because the convincing power of a ZKP is intrinsically linked to the interaction between the prover and the verifier, or in the case of NIZKPs, to the specific setup phase and the established pp. The simulated interaction in interactive ZKPs, or the tailored construction in NIZKPs, ensures that the proof is valid only within the context of the established protocol and cannot be reused or repurposed to convince an unintended $\hat{\mathcal{V}}$.

On the one hand, the simulation-based approach is a cornerstone of the security and privacy features of ZKPs. It ensures that $\mathcal{V}$ is convinced of the truth of S while being unable to replicate or reuse π to convince others, thus maintaining the confidentiality and integrity of the underlying secret or fact. On the other hand, we will see in Section V-B how this creates a vulnerability for some OC protocols.

4) Properties of Zero-Knowledge Proofs.: Zero-knowledge proofs are characterized by three fundamental properties: completeness, soundness, and zero-knowledge. We briefly introduce the properties here, for a rigorous treatment see e.g. Chapter 4 from [13].

- **Completeness**: This property ensures that if the statement is true, a honest prover $\mathcal{P}$ can always convince the honest verifier $\mathcal{V}$. In other words, a legitimate proof will not be wrongly rejected. Formally, for a true statement S, the probability that $\mathcal{V}$ accepts the proof from $\mathcal{P}$ is high.
- **Soundness**: Soundness guarantees that if the statement is false, no dishonest prover can convince the honest verifier of its truth. Essentially, a false statement cannot be successfully proven. For a false statement S, the probability that $\mathcal{V}$ accepts a proof from a dishonest prover is negligible.
- **Zero-Knowledge**: This property is central to ZKPs and states that the verifier learns nothing beyond the fact that the statement is true. That is, the proof does not reveal any additional information about the statement or the secret involved. Formally, for every strategy of the verifier, there exists a simulator that can produce a transcript indistinguishable from a real interaction, without access to the prover's secret.

These properties collectively ensure that ZKPs are effective, reliable, and secure cryptographic tools.

5) Zero-Knowledge Proofs for NP Problems.: A remarkable feature of zero-knowledge proofs is their universality: ZKPs exist for any problem in the complexity class NP (Non-deterministic Polynomial time). This was first demonstrated by Goldwasser *et al.* in their seminal work [14]. In essence "all languages in NP possess zero-knowledge proofs" [14], i.e. any decision problem where a solution can be verified in polynomial time has a zero-knowledge proof that can convince $\mathcal{V}$ of the existence of a solution without revealing the solution itself. This is a fundamental property of ZKPs that is used for Verifiable computations (Section IV-B1d) and OC-protocols based on Zero-knowledge Contingent Payments (Section IV-A).

III. Outsourcing Computation

In this section, we start by introducing the Actors involved in a Generic OC Protocol and we follow by introducing the steps needed for a complete workflow. Then we formalize the properties needed for a correct OC protocol and we use them to prove the impossibility theorem and its corollary.

A. Actors

In scenarios involving external computation or services facilitated through smart contracts (as defined Section II-C), three primary actors play pivotal roles: the Job Provider, the Worker, and the Job Verifier. Throughout the rest of the paper, we identify the Actor with the process/device dealing with requested actions.

a) Job Provider: The Job Provider (P) is an entity that requires certain tasks or computations to be executed outside the blockchain ledger $\mathbf{L}$. This actor possesses the necessary resources to remunerate for these services.

b) Worker: The Worker (W) is the actor responsible for performing the job outlined by P. This can involve various forms of computation or task execution, which occur outside the blockchain ledger $\mathbf{L}$.

c) Job Verifier: The Job Verifier (V) acts as a coordinator of the interaction between P and V and validator of the work performed by W. This role may be assumed by the smart contract $\mathbf{Sc}$ itself, which automatically verifies the submitted results against predefined criteria and conditions. Alternatively, V can be an external entity, like a trusted third-party, serving as a mediator or overseer.

B. OC-protocol steps

Every Outsourcing Computation (OC) protocol inherently comprises a multi-step process, characterized by a sequence of procedures that are contingent upon the roles of the participating actors. The specific order and execution of these steps, as well as their presence, are dependent on the OC methodology employed. Notably, certain actions within these protocols may be undertaken either by the Job Provider (P) or the Job Verifier (V), varying according to the chosen protocol. To accommodate this variability and provide clarity, these actions are categorized and detailed in separate subsections, and *not* in chronological order. Figure 2 offers a schematic representation of the flow across different OC methods, highlighting the dynamic interplay and sequence of actions as executed by the respective actors within the diverse OC protocols.

1) Job Provider-Specific Actions:

a) Initiation: The process begins with P identifying a need for a task or computation that must be executed outside the blockchain ledger $\mathbf{L}$. P prepares a comprehensive description of the job: a defined function $f : A \to B$ that formally define the task, an input x on which to perform f, specific requirements (e.g. technical specifications or deadlines) and/or criteria for successful completion $\mathfrak{d}_{\mathrm{P}}$, and the associated reward conditions $\mathfrak{d}_{\mathrm{W}}$ for executing the job, expressed in the native currency (*coins*) of the ledger $\mathbf{L}$ or the time limits to issue the rewards. A job proposal $\mathbf{J}$ can be seen as a tuple:

$$\mathbf{J} = (f, x, \mathfrak{d}_{\mathrm{P}}, \mathfrak{d}_{\mathrm{W}})$$

b) Deploying $\mathbf{J}$*:* To deploy a job, P encapsulates $\mathbf{J}$ into a smart contract $\mathbf{Sc_J}$. Then P performs $\mathrm{deploy}_{\mathbf{L}}(\mathbf{Sc_J})$, effectively broadcasting the job proposal to the network. When $\mathbf{J}$ is clear from context, we will omit that and simply write $\mathbf{Sc}$. Once deployed, the $\mathbf{Sc_J}$ becomes visible to all participants in the network.

2) Worker-Specific Actions: The next critical actions we deal with for a OC protocol involve the Worker (W) delivering the results of the computation. These actions are formalized as follows:

a) Job Acceptance and Execution: Upon reviewing the job proposal $\mathbf{Sc_J}$, Wevaluates and decides to undertake the task defined by the job $\mathbf{J}$. The acceptance of the job by W is a commitment to perform the computation $f(x)$ as specified in the job proposal. W signals its willingness to perform the job.

b) Result and Proof Preparation: After completing the computation needed for job $\mathbf{J}$, W prepares the result $y = f(x)$, which is the output of the computation function f applied to the input x. Along with the result y, W may also be required to provide a proof of correct execution π. This proof is essential to demonstrate that the computation was performed correctly and according to the specified requirements $\mathfrak{d}_{\mathrm{P}}$ of the job. In practice it means that either the W or a third party (e.g. Intel in case of Intel SGX as TEE, see Section IV-B1b) has to compute the specification of y, $\mathrm{desc}(y)$ (we use here the notation used by Pagnia *et al.* [10]), and therefore the proof π for solution y is:

$$\pi = \{y \in B \wedge y = f(x) \wedge \mathrm{desc}(y) = \mathfrak{d}_{\mathrm{P}}\} \tag{1}$$

It is important to underscore that the necessity for a proof in OC protocols is not universal. Detailed exploration of scenarios where proofs are not requisite within OC protocols will be delineated in Section IV-B.

c) Proof Submission: There are different methods to submit a proof, depending on the channel used. W may execute a transaction to $\mathbf{Sc}$ in ledger $\mathbf{L}$ that includes at least π, or W may send π in a channel $\mathbf{C}$, itself a different ledger $\mathbf{L}'$.

d) Result Submission: Depending on $\mathbf{C}$ used to transmit π, y is provided in $\mathbf{Sc}$ together with π or separately. In the latter case y may be encrypted with a decryption key embedded in π and visible only during verification. In case y is encrypted we say that the result is *committed*.

As such the actions Proof Submission and Result Submission are neither atomic nor sequential.

3) Job Verifier-Specific Actions:

a) Open Proof/Solution: Upon successful verification of π, V is able to open the solution. This is a *revealing* of the solution. Generally the payment involves a release of a key to decrypt the result.

b) Payment: Depending on the OC-protocol, the payment occurs either if the job state is updated to `done`, or after the smart contract has been deployed. In any case, payment is always managed by V, contingent upon the successful verification of W's delivery. If the solution is not visible yet by P, then the payment is done in a way that release the necessary secrets needed by P to open the solution.

4) Actions Not Specific to Any Actor:

a) Verification of Proof: Upon receipt of the proof, it is the responsibility of V to verify it. It's important to note that in certain OC protocols, P and V are the same entity. Therefore, depending on the specific protocol, this action may or may not be executed concurrently with the payment to W.

b) Verification of Result: Similar to the verification of proof, the task of verifying the result is assigned to V. This action is dependent on the method used, and may not be necessary if the proof π already includes the proof of correct computation.

C. Formalization of Correct OC

The goal of outsourcing computation can be seen as the problem of synchronization of two processes $proc_1$ and $proc_2$. In practice, we want $proc_1$ to be the (successful) verification of the proof of computation π or the solution y depending on the model used and $proc_2$ be the payment to the worker $\mathbf{W}$. Formally:

Definition 3 (Model)**.** *We say that a model M is a tuple*

$$M := (\mathcal{A}, \alpha, \nu, \mathrm{TX}_{\alpha}^{\mathbf{C}}, \mathrm{TX}_{W}^{\mathbf{L}}) \tag{2}$$

such that:

- *$\mathcal{A}$ is the set of actors involved: $\mathcal{A} = \{P, W, V\}$*
- *Actor α is $\alpha \in \{P, V\}$*
- *What α verifies is $\nu \in \{y, \pi\}$*
- *$\mathrm{TX}_{\alpha}^{\mathbf{C}}$ represents the fact that ν is sent through $\mathbf{C}$ and verified by α*
- *$\mathrm{TX}_{W}^{\mathbf{L}}$ represents the fact that V pays W on $\mathbf{L}$ via transaction $\mathrm{TX}_{W}^{\mathbf{L}}$*

If not said otherwise, we assume M models an asynchronous system (see Section II-D)

It is easy to understand the notation $\mathrm{TX} \in \mathbf{L}$ meaning that transaction TX has been accepted as valid and included by the consensus in ledger $\mathbf{L}$. We abuse this notation and write $\mathrm{TX} \in \mathbf{C}$ meaning that transaction TX has been (received and) accepted as valid by the "member of consensus" of the channel $\mathbf{C}$, i.e. α and $\mathbf{W}$. We now state the definitions needed to define a correct OC protocol, adapting the notation used in [10]. To keep track of malicious activities from one of the actors, we use the boolean variable m_{β} with $\beta \in \mathcal{A}$.

Definition 4 (Effectiveness)**.** *In a model M as defined in Definition 3, if both α and W behave correctly, $\mathrm{TX}_{\alpha}^{\mathbf{C}}$ and $\mathrm{TX}_{W}^{\mathbf{L}}$ are valid and consistent descriptions then both will be included in $\mathbf{C}$ and $\mathbf{L}$ respectively. If either of the transactions are not as expected, then both parties abort. Formally:*

$$\begin{aligned}(\mathrm{desc}(\mathrm{TX}_{\alpha}^{\mathbf{C}}) = \mathfrak{d}_{\boldsymbol{P}} \wedge \mathrm{desc}(\mathrm{TX}_{W}^{\mathbf{L}}) = \mathfrak{d}_{\boldsymbol{W}}) \wedge (m_{\alpha} = m_{\boldsymbol{W}} = \perp) \\ \implies \mathrm{TX}_{\alpha}^{\mathbf{C}} \in \mathbf{C} \wedge \mathrm{TX}_{W}^{\mathbf{L}} \in \mathbf{L}\end{aligned}$$

$$\begin{aligned}(\mathrm{desc}(\mathrm{TX}_{\alpha}^{\mathbf{C}}) \neq \mathfrak{d}_{\boldsymbol{P}} \vee \mathrm{desc}(\mathrm{TX}_{W}^{\mathbf{L}}) \neq \mathfrak{d}_{\boldsymbol{W}}) \wedge (m_{\alpha} = m_{\boldsymbol{W}} = \perp) \\ \implies \mathrm{TX}_{\alpha}^{\mathbf{C}} \notin \mathbf{C} \wedge \mathrm{TX}_{W}^{\mathbf{L}} \notin \mathbf{L}\end{aligned}$$

Definition 5 (Atomicity)**.** *There are no outcomes in which $\mathrm{TX}_{\alpha}^{\mathbf{C}} \in \mathbf{C}$ but $\mathrm{TX}_{W}^{\mathbf{L}} \notin \mathbf{L}$ or $\mathrm{TX}_{W}^{\mathbf{L}} \in \mathbf{L}$ and $\mathrm{TX}_{\alpha}^{\mathbf{C}} \notin \mathbf{C}$*

$$\neg \left(\left(\mathrm{TX}_{\alpha}^{\mathbf{C}} \in \mathbf{C} \wedge \mathrm{TX}_{W}^{\mathbf{L}} \notin \mathbf{L} \right) \vee \left(\mathrm{TX}_{W}^{\mathbf{L}} \in \mathbf{L} \wedge \mathrm{TX}_{\alpha}^{\mathbf{C}} \notin \mathbf{C} \right) \right)$$

Definition 6 (Timeliness)**.** *Any process $proc$ that behaves correctly will eventually write a valid transaction TX_{proc}, to its channel $\mathbf{C}$.*

We can now state the definition of Correct Outsourced Computation:

Definition 7 (Correct OC)**.** *A* Correct Outsourced Computation *protocol is the synchronization of two events:*

1) *transaction $\mathrm{TX}_{\alpha}^{\mathbf{C}}$ is successfully exchanged on $\mathbf{C}$ from W to α*
2) *transaction $\mathrm{TX}_{W}^{\mathbf{L}}$ from V to W is written on $\mathbf{L}$*

in an Effective, Atomic and Timeliness manner.

Algorithm 1 Fair Exchange using a generic OC protocol

```
setup(C, L, TX^C_α, TX^L_W, 𝔡_P, 𝔡_W)
if m_W = false then
    commit_W(TX^C_α)                    ▷ W transfers ν to α
end if
if verify(TX^C_α, 𝔡_P) and m_P = false then
    send_P(TX^L_W)                      ▷ α transfers c to W
else
    abort()                             ▷ α does not transfers c to W
end if
if verify(TX^L_W, 𝔡_W) = false then
    reveal_W(TX^C_α)                    ▷ W reveals the solution in ν to α
else
    abort()                             ▷ W invalidates ν
end if
```

D. Impossibility of OC without a Trusted Third Party

We proceed to show that outsourcing computation (OC) is impossible in the asynchronous setting without a trusted third party (TTP) by reducing OC to Fair Exchange

Lemma 1. *Let M be a model as defined in Definition 3. Let O be a protocol which solves OC as defined in Definition 7 in M. Then there exists a protocol S which solves Fair Exchange (Definition 1) in M.*

Proof. (Sketch) Consider two parties, α and $\mathbf{W}$, which are part of a fair exchange. Specifically α owns an item ν and wishes to exchange it against an item c. Algorithm 1 presents a protocol for Fair Exchange between α and $\mathbf{W}$ using OC. In particular, $\mathrm{TX}_{\alpha}^{\mathbf{C}}$ assigns ownership to α of ν and $\mathrm{TX}_{\mathbf{W}}^{\mathbf{L}}$ assigns ownership of c to $\mathbf{W}$. Therefore $\mathrm{TX}_{\alpha}^{\mathbf{C}}$ must be (validated and) included in $\mathbf{C}$ and $\mathrm{TX}_{\mathbf{W}}^{\mathbf{L}}$ must be included in $\mathbf{L}$ to correctly execute the exchange. In other words, if $\mathrm{TX}_{\alpha}^{\mathbf{C}} \in \mathbf{C}$ and $\mathrm{TX}_{\mathbf{W}}^{\mathbf{L}} \in \mathbf{L}$, then α receives desired ν and $\mathbf{W}$ receives desired c, i.e. α and $\mathbf{W}$ fairly exchange ν and c.

Observe that the definition of Timeliness (Definition 6) and Effectiveness (Definition 4) in OC are equivalent to the definition of Timeliness and Effectiveness in fair exchange protocols, as defined in Definition 1 and that Atomicity in Definition 5 is equivalent to Strong Fairness in Definition 1. Since Algorithm 1 clearly satisfies properties in Definitions 4, 5 and 6, then it satisfies properties in Definition 1 and therefore a OC protocol can be used to create a Fair Exchange one. □

We now state a second lemma that is necessary to prove the impossibility theorem:

Lemma 2. *[Corollary 1 in [10]] There is no strong fair exchange protocol tolerant against misbehaving nodes without a trusted third party.*

We are now ready to state the impossibility theorem:

Theorem 1. *There exists no asynchronous OC protocol tolerant against misbehaving nodes without a trusted third party.*

Proof. Assume there exists an asynchronous protocol O which solves OC. Then, due to Lemma 1 there exists a protocol which solves strong fair exchange. This is a contradiction, since from the impossibility Lemma 2 we know it is impossible to have fair exchange in a (closed) asynchronous model without a third party As this is a contradiction, there cannot exist such a protocol O. □

Following te works in [6], [15], we recognize that, our findings are only applicable to the closed model of Definition 3. We now proceed to show how it also works for the open model.

In the open model, the system (or environment) has the ability to compel α and $\mathbf{W}$ to make a decision, meaning transactions can be executed on their behalf in the event of a crash [16]. In the context of OC, this suggests that the distributed system $\mathbf{L}$, or to be more precise, the consensus of $\mathbf{L}$, can inscribe $\mathrm{TX}_{\mathrm{W}}^{\mathbf{L}}$ to $\mathbf{L}$ on behalf of $\mathbf{W}$. In this scenario, the consensus of $\mathbf{L}$ assumes the role of a TTP, as detailed in Section II-E.

However, a question arises regarding how the consensus participant(s) of $\mathbf{C}$ become aware that $\mathrm{TX}_{\alpha}^{\mathbf{C}}$ has been transmitted (as explained in Section II-B, $\mathbf{C}$ may be a simple secure channel between parties relaying messages, or a ledger $\mathbf{L}'$). In practical terms, a smart contract can only perform actions based on some input. Therefore, before inscribing $\mathrm{TX}_{\mathrm{W}}^{\mathbf{L}}$, the contract or consensus of $\mathbf{L}$ must observe and verify that $\mathrm{TX}_{\alpha}^{\mathbf{C}}$ was included in $\mathbf{C}$. Consequently, a protocol achieving OC must make one of the following assumptions: either there exists a TTP that will ensure correct execution of OC; or the protocol assumes α, or $\mathbf{W}$, or some other honest, online party (this can again be consensus of $\mathbf{L}$) will always deliver a proof for $\mathrm{TX}_{\alpha}^{\mathbf{C}}$ to $\mathbf{L}$ within a known, upper bounded delay δ, i.e., the protocol introduces some form of synchrony assumption (Section II-D). As argued in [10], we observe that introducing a TTP and relying on a synchrony assumption are equivalent:

Corollary 1. *In the design of an OC protocol, a decision must be made between introducing a trusted third party, or, equivalently, assuming some level of network synchrony.*

Proof. (Sketch) Assume that process of $\mathbf{W}$ does not crash and hence submits the necessary proof through $\mathbf{C}$, and that this message is delivered to the smart contract within a known upper bound δ, then we can be confident that OC will occur correctly. In this scenario, $\mathbf{W}$ (if $\mathbf{C}$ is a ledger) or α (if $\mathbf{C}$ is not a ledger) effectively acts as trusted third party.

Conversely, if we cannot make assumptions on *when* the message will be delivered to the smart contract, as is the case in the asynchronous model, a trusted third party is necessary to determine the outcome of the OC: the TTP observes $\mathrm{TX}_{\alpha}^{\mathbf{C}}$ and informs the smart contract or directly enforces the inclusion of $\mathrm{TX}_{\mathrm{W}}^{\mathbf{L}}$ in $\mathbf{L}$. This demonstrates how a TTP can be used to enforce synchrony, i.e., timely delivery of messages, in OC protocols. While the two models yield equivalent results, the choice between a TTP and network synchrony impacts the implementation details of an OC protocol. □

IV. OC Classification Framework

With the impossibility result in Theorem 1 and the OC model (Section III) in mind, we now introduce a new framework to classify and evaluate OC protocols. By leveraging Corollary 1, we can identify three kinds of OC protocols:

1) OC Protocols that introduce synchronicity, so that P and V are the same actor. In this case the blockchain used as a Synchronization device, more that a TTP (Section IV-A).
2) OC Protocols that introduce one or more trusted third parties as V, distinct from P (Section IV-B)
3) Hybrid OC Protocols that introduce a third party V that intervenes only when some of the actions are not performed honestly, e.g. a proof verification returns `false` (Section IV-C).

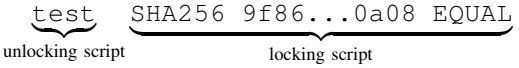
The classification framework introduced below is structured as follows: for each of three kinds of OC Protocols, we introduce the main components that characterize the kind, then we explicit its flow and finally explain the vulnerabilities of the kind of OC Protocol. This enables systematic classification of existing protocols and, at the same time, acts as a step-by-step guide for creating new OC schemes.

A. Synchronous Protocols

1) Main Components:

a) HTLC: Introducing synchronicity means creating a way to "block" the execution of the smart contract until a specific piece of data is entered. One method that has been successful at dealing with the synchronicity problem in Bitcoin and is currently used in ZCKP has been the *hash-time lock* contract (HTLC). Proposed by Tier Nolan for the atomic swap in the field of blockchain interoperability [17], the HTLC combines a *hash-lock* construct with a *time-lock* one.

The hash-lock in Bitcoin is a combination one opcode (i.e. Script language function) to compute the hash on the stack, the hash itself and one opcode to verify the equality of the hashes in the locking script together with the preimage of the hash as the unlocking script:



Combined with a time-lock [18], it becomes:

```
IF
    <hashlock>
ELSE
    <expiry time> CHECKLOCKTIMEVERIFY DROP
ENDIF
```

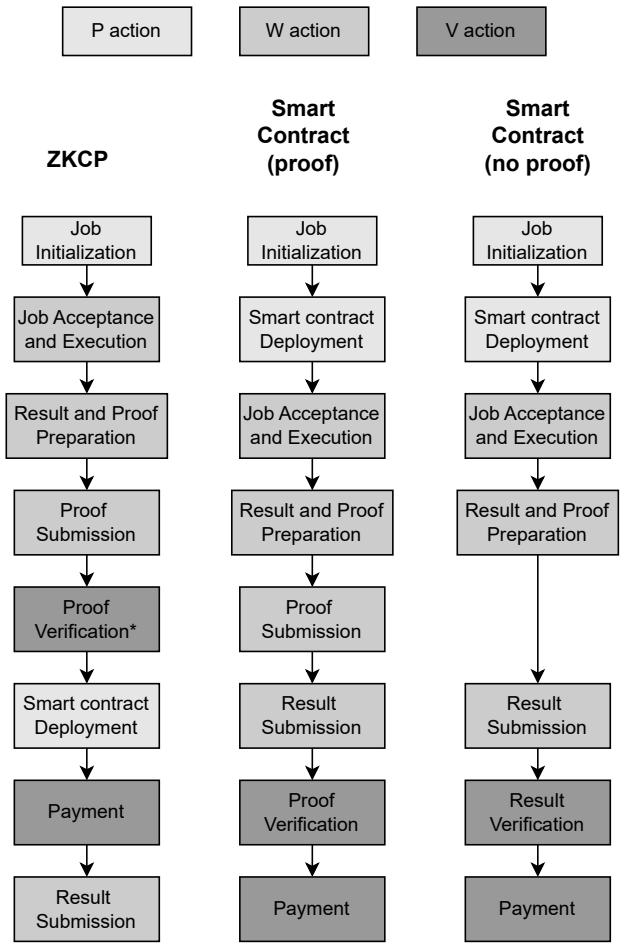


Fig. 2. Different flows in OC management between ZKCP model and the Smart Contract model. *: While the proof of verification is done by the job provider, in that specific case since the job provider and the job verifier coincide, we can assume that the job provider is acting as a verifier in that moment.

b) ZKCP: The synchronous methods that use it are the Zero Knowledge Contingent Payments (ZKCP) and their derivatives [19]–[24]. We formally define what a ZKCP protocol is and then we comment it.

Definition 8 (ZKCP ([24]))**.** *A* ZKCP *protocol is a tuple of algorithms* $(\text{Setup}, \text{Prove}, \text{Verify})$ *and a routine* $\mathcal{F}_{ex}[COM]$ *so defined:*

- $pp \leftarrow \text{Setup}(1^\lambda)$*: It is* $\text{nizkp}.Setup(1^\lambda)$ *of Definition 2*
- $(c, E, d, \pi) \leftarrow \text{Prove}(pp, s)$*: It is the prove algorithm that takes input as: the public parameter pp, the digital file s. It then outputs the ciphertext c, a commitment-opening pair* (E, d) *and a proof* π*.*
  - *Sample a random key* $k \leftarrow 0, 1^\lambda$*;*
  - *Generate* $c \leftarrow \text{Enc}(s, k)$*, where* Enc *is a symmetry encryption;*
  - *Commit to the key* $(E, d) \leftarrow COM.Commit(k)$
  - *Generate* $\pi \leftarrow \text{nizkp.Prove}(pp, (c, E), (k, s))$*;*
  - *Output* (c, E, d, π)*.*
- $b \leftarrow \text{nizkp}.Verify(pp, c, E, \pi)$*. See Definition 2*
- $\mathcal{F}_{ex}[COM]$*. It is the the trusted exchange functionality to guarantee the fairness and security of the payment.*

Note that $\mathcal{F}_{ex}[COM]$ may be the combination of the HTLC with exchange of messages through a channel $\mathbf{C}$. In Section VI-B2 we propose alternatives to the HTLC. We briefly explain how $\mathcal{F}_{ex}[COM]$ works in Section IV-A2.

2) Flow: With reference to Figure 2(a), we present the main flow of OC protocols that follow the synchronous model.

In this first phase of the OC protocol, $\mathbf{W}$ accepts the job $\mathbf{J}$ and start the execution. After having executed the job and having obtained a solution, the worker symmetrically encrypts the solution with a key K and creates a proof of correct computation π, as seen in Equation (1). Note that this zero knowledge proof is non-interactive (Definition 2), which means that there is no assurance that the proof has been provided by the effective worker, as explained in Section II-G3.

Upon completing the proof, the worker submits it to the job provider, who, as previously mentioned in the context of the synchronous model, also serves as the verifier.

Once the job provider receives the encrypted solution along with the proof of computation, they proceed to validate the proof's accuracy. Here, the job provider generates a hash using the symmetric key and the worker's address, then forms a HTLC and dispatches it to the blockchain. While synchronous methods are predominantly utilized in Bitcoin, they can also be adapted for more sophisticated, smart contract-enabled blockchains.

At this juncture, the job provider awaits the worker to submit the pre-image of the hash, denoted as K, which is the key originally used for encrypting the solution. If the worker does not provide K (and therefore does not get the payment), $\mathbf{P}$ can get spend the coins in the HTLC after a deadline, maintaining Atomicity.

3) Vulnerabilities: The main problem of synchronous models is the certainty of payments. Since the payment itself is ultimately enabled by $\mathbf{P}$ which deploys the smart contract HTLC and has to wait for the hash from $\mathbf{W}$, the success of a payment is in the hands of $\mathbf{P}$. In this case $\mathbf{P}$ can perform a griefing attack similar to the one executed in atomic swaps that relies on synchronicity methods. See for example [25], [26]. While the atomicity property is still respected, since $\mathbf{P}$ cannot open the result of the computation without the password, $\mathbf{W}$ still suffers, because he either lose time in executing the job, and also incurs into costs related to the computation itself.

$\mathbf{P}$ similarly suffer from this attack. On the one hand, $\mathbf{P}$ cannot delegate their work to multiple workers, since in case of successful execution, he has to pay all of them, while, on the other hand, he has to trust that $\mathbf{W}$ will deliver the result of the computation.

Another problem that may arise is due to the fact that the proof comes from Zero Knowledge schemes that are not interactive. As it's widely known, non-interactive, Zero Knowledge, proofs schemes from party A to party B can not be transferred to another party C, since by definition, they are no different from a simulation (Section II-G3). In this sense,

if P asks for a computation and inputs that have been done before, he may suffer from a replay or "copy" attack. We describe the problem in Section V-B.

B. TTP-based Protocols

1) Main Components:

a) Smart Contracts: Smart contracts (Section II-C) are the most preferred method in the literature to obtain a decentralized TTP, see e.g. [27]–[30], especially in case of blockchain whose set of validators/miners is conspicuous and spread across multiple geographical regions among other parameters (see e.g. [31]–[33] for an overview on measuring blockchain decentralization). A smart contract generally acts as a coordinator, ensuring P and W follow the protocol. It is also in charge to atomically make the payment in case the solution or the proof of solution are verified correctly. In this sense, the smart contract is a verifier V through the execution of a OC protocol: this is different from the synchronous case, where the contract HTLC only acts as payment vehicle.

In order to be able to do all these actions, smart contracts are generally employed on an EVM-compatible blockchain. In fact, the amount of opcodes/commands needed to coordinate the flow and verify proofs is much higher than those provided by the current iteration of the Script language of Bitcoin. Consequence of this complexity is that operations generally costs much in the in terms of native currency of the blockchain employed. We will analyze this problem in Section V-A.

b) TEEs: Intel SGX, a suite of CPU extensions, provides robust hardware-supported TEE (or enclave) capabilities. Each enclave comes with a safeguarded address space, immune to access attempts by any non-enclave code. The enclave memory is encrypted with the processor's key before leaving its confines. Intel SGX includes attestation mechanisms. These allow an attesting enclave to prove its correct instantiation to a validator [34], and to form a secure, authenticated channel for safely transmitting sensitive data.

In the context of the TEE-enabled OC protocols, Intel SGX [35] plays a pivotal role by offering verified execution and data secrecy for computations that are outsourced. Any outsourced program must be compatible with SGX, meaning it must natively support SGX enclave execution. Meanwhile, methods like Haven [36] and Panoply [37] that facilitate enclave execution for existing, unaltered legacy applications, exist separately from the OC protocol used.

TEEs generally cover the role of workers in OC protocols and depending from the protocol used, they may release a proof that is verified by a smart contract before releasing the entire solution. Protocols that use TEEs generally are OC protocols that use at least two trusted third parties: one is the blockchain/smart contract while the other is the company issuing the attestation.

In the literature, TEEs are either treated singularly for a single worker [38], or as a cluster of nodes that perform the computation, similarly to marketplace of computation [39]

c) Replication: Verifiable computation through replication [27], [29], [40]–[42] is performed by delegating computation to multiple workers, and then compare their solutions. If the solutions are equal, then P assumes that the computation has been done correctly, and therefore P may decide to compensate both equally. On the other hand, if the results of the computation are different from each other, P has to assess which of those workers has cheated or is faulty.

Similar to the case of trusted execution environments in replication based protocols, the workers can either provide the proof or provide the solution of the computation directly.

d) Verifiable Computation: The genesis of the Verifiable Computation concept can be traced back to the pioneering work of Babai *et al.* [43], where it has been explored under a spectrum of terminologies such as "checking computations", as initially introduced by Babai, along with "delegating computations" in Goldwasser [44] or "certified computation" by Micali [45] and "verifiable computing" in Gennaro *et al.* [7] and Parno *et al.* [46].

Verifiable Computation enables a computationally weak client to outsource the computation of a function f on input x to one or more workers. The particularity of this method is that the workers must return the result of the function evaluation, $y = f(x)$, together with a proof π that the computation of f was carried out correctly on the given value x. Note that, the creation and verification of π should require substantially less computational effort than computing $f(x)$ from scratch, an issue we analyze in Section V-A.

We now formally define a Verifiable Computation protocol using [7]:

Definition 9 (Verifiable Computation Scheme [7])**.** *A verifiable computation scheme* $\mathcal{VC} = (KeyGen, ProbGen, Compute, Verify)$ *consists of the four algorithms defined below.*

- $\mathrm{KeyGen}(F, \lambda) \rightarrow (PK, SK)$: *Based on the security parameter* λ*, the randomized key generation algorithm generates a public key* PK *that encodes the target function* F*, used by the worker to compute* F*. It also computes a matching secret key* SK*, which is kept private by the client.*
- $\mathrm{ProbGen}_{SK}(x) \rightarrow (\sigma_x, \tau_x)$ *: The problem generation algorithm uses the secret key* SK *to encode the function input* x *as a public value* σ_x *which is given to the worker to compute with, and a secret value* τ_x *which is kept private by the client.*
- $\mathrm{Compute}_{PK}(\sigma_x) \rightarrow \sigma_y$ *: Using the client's public key and the encoded input, the worker computes an encoded version of the function's output* $y = F(x)$*.*
- $\mathrm{Verify}_{SK}(\tau_x, \sigma_y) \rightarrow y \cup \perp$ *: Using the secret key* SK *and the secret "decoding"* τ_x*, the verification algorithm converts the worker's encoded output into the output of the function, e.g.,* $y = F(x)$ *or outputs* $\perp$ *indicating that* σ_y *does not represent the valid output of* F *on* x*.*

Example of verifiable computation used in TTP-based methods can be found in [41], [47].

2) Flow: When V is a smart contract, either the P or another third party (e.g. a "broker" [39]) must deploy it. In the smart contract, the deployer will also explicit the description of the job specification and the logic to verify it.

Workers' acceptance of the job is explicit in this case and generally is a transaction to the smart contract through **L**. Depending on the protocol, the job may have concurrent delegations (for example in the case of verification by replication [27]), or just one opening (as in the case of employing workers using trusted execution environments [38], [39]). After accepting the job, the worker(s) starts executing the computation.

After the execution, depending on the protocol, (each) W may provide a proof of correct computation that follows the specification of the job assignment or skip this step entirely. The proof can be explicit (the attestation done by the trusted execution environment) or may be in the form of a Zero Knowledge proof (similar to the ZKCP method explained in Section IV-A).

Then, W provides the solution, either in the form of a smart contract transaction, or as a payload exchanged through the channel instead of the ledger.

Proof verification is done by V. In this case V is the smart contract itself that, in case of successful verification, will atomically provide the payment to W.

3) Vulnerabilities: In those methods where the TTP skips the proof of correct computation entirely, the workers expose themselves to vulnerabilities such as the *copy attack* [40] (sometimes also called "lazy" attack [42]). In copy attacks, a malicious worker W_B waits for the other honest worker W_A to send the solution first and then W_B just copies the solution and resend it equally to the smart contract. By exploiting the asynchronicity of blockchain protocols, W_B aims to have its solution registered first on the blockchain. Since the attack is nearly costless, even if the probability of success is low, it is still worth doing. See Section V-C for more details.

Finally, even if the payment is atomic as long as the logic has been embedded into the smart contract, differently from the synchronous models, the job provider is vulnerable from the griefing attack from workers that commit to the execution of the competition and do not deliver (Section V-E).

C. Hybrid Methods

In some methods a third party is involved only if either P or W do not follow the protocol, as seen in e.g. [48]. In this case the cheater is penalized, generally in the form of a lost deposit that goes to W and/or V.

1) Main Components:

a) Smart Contracts: In hybrid methods, smart contracts serve a similar purpose as in the TTP methods (Section IV-B). An auxiliary smart contract can be deployed to handle situations where at least one actor behaves maliciously.

b) Mediators: Mediators are auxiliary entities that intervene when at least one actor exhibits malicious behavior. They can take the form of a smart contract, or external observers relative to the blockchain. Mediators can be human (such as arbitrators) or automated programs.

2) Flow: The flow in Hybrid Methods is equal to that of Synchronous Methods (Section IV-A2) or TTP based Methods (Section IV-B2). The difference is that if no solution is provided by W, then there is a penalization phase instead of a payment.

3) Vulnerabilities: The main vulnerability of hybrid methods, beside those of TTP and Synchronous ones, is in the working of Mediators and auxiliary smart contracts. It is easy to see that either smart contract vulnerabilities or collusion with the mediator may hinder the efficacy of the role.

V. Challenges

In Section IV we presented the different methods and mentioned their limitation and vulnerabilities. In this section, we explain them in more details, and in Section VI we propose new solutions to solve these difficulties.

A. Costs

The total costs for different OC protocols can be split between computational costs, and overhead cos . We define "overhead" as the set of all the cost not accounted as payments for the workers.

Let k be the milliseconds (ms) needed to complete the computation by a worker and n the number of workers. We first start with analyzing computational cost, and then we analyze the overhead costs. A summary can be found in Table I.

It is reasonable to assume that the amount of computation costs are linear in k. In fact, the more complex computation is, the more CPU time required to solve it since generally costs are measured on CPU time, then our assumption holds, we know that nearly all methods presented in Section III require one worker per task, the only exception are OC protocols based on replication which require n workers. Consequently, in this case P incurs in costs linear in $k \cdot n$.

We now analyze the overhead cost for the different OC protocols. The highest cost incurred by P are generally related to zero knowledge proof creation. Walfish *et al.* [49] present a performance evaluation of different verifiable computation methods (Figure 3 and 5 in [49]). Verification of computation can take as much as 10 times more of CPU time than the computation itself with a prover overhead cost going from 10^3 to 10^9 than the cost of the competition done in native C. It is clear that it is not possible to use verifiable computation for recurring or cheap tasks. ZKCP based OC-protocols may suffer from the same problems since the bulk of overhead costs are related to zero knowledge proof creation in verifiable computational protocols.

A more cost-effective approach may be to use a replication system as a suggested in [27], unless integrity or payment security are at stake. See Sections V-B, V-C and V-E for details.

In general, the only meaningful overhead of other methods presented in Section III is related to transaction fees.

Worker Type	Computation	Overhead	Total
ZKCP	$\mathcal{O}(k)$	$\mathcal{O}(\mathbf{poly}(k))$	$\mathcal{O}(\mathbf{poly}(k))$
Verifiable computation			
TEE	$\mathcal{O}(k)$	$\mathcal{O}(1)$	$\mathcal{O}(k)$
Replication	$\mathcal{O}(nk)$	$\mathcal{O}(1)$	$\mathcal{O}(nk)$

TABLE I
WORKER TYPE FOR OC-PROTOCOLS (LEFT) AND RELATIVE COSTS (RIGHT). NOTE THAT OVERHEAD COSTS FOR ZKCP AND VERIFIABLE COMPUTATION BASED PROTOCOLS ARE $\mathbf{poly}(k)$ SINCE THE COST ITSELF VARIES BASED ON THE ZK PROOF SYSTEM USED [49]

B. Collusion

One of the pervasive challenges in Outsourced Computation (OC) protocols is the potential for collusion among workers. This issue is particularly acute in replication-based protocols, where P cannot reliably ascertain whether delegated workers will collude. The incentives for such collusion are substantial, especially in tasks requiring significant computational resources. By collaborating, workers can effectively halve their workload while still receiving full compensation.

Some studies, such as Avizheh *et al.* [40], opt to disregard the possibility of worker cooperation, a premise we find overly optimistic. Conversely, other research endeavors propose strategies to mitigate this risk. For instance, Dong *et al.* [27] suggest requiring a collateral from all participating workers and implementing a supplementary smart contract $\mathbf{Sc}'$. This contract offers incentives for workers to defect from any collusion agreement. Specifically, if W_A colludes with W_B but subsequently reports W_B as a "cheating worker" through $\mathbf{Sc}'$, W_A stands to gain both the shared computational reward and an additional bounty for identifying the cheater. This arrangement creates a deterrent against collusion from a game-theoretic perspective, providing economic security in the absence of cryptographic guarantees (refer to Section V-F).

Collusion can also arise between W and V, particularly when the latter's role is not fully automated. Proposals addressing this issue, as discussed in works like Dang *et al.* [39], generally concern OC protocols reliant on trusted third parties (TTPs). This form of collusion is intrinsic to protocols that separate the roles of P and V, and is not present in synchronous protocols where these roles are unified. Addressing this type of collusion necessitates a fundamental re-design of the architecture of the system to eliminate the need for a TTP.

Lastly, collusion can occur between the P and V. This is especially pertinent in synchronous OC protocols, where they are the same entity, but can also be a concern in hybrid OC protocols. Such collusion adversely impacts workers, a topic further explored in Section V-E.

C. Sabotage

If collusion refers to the joint effort of W_A and W_B in a cooperative task, a counteractive phenomenon occurs when W_A engages in deliberate actions aimed at undermining W_B. This can manifest as sabotage, where W_A actively disrupts or subverts the work of W_B.

One notable method involves the copy attack delineated in Section IV-B3, wherein W_B pilfers the output of W_A and falsely claims it as their own. To counteract such attacks, the literature proposes novel approaches including commit-reveal schemes, potentially incorporating proofs akin to those found in synchronous OC protocols as in [40]. However, these schemes complicate the protocol design, potentially exacerbating the challenge of establishing security proofs. Notably, many replication-based OC protocols discussed in Section IV-B lack formal security validation and rely on game-theoretic analysis, see Section V-F.

Another strategy to deter "lazy" workers from appropriating computational results involves embedding unique worker identifiers within the payload or solution, subsequently encrypting this as part of the proof. If W_B expropriates the encrypted results from W_A and re-encrypts them, the P can employ methods like Public-Key Encryption with Equality Test (PKEET), as proposed by Yang [50]. PKEET enables verification of whether two ciphertexts, c_{pk} and $c_{pk'}$, encrypted with public keys pk and pk' respectively, originate from the same plaintext message m. This technique allows P to discern between legitimate and fraudulent solutions. However, integrating such methods into smart contracts, even those based on EVM, is not feasible. Consequently, proof verification may not lead to payment atomically, which leads to the issues outlined in Section V-E.

A recent innovation by Susillo *et al.* [42] introduces the concept of Public-Key Encryption with Equality Test against Lazy Encryptors (PKEET-LE) to mitigate the copy attack. This advancement adapts PKEET for use in Smart Contract-based OC protocols. Nevertheless, PKEET-LE imposes additional procedural steps in the execution of successful OC protocols, complicating the overall process.

D. Computational Expression

The main components used to obtain OC protocols directly affects the computational abilities of the job. For example, zero knowledge proofs based protocols are inherently limited in their expression due to the current limitation of the proof system itself. Furthermore, it may be too costly to have the verification of complications as seen in the Section V-A.

On the other hand, OC protocols based on TTP involvement, have more expression possibilities due to the fact that the competition is outsourced to general-purpose computational architectures. In this case W can do general purpose com-

putation since verification is either based on the replication system (Section IV-B1c), or the attestation system of TEEs (Section IV-B1b).

This is reflected on the goal of the proposals. For example, proposal based on ZKPs are designed to support a specific computation (e.g. polynomial evaluation [51] or matrix computations [30]). On the other hand, TEE-enabled proposals can be employed in nearly any kind of computation, as highlighted by the proposal of Dang *et al.* [39], where the authors create the design for a marketplace of computations where no explicit limit is set on the kind

E. Payment Certainty

To achieve a fair exchange in OC protocols, it is imperative to ensure the property of Atomicity, as defined in Definition 5. However, atomicity alone does not fully capture the satisfaction level of the interacting parties. Consider a scenario where W falls victim to a griefing attack by P, as possible in ZKCP based OC protocols discussed in Section IV-A. In such a case, W incurs a loss in terms of time and resources spent on the computation, without receiving any compensation, even though the P is unable to access the solution. This type of exploit, derived from the context of atomic swaps in literature, is termed a *griefing attack* (see, for instance, [52]). A symmetrical risk exists for P, targeted by a W-initiated attack, as explored in Section IV-A3. We propose possible mitigation strategies in Section VI-B1.

Furthermore, the griefing attack can be defined as 'security of payment' and mathematically expressed as

$$\mathbb{P}[\text{griefing attack}] = \mu \tag{3}$$

where μ is a neglegible function. Incorporating escrow or collateral mechanisms, coupled with penalties for malicious behavior (refer to [38]), further strengthens the security model.

Another innovative approach involves the use of micropayment channels, leading to an optimistic fair exchange or gradual release of secrets, as discussed in [39]. This system allows for incremental payments or information release, reducing the risk of total loss in case of a breach. Notably, such micropayment channels can be efficiently implemented in any Trusted Execution Environment (TEE) based system, including TEE-based payment channels like those described in [53]. The integration of TEEs provides an additional layer of security, ensuring that the computation and transaction processes remain tamper-proof and reliable.

The incorporation of mediators as an alternative to prevent cheating is another consideration. However, mediators, acting as trusted third parties, introduce their own set of risks, including the potential for collusion with either P or W, which could unfairly disadvantage the other party as shown in Section IV-B3. Therefore, while mediators can offer a solution, their use necessitate consideration of the additional risks they introduce.

F. Computational Security and Formal Treatment

One of the key challenges in comparing OC protocols across academic literature is the absence of a universally accepted framework for security assessment. In the existing body of work, the predominant theoretical foundations for analysis are game theory and cryptographic security. The use of game theory is particularly prevalent in replication-based OC protocols. However, despite assertions to the contrary in some studies, this approach generally yields, at best, economic security rather than rigorous security proofs. This is primarily due to the reliance on the assumption that all parties involved are rational actors. Such an assumption is frequently invalidated in practice, as participants may willingly incur losses in a specific OC protocol to achieve gains in an external system. This behavior might appear irrational when viewed solely within the confines of the OC protocol, but becomes rational when considering the broader strategic landscape.

Conversely, cryptographic-based formalism appears to be a more appropriate approach for OC protocols, as it does not hinge on any presumptions about human behavior. Nevertheless, even this method is not without its limitations. For instance, the series of papers on Zero-Knowledge Contingent Payment (ZKCP) [19]–[24] highlights several errors in earlier proofs, prompting subsequent research and refinements in the field.

Distinct from other replication-based OC protocols, the work cited in [41] represents a notable effort to formally prove the security of their system. This endeavor stands out because it combines the replication-based methodology with rigorous formal proof, an approach that is relatively uncommon in this domain.

Moreover, the concept of payment certainty in OC protocols, as discussed in the Section V-E, offers a compelling example of how cryptographic analysis can be applied. Here, payment certainty can be effectively modeled as a probabilistic concept (Equation (3)), allowing for a more nuanced and mathematically grounded analysis. This approach underscores the potential for cryptographic methods to provide robust, quantifiable security assurances in the context of OC protocols.

VI. Discussion

In the following, we present a discussion on enhancing the current state of the art. In the Section VI-A, we present the OC trilemma, direct consequence of the challenges we highlighted and treated in Section V. In the Section VI-B, we give our suggestions for new directions on blockchain enhanced OC protocols.

A. OC Trilemma

In analyzing the challenges inherent in Outsourced Computation (OC) protocols, we identify a fundamental trilemma involving payment certainty, security of results, and expressive (general purpose) computation. This trilemma, illustrated in Figure 3, reveals that current OC solutions cannot simultaneously achieve all three.

This section is so composed: in Sections VI-A1, VI-A2, and VI-A3 we explain each side of the OC trilemma, and in Section VI-A4 we provide our considerations.

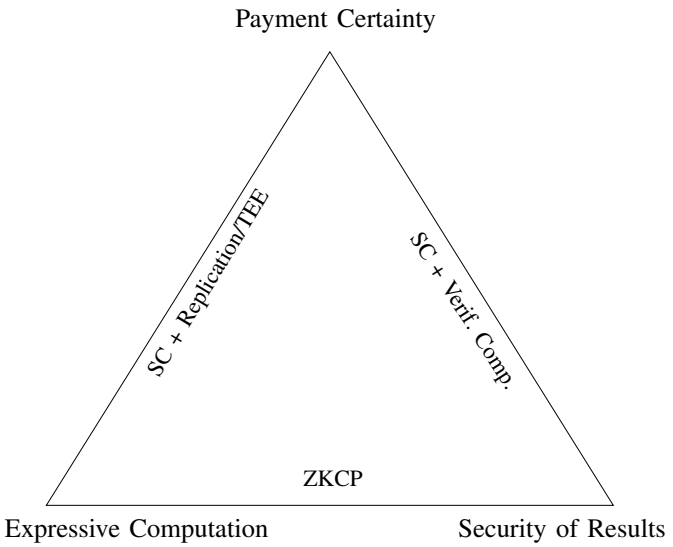


Fig. 3. The OC Trilemma

1) Payment certainty & Expressive Computation: Secure payment can be achieved in conjunction with general-purpose computation using OC protocols that integrate smart contracts and replication-based verification of outsourced computation. As detailed in Section IV-B, this approach allows parties to utilize external servers with a general-purpose architecture for performing computations, while employing a smart contract to facilitate coordination between P and W. In this model, although secure payment with general-purpose computation is feasible, securing the results remains a challenge. As discussed in Section V-F, current replication-based OC protocols lack cryptographically secure methods to ensure result integrity, a concern further elaborated in the context of the Copy Attack in Section IV-B3.

An alternative method to achieve both expressive computation and security of payments is through the combined use of a smart contract and a Trusted Execution Environment (TEE). The smart contract ensures payment security, while TEEs, by enabling general computation and providing execution attestation, are posited to also secure the results. However, the current deployment of TEEs falls short of this security assurance. The uncertainty surrounding the vulnerability of these environments is a significant concern, with a substantial body of literature pointing to numerous potential weaknesses. Additionally, the reliability of the result attestation hinges on the trust placed in the TEE suppliers. Therefore, despite these measures, achieving the security of results remains elusive in this scenario.

2) Secure Results & Expressive Computation: It is feasible to achieve both general-purpose computation and security of results if the security of payments is not a priority. As explored in Section IV-A, Zero-Knowledge Contingent Payment (ZKCP) protocols utilize external servers for computation, leveraging the robustness of zero-knowledge proofs to generate cryptographically valid certificates that verify correct computation. However, these protocols, which operate without a third party and rely on synchronization mechanisms for fair exchange, are vulnerable to griefing attacks (as discussed in Section IV-A3). This vulnerability presents a significant risk: the worker might complete the task without receiving payment. Consequently, ZKCP protocols, in their current form, fail to ensure the security of payments.

Importantly, the use of smart contracts invariably enforces the security of payments. Acting as a neutral third-party coordinator, the smart contract provides both the provider and the worker with certainty regarding payment. This process may involve the use of deposits or collateral to deter bad behavior, ensuring that both parties receive their due compensation. Notably, in scenarios where the provider fails to make a direct payment, the smart contract's stipulations can theoretically result in the worker receiving greater compensation than initially agreed.

3) Secure Results & Payment certainty: Finally, integrating a smart contract with a verifiable computation system presents a promising avenue for simultaneously ensuring the security of both payments and computational results. This combination leverages the strengths of smart contracts in terms of payment security and the robustness of verifiable computation systems in authenticating results. However, a significant limitation arises when it comes to enabling workers to perform general-purpose computations within this model.

This constraint is not a result of any inherent lack of theoretical expressiveness in the zero-knowledge computation schemes that underpin these systems. Rather, it is the practical affordability and scalability of such schemes that pose challenges. The computational and financial overhead associated with verifying complex, general-purpose tasks often outweighs the direct benefits of performing these tasks. As a result, the cost-effectiveness of verifiable computation becomes a critical bottleneck.

4) Considerations:

ZKCP vs Verifiable computation: The unique advantage of Zero-Knowledge Contingent Payment (ZKCP) protocols in facilitating expressive computation lies in their off-chain proof verification process. Unlike on-chain verification, where each transaction or computational proof is validated and recorded on the blockchain, off-chain verification occurs outside of the blockchain network. This distinction in the verification location plays a critical role in the cost-effectiveness and efficiency of ZKCP protocols.

Off-chain verification significantly reduces the computational burden and associated costs compared to on-chain verification methods. In on-chain systems, the process of validating and recording each proof on the blockchain requires substantial computational resources and incurs higher transaction fees, especially on networks with high demand and limited scalability. However, by conducting proof verification off-chain, ZKCP protocols avoid these bottlenecks and resource-intensive processes.

Smart Contracts and Payment Security: The role of smart contracts in ensuring the security of payments in outsourced

computation protocols is highly critical. In fact, by acting as impartial, automated third-party coordinators, smart contracts instill a layer of trust and reliability in the payment process. This aspect is vital in a domain where transactions and contractual fulfillments often occur between parties who may not have prior interactions or established trust.

Smart contracts function by encoding the terms of the agreement into a programmable format that is both transparent and immutable once deployed on the blockchain. This characteristic provides both the provider and the worker with unequivocal assurance regarding the terms of payment. In practice, this might involve mechanisms like escrow services, where the funds are held by the smart contract and only released upon the fulfillment of predefined conditions. Additionally, the use of deposits or collateral is a common practice in these contracts. This serves as a deterrent against malicious or negligent behavior, ensuring a level of accountability and encouraging adherence to the agreed terms.

B. New Directions

As part of our study on outsourced computation with smart contracts, we found some inefficiencies that we want to address. The following are some suggestions on how to improve the current state of the art with respect to outsourced computation with smart contracts.

1) Griefing attack in ZKCP: It is interesting to see that no ZKCP paper takes into account the fact that there can be some cheating regarding the payment, i.e the griefing attack mentioned in Section IV-A3, while other (non synchronous) methods acknowledge this problem (e.g. [54], [55]). As mentioned, even if the atomicity property is respected, the provider can still refuse to pay the worker and the worker will lose time and money for the computation without gaining anything in return.

Moreover, no ZKCP proposal mentions the fact that this problem has already been solved in the literature regarding atomic swaps. One way to solve this thing is by providing deposits or collateral before the actual run of the protocol as seen in the literature on HTLC used for atomic swaps [56] and Lightning Network [52]. We suggest here that ZKCP-based OC protocols include this kind of research:

NQ1: How can griefing attack mitigation strategies, as identified in atomic swap literature, be effectively adapted and applied to outsourced computation protocols within blockchain ecosystems?

NQ2: How does the introduction of deposit requirements for griefing attack mitigation in blockchain-based OC protocols impact the attractiveness and economic viability of these platforms for job providers, potentially affecting their decision to engage in outsourced computation?

2) Syncrhonicity improvements: The first one is related to the synchronization method of OC protocols based on synchronization mechanisms. Currently, the only method presented in literature is the HTLC contract. This method can be improved in two ways.

The first one is by expanding the HTLC to multiple parties, essentially creating the possibility for the delegation of a job from a *group* of providers to a *group* of workers. Multiple workers may do the same job, as in the replication method, or by coordinating with themselves and split the computation. One way to extend the HTLC to a multiparty HTLC (MP-HTLC) can be found in [57].

Another possibility is to leverage the new methods and signing algorithms that are present in Bitcoin, such as the Schnorr signatures. For example the point-time lock contract (PTLC), enabled by adaptor signatures, can be used instead of the HTLC. Here is how it would work.

Assume a key-pair (x, Y), where x is the private key and $Y = xG$ is the public one (G is called generator). To sign a message m using Schnorr Signatures, the signer selects a random nonce k and computes the point $R = kP$. Then, the signer computes the challenge $e = H(R||m)$, where H is a cryptographic hash function, and $||$ denotes concatenation. The signature is the pair (R, s), where $s = k + ex$. Assuming the verifier has message m and public key Y, it can verify it by doing $sG \stackrel{?}{=} R + eY$.

To embed a secret t in the signature by computing $T = tG$ and $e' = H(R + T||m)$. This way s is computed as $s = (k + t) + e'x$, but the signer publishes the *adaptor* signature $(R+T, s-t)$. It is easy to see that the verifier can not verify the signature without t.

It is possible to have a synchronization method using PTLC as HTLC, where the secret is protected by the discrete logarithm problem (roughly, if an attacker knows T it can never recover t) in the former and by the pre-image resistance property of hash functions in the latter.

The introduction of new synchronization mechanisms can potentially reduce the amount of data that P and W need to write on a blockchain. This reduction could lead to improvements in both cost and privacy for ZKCP protocols. Currently, HTLCs are not standard transactions and can be tracked during the execution of the computation. This tracking capability can compromise privacy. Additionally, the larger locking scripts hashed as part of the Pay-to-Script-Hash (P2SH) transaction [58] have to be provided by the W. Larger transactions result in higher fees, increasing the cost of the protocol. Finally, HTLCs are known to be easy to disrupt economically [59]

Point-Time Lock Contracts (PTLCs), on the other hand, incorporate the locking mechanism within the transaction itself, as the secret is embedded in the signature. As a result, a PTLC is indistinguishable from other transactions from the point of view of a blockchain observer, providing an additional layer of privacy for the provider and the worker.

On the other hand since the HTLC is the building block of Bitcoin layer 2s such as Lightning Network, a new approach for ZKCP protocols may be to verify the computation and then exchange the payment through the Lightning Network [60] itself. That would decrease the overhead in the computation since Lightning Network has lower fees than Bitcoin. Then the provider will not have to pay the fees and therefore

can outsource relatively less costly computation and therefore increase the number of contracts full stop.

Another method that can be used to achieve synchronicity between the blockchains is the time-lock puzzle [61] as seen in the work of Barbàra *et al.* [62].

In summary the new research questions we suggest are:

NQ3: How do Point-Time Lock Contracts (PTLCs) with adapter signatures improve the privacy and efficiency of Zero Knowledge Contingent Payment (ZKCP) protocols compared to traditional Hashed Time Lock Contracts (HTLCs)?

NQ4: How does utilizing the Lightning Network for payment exchanges in ZKCP protocols affect the overall efficiency, cost, and scalability of outsourced computation on blockchain platforms?

3) MPC based OC: It is interesting to see a big difference between papers that came out after the deployment of the Ethereum blockchain and those papers that we can call pre-Ethereum.

Pre-Ethereum OC protocols mainly used blockchains to incentivize correct execution. For example the works by Bentov *et al.* [63], [64] and Andrychowicz *et al.* [65], [66] depicts methods to incentivize external server to provide multi-party computation in a correct way. In this case the solution would be verified by the provider of the job and the blockchain will just be used to pay the servers involved in the actual computation.

The papers that came after the deployment of Ethereum see OC protocols as a way to expand the smart contract possibilities. This attitude can be seen in all papers cited before.

We argue that the post-Ethereum approach may gain some insight if combined with the previous literature on the topic[3]. For example, replication methods are based on the fact that each server will provide the *same* exact computation. Beside having some vulnerabilities, such as the copy attack that we mentioned in Section V-B, the replication method it also highly inefficient since is performed equally by all workers.

In fact this computation could be done in a multi-party way so that multiple workers would do this computation and they will participate together in this kind of computation. This would inherently solve the collusion problem that we mentioned before. Another advantage is that it is possible to introduce the concept of "honest threshold" [11] which is highly used in the threshold-cryptosystem literature but that we did not see in the literature related to OC.

NQ5: How can multi-party computation be effectively implemented in OC protocols to mitigate collusion problems and enhance overall computational efficiency?

4) OC Oracles: A critical observation from the review of blockchain-related OC literature is the conspicuous absence of any significant mention of oracles. We posit that oracles can be viewed as a manifestation of outsourced computation, especially those that have conditional payments [68]. Existing literature typically frames oracles as intermediaries in a bet between two parties, Alice and Bob. The oracle performs a computation based on an external event and records the result on the blockchain, determining the winner of the bet. However, this perspective overlooks the fact that the oracle is essentially performing a computation submitted by either Alice or Bob, that becomes the job provider, thereby subjecting it to the principles of OC protocols. Consequently, their protocols can be analyzed within the classification of our OC model.

For instance, Chainlink [5] is a decentralized oracle network of multiple data providers, working mainly in the Ethereum ecosystem. If a data provided submit a wrong piece of data, it is penalized or excluded. Wrongness of data is defined as data that is different from the data provided by the other members of the network. In this sense, then, the interaction between users and Chainlink can be interpreted as an OC protocol employing replication.

Conversely, a Discreet Log Contract [69], [70] is a privacy preserving oracle for the Bitcoin ecosystem. The system works so that in case of a bet between Alice and Bob, each party generates a set of public keys for every possible outcome. These keys are derived in a way that the oracle's signature on the outcome can be combined with these keys to create a valid spending transaction.

Once the event occurs, the oracle broadcasts a signed message representing the outcome. The particular signature is crucial: It allows the unlocking of one of the contingent transactions. From that signature, the party benefiting from the outcome combines the oracle's signature with their own to create a valid signature for the corresponding contingent transaction using the Adaptor Signatures mechanism introduced in Section VI-B2, can be interpreted as an OC protocol employing a synchronization mechanism.

This perspective opens up new avenues for research, including but not limited to the following questions:

NQ6: How can the principles of OC be applied to enhance the efficiency and reliability of oracles in blockchain technology?

NQ7: How can the OC framework be leveraged to develop new oracle protocols for blockchain technology?

5) Scalability enhancements: Similar to the case of oracles (Section VI-B4), also proposals in the literature on blockchain validating bridges (sometimes referred as *Layer 2s* or L2s) do not interpret their design through the lens of outsourced computation.

A validating bridge workflow can be roughly summarized like this, as proposed in the work by McCorry *et al.* [71]. At first, a bridge is deployed on a blockchain, indicating the initialization of the Layer 2 solution. In practice, this is the deployment of a smart contract. Then, the process commences with user transactions being sent to a Sequencer. The Sequencer's role is to order these transactions to ensure their correct sequence before they are batched for processing. Subsequent to ordering, an Executor takes over. The Executor is responsible for committing these transactions to

[3]We are aware of only one proposal that follows this direction, in the work of Li *et al.* [67], but it is still in the pre-print phase

the blockchain. A Challenger monitors an off-chain database storing a copy of transaction data and is tasked with providing "Fraud proof" by verifying the correctness of the Executor's committed transactions.

Using the classification model provided in Section IV, it is possible to interpret a validating bridge as a hybrid kind of OC protocol, where the Sequencer is P, the Executor is W and the Challenger is V, which intervenes only if W does not provide correct computation by providing a proof of fraudulent behavior.

By seeing validating bridges as OCPs, we came up with these new research questions:

NQ8: How can the principles of OC be applied to enhance the efficiency and reliability of validating bridges in blockchain technology?

NQ9: How can the OC framework be leveraged to develop new validating bridges for blockchain technology?

NQ10: Are the financial incentives for V appropriate, or a new design an incentive-compatible protocol is necessary?

VII. Related Works

The field of outsourced computation using smart contracts has been addressed from different perspectives, yet a comprehensive, systematic analysis that covers all aspects of the subject is still lacking.

In the work by Simunic *et al.* [72], the authors examine the application of verifiable computation within the context of blockchain technology. Although their work offers a general overview of verifiable computation, it falls short of providing an in-depth analysis. The study remains superficial and does not delve into different approaches, nor does it offer valuable insights or analysis. Therefore, it does not fully address the complexity and diversity of outsourced computation in smart contracts.

The study conducted by Li *et al.* [73] investigates the concept of computation within smart contracts. However, it focuses on how smart contracts can perform computations internally, which is the opposite of the topic at hand. This approach deviates from the essence of outsourced computation, which is centered around the idea of externalizing computations. Therefore, while Li's work is related, it fails to fill the research gap we aim to address.

Lastly, the paper by Dorsala *et al.* [74] presents a systematization of knowledge (SoK) on Trusted Execution Environment (TEE)-assisted confidential smart contracts. Although it shows the importance of confidentiality in smart contracts, it is largely focused on TEE-only computation and does not consider the concept of 'outsourced' computation. Moreover, it overlooks the potential role of smart contracts in managing the payment for outsourced computations. The study also discusses projects that substitute smart contracts with TEEs for confidentiality, thereby replacing the Ethereum Virtual Machine (EVM) with a TEE, which diverts from our main interest.

VIII. Conclusion

In this paper, we explored the novel issue of blockchain-based outsourced computation protocols within the field of blockchains, drawing parallels to established research domains in database and distributed systems. Our work rigorously formalizes the outsourced computation challenge, revealing a critical reliance on trusted third parties, a finding that challenges prevailing beliefs within the blockchain community. In response, we have developed a model for assessing and creating outsourced computation protocols (OCPs), emphasizing the role of trust assumptions in their design and evaluation.

Our comparative analysis scrutinizes the contemporary hurdles highlighted in existing literature, enhancing the comprehension of the similarities and divergences among various OCP methodologies. More importantly, this paper delves into the implications of these challenges on OCPs, as well as their impact on blockchain ecosystem, especially on scalability.

We introduce a new conceptual trilemma that interweaves the expressiveness of computation, the certainty of payment, and the integrity of results verification in outsourced computation protocols. This trilemma serves as a cornerstone in understanding the complex dynamics at play in OC environments. Furthermore, we connect the dots between outsourced computation and the field of blockchain oracles, proposing that insights from our analysis could offer a comprehensive model for future OCP designs. This perspective not only addresses current gaps but also sets a foundation for innovative approaches in the realm of blockchain-based outsourced computation.

References

[1] W. Zou, D. Lo, P. S. Kochhar, X.-B. D. Le, X. Xia, Y. Feng, Z. Chen, and B. Xu, "Smart contract development: Challenges and opportunities," *IEEE Transactions on Software Engineering*, vol. 47, no. 10, pp. 2084–2106, 2021.

[2] H. Yan, N. Jiang, K. Li, Y. Wang, and G. Yang, "Collusion-free for cloud verification toward the view of game theory," *ACM Trans. Internet Technol.*, vol. 22, no. 2, nov 2021.

[3] L. W., Y. T., and J. X., "Achieving reliable and anti-collusive outsourcing computation and verification based on blockchain in 5g-enabled iot," *Digital Communications and Networks*, vol. 8, no. 5, pp. 644–653, 2022. [Online]. Available: https://doi.org/10.1016/j.dcan.2022.05.012

[4] A. A. Khan, A. A. Laghari, T. R. Gadekallu, Z. A. Shaikh, A. R. Javed, R. Mamoon, V. V. Estrela, and A. Mikhaylov, "A drone-based data management and optimization using metaheuristic algorithms and blockchain smart contracts in a secure fog environment," *Computers and Electrical Engineering*, vol. 102, p. 108234, 2022.

[5] L. Breidenbach, C. Cachin, B. Chan, A. Coventry, S. Ellis, A. Juels, F. Koushanfar, A. Miller, B. Magauran, D. Moroz *et al.*, "Chainlink 2.0: Next steps in the evolution of decentralized oracle networks," *Whitepaper*, 2021. [Online]. Available: https://research.chain.link/whitepaper-v2.pdf

[6] N. Asokan, "Fairness in electronic commerce," Ph.D. dissertation, IBM, 1998.

[7] R. Gennaro, C. Gentry, and B. Parno, "Non-interactive verifiable computing: Outsourcing computation to untrusted workers," in *Advances in Cryptology – CRYPTO 2010*, T. Rabin, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 465–482.

[8] I. Eyal, A. E. Gencer, E. G. Sirer, and R. V. Renesse, "Bitcoin-NG: A scalable blockchain protocol," in *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*. Santa Clara, CA: USENIX Association, Mar. 2016, pp. 45–59. [Online]. Available: https://www.usenix.org/conference/nsdi16/technical-sessions/presentation/eyal

[9] L. Lamport, "A simple approach to specifying concurrent systems," *Commun. ACM*, vol. 32, no. 1, p. 32–45, jan 1989. [Online]. Available: https://doi.org/10.1145/63238.63240

[10] H. Pagnia and F. C. Gartner, "On the impossibility of fair exchange without a trusted third party," Darmstadt University of Technology, Tech. Rep., 1999.

[11] O. Goldreich, S. Micali, and A. Wigderson, "How to play any mental game," in *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, ser. STOC '87. New York, NY, USA: Association for Computing Machinery, 1987, p. 218–229. [Online]. Available: https://doi.org/10.1145/28395.28420

[12] S. Goldwasser, S. Micali, and C. Rackoff, "The knowledge complexity of interactive proof-systems," in *Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing*, ser. STOC '85. New York, NY, USA: Association for Computing Machinery, 1985, p. 291–304. [Online]. Available: https://doi.org/10.1145/22145.22178

[13] O. Goldreich, *The Foundations of Cryptography - Volume 1: Basic Techniques*. Cambridge University Press, 2001. [Online]. Available: http://www.wisdom.weizmann.ac.il/%7Eoded/foc-vol1.html

[14] O. Goldreich, S. Micali, and A. Wigderson, "How to prove all np statements in zero-knowledge and a methodology of cryptographic protocol design (extended abstract)," in *Advances in Cryptology — CRYPTO' 86*, A. M. Odlyzko, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1987, pp. 171–185.

[15] M. J. Fischer, N. A. Lynch, and M. Paterson, "Impossibility of distributed consensus with one faulty process," *J. ACM*, vol. 32, no. 2, pp. 374–382, 1985. [Online]. Available: https://doi.org/10.1145/3149.214121

[16] E. Kokoris-Kogias, P. Jovanovic, L. Gasser, N. Gailly, E. Syta, and B. Ford, "Omniledger: A secure, scale-out, decentralized ledger via sharding," in *2018 IEEE Symposium on Security and Privacy (SP)*, 2018, pp. 583–598. [Online]. Available: https://doi.org/10.1109/SP.2018.000-5

[17] T. Nolan, "Alt chains and atomic transfers," https://bitcointalk.org/index.php?topic=193281#msg2224949, 2013.

[18] P. Todd, "OP_CHECKLOCKTIMEVERIFY," 2014. [Online]. Available: https://github.com/bitcoin/bips/blob/master/bip-0065.mediawiki

[19] G. Maxwell. (2011, November) Zero knowledge contingent payment. Accessed: 2023-10-31. [Online]. Available: https://en.bitcoin.it/w/index.php?title=Zero_Knowledge_Contingent_Payment

[20] W. Banasik, S. Dziembowski, and D. Malinowski, "Efficient zero-knowledge contingent payments in cryptocurrencies without scripts," in *Computer Security – ESORICS 2016*, I. Askoxylakis, S. Ioannidis, S. Katsikas, and C. Meadows, Eds. Cham: Springer International Publishing, 2016, pp. 261–280.

[21] M. Campanelli, R. Gennaro, S. Goldfeder, and L. Nizzardo, "Zero-knowledge contingent payments revisited: Attacks and payments for services," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 229–243. [Online]. Available: https://doi.org/10.1145/3133956.3134060

[22] G. Fuchsbauer, "Wi is not enough: Zero-knowledge contingent (service) payments revisited," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 49–62. [Online]. Available: https://doi.org/10.1145/3319535.3354234

[23] K. Nguyen, M. Ambrona, and M. Abe, "Wi is almost enough: Contingent payment all over again," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 641–656. [Online]. Available: https://doi.org/10.1145/3372297.3417888

[24] Z. Zhou, X. Cao, J. Liu, B. Zhang, and K. Ren, "Zero knowledge contingent payments for trained neural networks," in *Computer Security – ESORICS 2021*, E. Bertino, H. Shulman, and M. Waidner, Eds. Cham: Springer International Publishing, 2021, pp. 628–648.

[25] S. Mazumdar, "Towards faster settlement in htlc-based cross-chain atomic swaps," in *2022 IEEE 4th International Conference on Trust, Privacy and Security in Intelligent Systems, and Applications (TPS-ISA)*, 2022, pp. 295–304. [Online]. Available: https://doi.org/10.1109/TPS-ISA56441.2022.00043

[26] E. Heilman, S. Lipmann, and S. Goldberg, "The arwen trading protocols," in *Financial Cryptography and Data Security - 24th International Conference, FC 2020, Kota Kinabalu, Malaysia, February 10-14, 2020 Revised Selected Papers*, ser. Lecture Notes in Computer Science, J. Bonneau and N. Heninger, Eds., vol. 12059. Springer, 2020, pp. 156–173. [Online]. Available: https://doi.org/10.1007/978-3-030-51280-4_10

[27] C. Dong, Y. Wang, A. Aldweesh, P. McCorry, and A. van Moorsel, "Betrayal, distrust, and rationality: Smart counter-collusion contracts for verifiable cloud computing," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 211–227.

[28] T. Li, Y. Fang, Y. Lu, J. Yang, Z. Jian, Z. Wan, and Y. Li, "Smartvm: A smart contract virtual machine for fast on-chain dnn computations," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 12, pp. 4100–4116, 2022. [Online]. Available: https://doi.org/10.1109/TPDS.2022.3177405

[29] A. Küpçü and R. Safavi-Naini, "Smart contracts for incentivized outsourcing of computation," in *Data Privacy Management, Cryptocurrencies and Blockchain Technology*, J. Garcia-Alfaro, J. L. Muñoz-Tapia, G. Navarro-Arribas, and M. Soriano, Eds. Cham: Springer International Publishing, 2022, pp. 245–261.

[30] J. Liu and L. F. Zhang, "Privacy-preserving and publicly verifiable matrix multiplication," *IEEE Transactions on Services Computing*, vol. 16, no. 3, pp. 2059–2071, 2023.

[31] A. R. Sai, J. Buckley, B. Fitzgerald, and A. L. Gear, "Taxonomy of Centralization in Public Blockchain Systems: A Systematic Literature Review," *arXiv:2009.12542 [cs]*, Sep. 2020, arXiv: 2009.12542. [Online]. Available: https://doi.org/10.1016/j.ipm.2021.102584

[32] C. Troncoso, M. Isaakidis, G. Danezis, and H. Halpin, "Systematizing Decentralization and Privacy: Lessons from 15 Years of Research and Deployments," *Proceedings on Privacy Enhancing Technologies*, vol. 2017, no. 4, pp. 404–426, Oct. 2017. [Online]. Available: https://www.sciendo.com/article/10.1515/popets-2017-0056

[33] S. P. Gochhayat, S. Shetty, R. Mukkamala, P. Foytik, G. A. Kamhoua, and L. Njilla, "Measuring Decentrality in Blockchain Based Systems," *IEEE Access*, vol. 8, pp. 178 372–178 390, 2020. [Online]. Available: https://ieeexplore.ieee.org/document/9205256/

[34] I. Anati, S. Gueron, S. P. Johnson, and V. R. Scarlata, "Innovative technology for cpu based attestation and sealing," Intel Corporation, Tech. Rep., 2013.

[35] F. McKeen, I. Alexandrovich, A. Berenzon, C. V. Rozas, H. Shafi, V. Shanbhogue, and U. R. Savagaonkar, "Innovative instructions and software model for isolated execution," in *Proceedings of the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy*, ser. HASP '13. New York, NY, USA: Association for Computing Machinery, 2013. [Online]. Available: https://doi.org/10.1145/2487726.2488368

[36] A. Baumann, M. Peinado, and G. Hunt, "Shielding applications from an untrusted cloud with haven," in *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*. Broomfield, CO: USENIX Association, Oct. 2014, pp. 267–283. [Online]. Available: https://www.usenix.org/conference/osdi14/technical-sessions/presentation/baumann

[37] S. Shinde, D. Le Tien, S. Tople, and P. Saxena, "Panoply: Low-tcb linux applications with sgx enclaves," in *Network and Distributed System Security (NDSS17)*, 2017. [Online]. Available: https://www.ndss-symposium.org/wp-content/uploads/2017/09/ndss2017_07-5_Shinde_paper.pdf

[38] M. Król and I. Psaras, "SPOC: secure payments for outsourced computations," *CoRR*, vol. abs/1807.06462, 2018. [Online]. Available: http://arxiv.org/abs/1807.06462

[39] H. Dang, D. Le Tien, and E. Chang, "Towards a marketplace for secure outsourced computations," in *Computer Security – ESORICS 2019*, K. Sako, S. Schneider, and P. Y. A. Ryan, Eds. Cham: Springer International Publishing, 2019, pp. 790–808.

[40] S. Avizheh, M. Nabi, R. Safavi-Naini, and M. Venkateswarlu K., "Verifiable computation using smart contracts," in *Proceedings of the 2019 ACM SIGSAC Conference on Cloud Computing Security Workshop*, ser. CCSW'19. New York, NY, USA: Association for Computing Machinery, 2019, p. 17–28.

[41] M. R. Dorsala, V. Sastry, and S. Chapram, "Fair payments for verifiable cloud services using smart contracts," *Computers & Security*, vol. 90, p. 101712, 2020.

[42] W. Susilo, F. Guo, Z. Zhao, Y. Jiang, and C. Ge, "Secure replication-based outsourced computation using smart contracts," *IEEE Transactions on Services Computing*, vol. 16, no. 5, pp. 3711–3722, 2023.

[43] L. Babai, L. Fortnow, L. A. Levin, and M. Szegedy, "Checking computations in polylogarithmic time," in *Proceedings of the Twenty-Third Annual ACM Symposium on Theory of Computing*, ser. STOC '91. New York, NY, USA: Association for Computing Machinery, 1991, p. 21–32. [Online]. Available: https://doi.org/10.1145/103418.103428

[44] S. Goldwasser, Y. T. Kalai, and G. N. Rothblum, "Delegating computation: Interactive proofs for muggles," in *Proceedings of the Fortieth Annual ACM Symposium on Theory of Computing*, ser. STOC '08. New York, NY, USA: Association for Computing Machinery, 2008, p. 113–122. [Online]. Available: https://doi.org/10.1145/1374376.1374396

[45] S. Micali, "Computationally sound proofs," *SIAM J. Comput.*, vol. 30, no. 4, p. 1253–1298, oct 2000. [Online]. Available: https://doi.org/10.1137/S0097539795284959

[46] B. Parno, M. Raykova, and V. Vaikuntanathan, "How to delegate and verify in public: Verifiable computation from attribute-based encryption," in *Theory of Cryptography*, R. Cramer, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 422–439.

[47] M. R. Dorsala, V. N. Sastry, and S. chapram, "Fair protocols for verifiable computations using bitcoin and ethereum," in *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*, 2018, pp. 786–793. [Online]. Available: https://doi.org/10.1109/CLOUD.2018.00107

[48] J. Teutsch and C. Reitwiessner, *A Scalable Verification Solution for Blockchains*, 2017, pp. 377–424.

[49] M. Walfish and A. J. Blumberg, "Verifying computations without reexecuting them," *Commun. ACM*, vol. 58, no. 2, p. 74–84, jan 2015. [Online]. Available: https://doi.org/10.1145/2641562

[50] G. Yang, C. H. Tan, Q. Huang, and D. S. Wong, "Probabilistic public key encryption with equality test," in *Topics in Cryptology - CT-RSA 2010*, J. Pieprzyk, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 119–131.

[51] Y. Guan, H. Zheng, J. Shao, R. Lu, and G. Wei, "Fair outsourcing polynomial computation based on the blockchain," *IEEE Transactions on Services Computing*, vol. 15, no. 5, pp. 2795–2808, 2022.

[52] S. Mazumdar, P. Banerjee, and S. Ruj, "Time is money: Countering griefing attack in lightning network," in *2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, 2020, pp. 1036–1043. [Online]. Available: https://doi.org/10.1109/TrustCom50675.2020.00138

[53] J. Lind, O. Naor, I. Eyal, F. Kelbert, E. G. Sirer, and P. Pietzuch, "Teechain: A secure payment network with asynchronous blockchain access," in *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, ser. SOSP '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 63–79. [Online]. Available: https://doi.org/10.1145/3341301.3359627

[54] R. Kumaresan and I. Bentov, "Amortizing secure computation with penalties," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '16. New York, NY, USA: Association for Computing Machinery, 2016, pp. 418–429. [Online]. Available: https://doi.org/10.1145/2976749.2978424

[55] R. Kumaresan, V. Vaikuntanathan, and P. N. Vasudevan, "Improvements to secure computation with penalties," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 406–417. [Online]. Available: https://doi.org/10.1145/2976749.2978421

[56] T. Nadahalli, M. Khabbazian, and R. Wattenhofer, "Grief-free atomic swaps," in *2022 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, 2022, pp. 1–9. [Online]. Available: https://doi.org/10.1109/ICBC54727.2022.9805490

[57] F. Barbàra and C. Schifanella, "MP-HTLC: Enabling blockchain interoperability through a multiparty implementation of the htlc," *Concurrency and Computation: Practice and Experience*, 2022. [Online]. Available: https://doi.org/10.1002/cpe.7656

[58] G. Andresen, "Address format for pay-to-script-hash," 2011. [Online]. Available: https://github.com/bitcoin/bips/blob/master/bip-0013.mediawiki

[59] I. Tsabary, M. Yechieli, A. Manuskin, and I. Eyal, "MAD-HTLC: because HTLC is crazy-cheap to attack," pp. 1230–1248, 2021. [Online]. Available: https://doi.org/10.1109/SP40001.2021.00080

[60] J. Poon and T. Dryja, "The bitcoin lightning network: Scalable off-chain instant payments," 2016.

[61] R. L. Rivest, A. Shamir, and D. A. Wagner, "Time-lock puzzles and timed-release Crypto," Tech. Rep., 1996.

[62] F. Barbàra, N. Murru, and C. Schifanella, "Towards a broadcast time-lock based token exchange protocol," in *Euro-Par 2021: Parallel Processing Workshops - Euro-Par 2021 International Workshops*, ser. Lecture Notes in Computer Science, 2021. [Online]. Available: https://doi.org/10.1007/978-3-031-06156-1_20

[63] I. Bentov and R. Kumaresan, "How to use bitcoin to incentivize correct computations," in *Advances in Cryptology – CRYPTO 2014*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 421–439. [Online]. Available: https://doi.org/10.1007/978-3-662-44381-1_24

[64] R. Kumaresan and I. Bentov, "How to use bitcoin to incentivize correct computations," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '14. New York, NY, USA: Association for Computing Machinery, 2014, p. 30–41. [Online]. Available: https://doi.org/10.1145/2660267.2660380

[65] M. Andrychowicz, S. Dziembowski, D. Malinowski, and Ł. Mazurek, "Fair two-party computations via bitcoin deposits," in *Financial Cryptography and Data Security*, R. Böhme, M. Brenner, T. Moore, and M. Smith, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 105–121.

[66] M. Andrychowicz, S. Dziembowski, D. Malinowski, and L. Mazurek, "Secure multiparty computations on bitcoin," in *2014 IEEE Symposium on Security and Privacy*, 2014, pp. 443–458. [Online]. Available: https://doi.org/10.1109/SP.2014.35

[67] Y. Li, K. Soska, Z. Huang, S. Bellemare, M. Quintyne-Collins, L. Wang, X. Liu, D. Song, and A. Miller, "Ratel: Mpc-extensions for smart contracts," Cryptology ePrint Archive, Paper 2023/1909, 2023, https://eprint.iacr.org/2023/1909. [Online]. Available: https://eprint.iacr.org/2023/1909

[68] V. Madathil, S. A. K. Thyagarajan, D. Vasilopoulos, L. Fournier, G. Malavolta, and P. Moreno-Sanchez, "Cryptographic oracle-based conditional payments," in *30th Annual Network and Distributed System Security Symposium, NDSS 2023, San Diego, California, USA, February 27 - March 3, 2023*. The Internet Society, 2023. [Online]. Available: https://www.ndss-symposium.org/ndss-paper/cryptographic-oracle-based-conditional-payments/

[69] T. Dryja, "Discreet log contracts," MIT Digital Currency Initiative, Tech. Rep., 2019.

[70] T. L. Guilly, N. Kohen, and I. Kuwahara, "Bitcoin oracle contracts: Discreet log contracts in practice," in *IEEE International Conference on Blockchain and Cryptocurrency, ICBC 2022, Shanghai, China, May 2-5, 2022*. IEEE, 2022, pp. 1–8. [Online]. Available: https://doi.org/10.1109/ICBC54727.2022.9805512

[71] P. McCorry, C. Buckland, B. Yee, and D. Song, "Sok: Validating bridges as a scaling solution for blockchains," *IACR Cryptol. ePrint Arch.*, p. 1589, 2021. [Online]. Available: https://eprint.iacr.org/2021/1589

[72] S. Simunic, D. Bernaca, and K. Lenac, "Verifiable computing applications in blockchain," *IEEE Access*, vol. 9, pp. 156 729–156 745, 2021.

[73] R. Li, Q. Wang, Q. Wang, D. Galindo, and M. Ryan, "Sok: Tee-assisted confidential smart contract," *Proc. Priv. Enhancing Technol.*, vol. 2022, no. 3, pp. 711–731, 2022. [Online]. Available: https://doi.org/10.56553/popets-2022-0093

[74] M. R. Dorsala, V. Sastry, and S. Chapram, "Blockchain-based solutions for cloud computing: A survey," *Journal of Network and Computer Applications*, vol. 196, p. 103246, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1084804521002447