Prometheus: Scalable High-load Computing & Machine Learning on top of Bitcoin

Maxim Orlovsky*[1,2] and Sabina Sachtachtinskagia†[1,3]

[1]Pandora Core AG, Switzerland
[2]BICA Labs, Switzerland
[3]Athens University of Economics and Business, Greece

Version 0.1.0, working draft

Abstract

This work defines Prometheus, a protocol for running high-load & machine learning computing inside a trustless decentralized environment in Byzantine-tolerant way, utilizing bitcoin blockchain and script functionality, layer 2 state channels, zero knowledge computation integrity arguments, economic incentives and p2p networking.

This work covers questions of Prometheus design rationale, architecture, formal protocol definition, transaction-based model and specifies protocol operations with demonstrable Nash equilibrium; the latter part is based on our previous work on incentive-based interactive settlement scheme for computation integrity proofs [22].

This work will be followed by the papers on (1) specific probabilistically checkable proof schemes for high-load & machine learning computing, addressing the problems of non-determinism in artificial neural networks, data privacy and federated learning and (2) underlying P2P communication protocol for global state replication and state channel management.

1 Background & Rationale

The modern success of cloud computing is related to the economy of scale phenomenon, enhanced by the exponentially increasing amounts of produced data. However, it has created systematic risks and fragility, namely vanishing privacy, possibilities of totalitarian control, surveillance, single points of failure and censorship. It has put at risk the further development of machine learning technologies, which have started to be monopolised together with the access to big data. The #FreeAI Manifesto [3] had declared an initiative to oppose this dramatic trend, and this work aims at the development of the technology stack able to address the issue.

*orlovsky@pandoracore.com
†sabina.aueb@gmail.com

The problem of censorship-resistant computing consists of many parts, starting from finding and buying information (big data, machine learning models) in a censorship-resistant way, owning it with proper privacy and plausible deniability – and up to computations using big data in anonymous and censorship-resistant and (desirably) private manner. Prometheus protocol addresses the latter issue, since the former two matters (finding and owning information) can be solved with many other existing initiatives, such as dark/anonymous markets, cryptography technologies, zero-knowledge, distributed storage and data transfer (Torrent, IPFS) etc [8, 10, 15, 11, 9].

The actual technical problem we need to tackle in order to reach our goal is the proofs for computational integrity (correctness of the actual computations) for the computing performed by some non-trusted (anonymous) third parties – without the need to repeat the whole volume of the actual computing. This issue is usually addressed by the use of probabilistically checkable proofs (PCP) – however, they leave untouched the question of coupling the payments for the consumed computational resources with the actual proof. This question was resolved in our previous work [22].

Ability to run PCP schemes without trusted checking party provides a way to codify the actual protocol in terms of smart contracts, so it can be used in a very compact form with existing economic blockchain layer. Our approach to protocol design aims at the smallest possible size of the shared system state so it can be implemented with any non-Turing complete smart contract system with storing nothing else than the account balances (or UTXOs) and their unlocking conditions for the involved parties. For example, the protocol can be implemented with the existing Bitcoin script functionality with only two on-chain transactions per whole protocol computing cycle. In such system the PCP proofs are used at the second layer on top of slow blockchain, reducing the footprint, price and increasing scalability for the real-world business.

2 Generic Protocol

2.1 Architecture

A core of the protocol is based on the following main algorithms and cryptographic schemes:

IBISS-CIP Incentive-based interactive settlement scheme for computation integrity arguments: a game theory model for providing provably-safe economic incentives [22]

Bitcoin including Bitcoin as a mean of payment, bitcoin blockchain as a safety and immutability layer, Bitcoin Script and HTLC smart contracts for the settlement algorithm logic.

PCP probabilistically checkable proofs [7, 13] as non-interactive computation integrity verification technology

P2P Networking for building decentralised network of nodes

The protocol is designed in a modular way, with four main components:

Settlement algorithm based on a game theory model with provable Nash equilibrium described in Section 3 – an implementation of IBISS-CIP scheme [22]

Transaction model defining set of interdependent off-chain and on-chain transactions with corresponding HLTV/CLTV/multisig smart contracts, described in Section 4

Networking protocol for P2P node communications and information exchange

HLML-PCP schemes PCP extension schemes for high-load computing and machine learning-specific tasks (HLML)

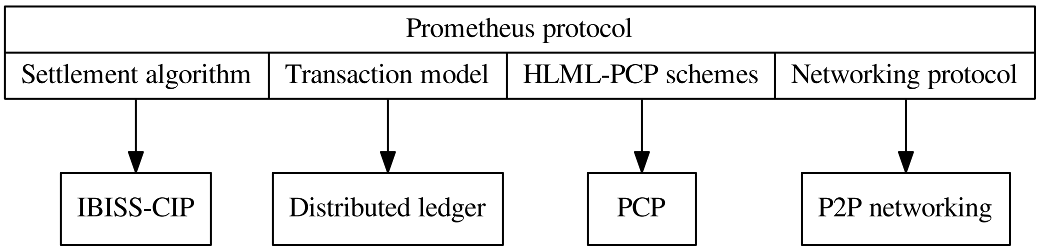


Figure 1: Architecture of the Prometheus protocol

Particularities of specific PCP gadgets for high-load & machine learning computing, as well as P2P protocol are not covered by these document.

2.2 Scopes and Actors

The protocol operates within two different scopes: **global scope** Γ and **channel scope** Λ.

The protocol design includes the following types of participants (actors):

Client node C: party prepared to pay for the actual computing and provide necessary data for it.

Worker node W: party prepared to perform actual computing for some reward.

Verifier node V: party prepared to verify the actual computing using HLML-PCP.

Arbiter nodes $\mathbb{A}$ (global) and their subset $\mathcal{A}$ (for a given computing contract): a special sets of nodes committed to do the arbitration work

Common fund F: used as a wild card destination for penalty payments. This can be a deterministic unspendable contract, or a special contact for funding protocol development etc. The particular value form of common fund is defined by each of Prometheus contracts (see below).

Clients, Workers and Verifiers (as well as the common funds) operate only within *channel scope*, while Arbiters are defined within the *global scope* (denoted by $\mathbb{A} \mid \Gamma$) and are able to participate in *channel scope* (denoted by $\mathcal{A} \mid \Lambda$).

2.3 Layers, Contracts and States

The protocol operates using different layers of abstraction, each of which contains its own set of contracts:

1. **Global consensus state** $\mathsf{G} \mid \Gamma$ based on blockchain-stored state Γ^{Ω} and off-chain data Γ^{Θ}, committed to the blockchain with hashes and replicated by all parties participating the protocol:

   Prometheus contracts $\{\mathsf{P}_i\} \mid \Gamma$: specially-formed contracts encoding pre-defined sets of global parameters. These are *commitment-based agreement*, i.e. agreements that can be referenced by any independent party as a commitment to follow/accept these pre-defined values. Basically, it is a tuple of meta-parameters:
   $\mathsf{P} = \langle w, v, n, a, F, \alpha, \beta, \gamma, \delta, \varepsilon, \mu, \sigma, t_1, t_2, t_3, t_4 \rangle$
   For the meaning of the constants see Table 1.

   Arbitration offers $\{\mathsf{O}_i\} \mid \Gamma$: specially-formed contracts from parties that want to act as Arbiters in any future computation contract. These contracts are committed to particular Prometheus contract P_i and must contain proofs of the locked arbiter's stake of size a (defined by P_i): $\mathsf{O} = \langle \mathsf{P}_i, a_p \rangle$. Parties that have made these commitments constitutes global set $\mathbb{A}$.

   Computational contracts $\{\mathsf{C}_i\} \mid (\Gamma \vee \Lambda)$: *economically-binding agreements* committed to some particular contract P_i between parties C, W, V containing proofs of their locked stakes w_p, v_p (corresponding to the amounts defined by P_i), computational reward c and computation-related `payload`: $\mathsf{C} = \langle \mathsf{P}_i, C, W, V, c, w_p, v_p, \mathtt{payload} \rangle$. Computational contract locks deposits and includes all possible settlement schemes (see Table 2).

   Arbitration contracts $\{\mathsf{A}_i\} \mid (\Gamma \vee \Lambda)$: *economically-binding agreements* committed to some particular contract C_i by Arbiters (denoted as $\mathcal{A}$) that were randomly selected from $\mathbb{A}$ to participate in Arbitration stage of the computing contract. The contract contains proofs of the arbiter's stake commitments (a_p): $\mathsf{A} = \langle \mathsf{C}_i, \mathcal{A}, \{a_{p_i} \mid \forall A_i \in \mathcal{A}\} \rangle$

2. **Multiparty state channel** $\mathsf{M}(\mathsf{C}_i) \mid \Lambda$ originating from each computational contract C_i: $\mathsf{M}(\mathsf{C}_i) = \langle \mathsf{C}_\mathsf{i}, \Upsilon, \pi, \mathtt{Encrypt}(\Pi, h_p) \rangle$. The channel operates the following state data:

   Partially-signed transaction tree Υ, which defines:

   - Current **operational state** of the channel (see below for the details);
   - **Hash-lock value** h provided by the Worker and used in hash-locked transactions within Υ to bind the use of reward c by the Worker with the ability of the Client to read the results of computation, encrypted with this hash *preimage* h_p (see below);

- **Verifier's verdict** $r = \{\emptyset, 0, 1\}$ on Worker's proofs of computational integrity. Verdict equals to $\emptyset$ before the completion of the Verification phase (see below);
- **Set of Arbiters** $\mathcal{A}$ (equals to $\emptyset$ befor the start of arbitration phase);
- **State of arbitration contract** A_i and corresponding signed but unpublished transactions
- **Arbitration decision** $d = \{\emptyset, 0, 1\}$: *economically-binding decision* provided by $\mathcal{A}^+$ (subset of $\mathcal{A}$ which have signed to the decision). Equals to $\emptyset$ before the completion of arbitration phase – or in case if no definite decision was reached.

Non-interactive probabilistic proofs π of computational integrity provided by the Worker at the end of computation phase.

Computation results Π encrypted with *preimage* h_p by the Worker.

3. **Internal state** of the actors (C, W, V, A_i): each party acts as a part of channel state machine (i.e. it is replicated & semi-synchroneous state machine). Thus, the internal state of the party is tightly related to the state of the channel (see below). When required, the current state of the party is denoted with precondition notation: for instance, $W \mid \Lambda^{\mathsf{C}}$ represents Worker node under computing state, while $\Lambda^{\mathsf{C}} \mid W$ represents channel in computing state with Worker node signed confirmation (agreement) on the state published into the channel.

For the complete list of state parameters see Table 1.

Multiparty state channel operates as a deterministic finite state machine with the following possible states (see also Fig. 2):

Setup Λ^{S}, performed by all parties interactively

Computation Λ^{C}, performed by Worker

Verification Λ^{V}, performed by Verifier

Arbitration Λ^{A} (optional phase), performed by special group of Arbiters

Closed $\Lambda^{\times}$ with one of the settlement schemes published to the global state G (for settlement schemes see next section and Table 2). The used settlement scheme is denoted in the subscript of the channel state, i.e. $\Lambda^{\times}_{3}$.

State transitions are denoted by arrows with state transition conditions listed above the arrow.

The normal state transition path is defined by $\Lambda^{\mathsf{S}} \xrightarrow{\mathsf{C}} \Lambda^{\mathsf{C}} \xrightarrow{\pi} \Lambda^{\mathsf{V}} \xrightarrow{r=1} \Lambda^{\times}_{3}$.

3 Settlement Algorithm

Prometheus *settlement algorithm* provides economic incentives and settlement schemes with a provable Nash equilibrium for non-Byzantine behaviour of all of the rational participants. It is developed on top of incentive-based interactive settlement scheme for computation integrity proofs [22].

The algorithm runs in five phases (Fig. 2), each of which corresponds to particular state of the channel (Fig. 2)

Table 1: Summary of state parameters

Parameter	Description	Bounding-conditions	Defined by
	1. Actors, signatures and stakes		
W, V, C	Worker, Verifier and Client as actors identified by their signatures	$W \neq V \neq C$	C
w, v	Stakes deposited by Worker and Verifier	$w \gg z$, $v \gg y$, $w \leq v$	P
c	Amount the Client is willing to pay for the computation	$c \gg z \gg y$	C
z	Cost incurred by the Worker for performing the computation	$z \ll c$	W
y	Cost incurred by the Verifier for performing the verification	$y \ll z \ll c$	V
F	*Common fund* public key/address which is controlled by external body (can be charity, donations to development, deterministic proof of burn address or anything else)		P
$\mathbb{A}$	Global set of pre-registered Arbiters	$\mathbb{A} \neq \emptyset$	O
$\mathcal{A} = \{A_i\}$	Set of randomly-selected Arbiters as actors identified by their signatures	$\mathcal{A} \subset \mathbb{A} \mid \mathcal{A} \mid \geq 3$	M
$\mathcal{A}^+$	Subset of Arbiters that have made a coherent decision	$\mathcal{A}^+ \subset \mathcal{A}$	M
$\mathcal{A}^-$	Subset of Arbiters that didn't participate or agreed with a coherent decision	$\mathcal{A}^- = \mathcal{A} \setminus \mathcal{A}^+$ $\mathcal{A} = \mathcal{A}^+ \cup \mathcal{A}^-$ $\mid \mathcal{A}^+ \mid > \mu \mid \mathcal{A}^- \mid$	M
n	Size of $\mathcal{A}$ (amount of Arbiters participating arbitration)	$3 \leq n \ll \mid \mathbb{A} \mid$	P
a	Standard stake of Arbiter locked in blockchain-based contract	$a \gg v$	P
	2. Cryptographic functions and arguments		
h, h_p	Hashed value h and its *preimage* h_p initially selected and known only to the Worker. h_p is used to encrypt the content of the data file.	$h = \mathtt{HASH}(h_p)$	W
t_1, t_2, t_3, t_4	*Timelock* conditions ordered in such a way that smaller-indexed timelock precedes in time timelocks with larger indexes	$t_i \prec t_{i+n}$	P
T^k_n	k-of-n threshold multisignature	$n = \mid \mathcal{A} \mid, k = \lceil \mu n \rceil$	M
d	*Decision* taken by Arbiters, such as $\mid d = 1$ is a pre-condition satisfying that the *voting majority* of Arbiters $\mathcal{A}^+$ found Worker W of being faulty.	$d \in \{0, 1\}$	M
	3. Meta-parameters		
α	Fraction of stake taken from Worker's deposit w in case of its timeout	$0 < \alpha \leq 1$	P
β	Fraction of stake taken from Verifier's deposit v in case of its timeout	$0 < \beta \leq 1$	P
γ	Fraction of Worker's stake w that is awarded to the Client as a compensation payment in case the Arbitation was not being able to reach a final decision	$0 < \gamma < \varepsilon$	P
δ	Fraction of stake w taken from Worker W if it was found faulty by the Verifier and didn't appeal	$\delta \leq w - (1 - \sigma)c$	P
ε	Fraction of stakes that all participants $(W, V, \mathcal{A})$ are penalised in case of failed Arbitration (i.e. when $\mid \mathcal{A}^+ \mid < \mu \mid \mathcal{A} \mid$)	$\gamma < \varepsilon \leq 1$	P
μ	*Arbiter voting majority*: threshold for Arbiters decision making	$\mu > 0.5$	P
σ	Fraction of computation used by Worker in its *computation argument* – and subsequently verified by Verifier. It also defines the proportion of Client's payment c that goes to Worker W in case of successful computing (correspondingly, Verifier V will receive reward of $1 - \sigma c$)	$0 \leq \sigma \leq 1$	P

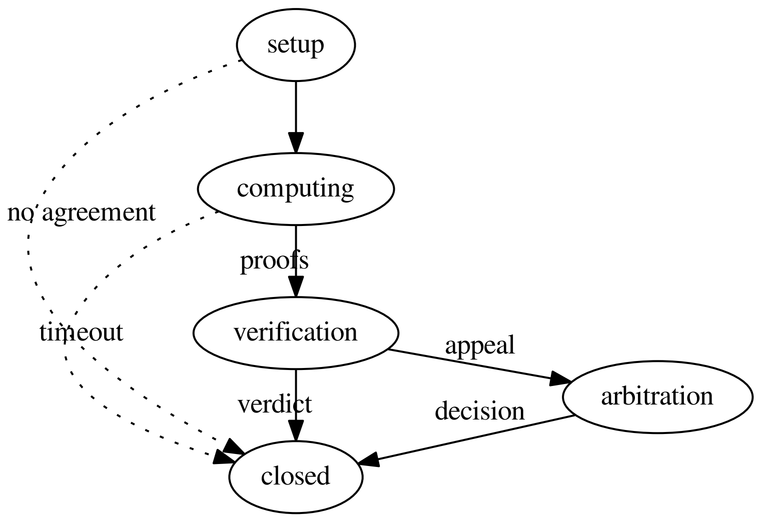


Figure 2: Possible channel states and transitions

3.1 Algorithm phases

3.1.1 Setup – Λ^{S}

During the Setup phase, the Client node splits the actual computing task into *batches* suitable for parallel processing in a distributed P2P network.

For each batch, the client selects Worker and Verifier nodes on some open market or by auction (the actual selection is not part of this protocol). They agree upon the contract conditions (by committing to particular Prometheus contract P_i) and computation reward c in the process of off-chain P2P communications.

To secure the actual contract Client, Worker and Verifier put c, w, v under the *computation contract* as described in section 2.3 and publish it to the blockchain.

3.1.2 Computation – Λ^{C}

Worker node downloads all necessary data according to the contract and runs actual computation. Upon their completement the Worker encrypts the results with h_p value, prepares PCP proof π with one of the HLML-PCP schemes and updates the state of the channel with these data. The proof is constructed for some pre-defined portion of the data σ.

If the phase is not completed within the pre-defined time, the Client or the Verifier has the right to terminate the contract and penalise the Worker (see settlement scheme #1 in Fig. 3 and Table 2)

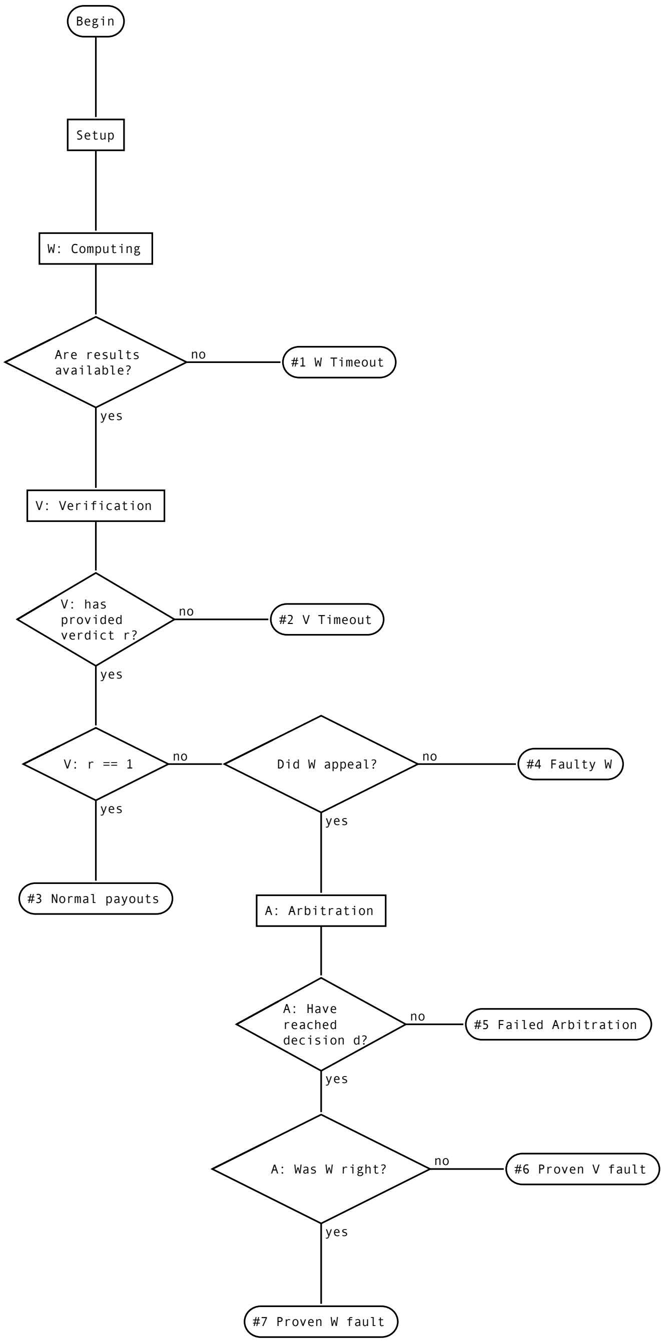


Figure 3: Prometheus settlement algorithm flowchart

Rectangles corresponds to contract phases and channel states; numbered exists (#1, #2 etc) corresponds to the settlement schemes from the Table 2

3.1.3 Verification – Λ^{V}

Verifier node downloads source data with proofs π, checks them and reports *verdict* r into the channel. If the Worker found faulty, it has the right to either *appeal* and start Arbitration procedure – or *admit the fault* by signing another transaction corresponding to settlement scheme #4 (see Table 2). In this case Worker will have a smaller penalty and we will prevent unnecessary computation load.

If the phase is not completed within the pre-defined time, the Worker has the right to terminate the contract and penalise Verifier (see settlement scheme #2 in Fig. 3 and Table 2)

3.1.4 Arbitration – Λ^{A}

Arbiters $\mathcal{A}$ participating the Arbitration phase of the particular computing contract must be chosen from the global arbiters pool/set $\mathbb{A}$ (established well before the initiation of the computing contract) with some deterministic randomness procedure. This is required in order to prevent actors of spamming the Arbiters pool with their agents once the contract has reached Arbitration stage.

Before become part of the $\mathcal{A}$ each of the selected Arbiters must confirm their participation via off-chain messaging. Jointly Arbiters constituting $\mathcal{A}$ sign their stakes into the computing contract with economically binding contract A as described in section 2.3 above.

Arbiters reach decision in process of independent checks of π proofs and P2P communications. To reach decision at least μ part of all Arbiters must come to the same coherent conclusion, where μ is a so-called *voting majority* – a parameter defined by Prometheus contract P_i.

If Arbiters are unable to reach a *coherent decision* ($d = \emptyset$) with μ% of votes, the computing contract defaults to payment scheme #5 (Table 2) where all of the contract participants (except of the Client) are penalised and the Client is compensated with some slice of their stake.

If the coherent decision has being made, the contract completes with either #6 or #7 payment schemes (Table 2) depending on the decision. In both schemes, the parties found faulty are penalised; and it the Worker computation results are found correct, the Worker receives its payment and the Client is provided with the results.

3.1.5 Payouts and Channel Closing – $\Lambda^{\times}$

Depending on the completion of the phases 2, 3 and 4 there might be different ways of the final settlement, numbered on the Fig. 3 from #1 to #7 and described below in the Table 2.

The channel is closed by publishing the singed branch of the transaction tree Υ from the channel scope into the global scope: $\Lambda^{\times} \xrightarrow{\Upsilon} \Gamma$.

In cases when the computations by the Worker were proven to be correct (schemes #2, #3 and #7) in order to take its reward the Worker must publish to the blockchain a spending transaction containing *preimage* h_p used for encrypting the resulting data payload – thus allowing Client to read its value and decrypt the data.

Table 2: Settlement schemes

ID	Scheme	Payout schedule	Precondition | postcondition[1],[2]	Deposit redistribution[1]
#1	Worker timeout	Penalise Worker with a fraction of stake	$\ldots\Lambda^{\mathsf{C}} \xrightarrow{\pi=\emptyset}$ $\mid\ t_3 \wedge (C \vee V)$	$W : -\alpha w \to F$
#2	Verifier timeout	Pay to Worker, penalise Verifier with a fraction of stake to Client	$\ldots\Lambda^{\mathsf{V}} \xrightarrow{r=\emptyset}$ $\mid\ t_1 \wedge W \wedge h_p$	$V : -\beta v \to C$, $C : -\sigma c \to W$
#3	Normal payouts	Pay to Worker and Verifier	$\ldots\Lambda^{\mathsf{V}} \xrightarrow{r=1}$ $\mid\ W \wedge h_p \wedge V$	$C : -\sigma c \to W$, $C : -(1-\sigma)c \to V$
#4	Faulty Worker, no appeal	Pay to Verifier from Worker stake, penalise Worker with a fraction of stake	$\ldots\Lambda^{\mathsf{V}} \xrightarrow{r=0}$ $\mid\ t_2 \wedge V \wedge W$	$W : -(1-\sigma)c \to V$, $W : -\delta w - (1-\sigma)c \to F$
#5	Failed Arbitration (no coherent decision)	Penalise all participants, compensate Client	$\ldots\Lambda^{\mathsf{V}} \xrightarrow{r=0} \Lambda^{\mathsf{A}} \xrightarrow{d=\emptyset}$ $\mid\ t_4 \wedge C$	$W : -\gamma w \to C$, $W, V, \mathcal{A} :$ $-\varepsilon(w + v + \sum a) \to F$
#6	Arbitration proved Worker fault	Penalise Worker with full stake transferred to Arbiters, penalise faulty Arbiters	$\ldots\Lambda^{\mathsf{V}} \xrightarrow{r=0} \Lambda^{\mathsf{A}} \xrightarrow{d=1}$ $\mid\ T^k_n(\mathcal{A}^+)$	$W : -w \to \mathcal{A}^+$, $\forall A_i \in \mathcal{A}^- : -a_i \to F$
#7	Arbitration proved Verifier fault	Pay to Worker, penalise Verifier with full stake transferred to Arbiters and Fund, penalise faulty Arbiters	$\ldots\Lambda^{\mathsf{V}} \xrightarrow{r=0} \Lambda^{\mathsf{A}} \xrightarrow{d=0}$ $\mid\ T^k_n(\mathcal{A}^+) \wedge W \wedge h_p$	$C : -c \to W$ $V : -w \to \mathcal{A}^+$, $V : -(v-w) \to F$, $\forall A_i \in \mathcal{A}^- : -a_i \to F$

3.2 Game mechanics

To analyse all possible game strategies we need to pay special attention to the timelock postconditions t_1–t_4. In fact, their existence splits the whole game into separate timeslots, within which only designated players have ability to "move" (i.e. to sign some particular version of payout scheme or to wait/default/pass move by timeout to the players of the next timeslot). The analysis of this moves and timeslots is given in the Fig. 4.

At the start of the game (setup phase), a complete set of seven possible payout transactions spending the deposited reward and stakes is created. Each of the transactions is pre-signed by all actors that are **not** present on the path leading to the settlement scheme corresponding to the transaction. Through the process of game each participant decides which transaction to sign and, by doing so, it adds to the post-condition criteria for the corresponding settlement scheme from the Table 2 – until the completion and publication of one of them. Each signature on the path is equal to the change in the state of the channel and each game end-leaf (exit) is an implementation of some particular payout

[1]See Table 1 for the summary of the used state parameters

[2]The precondition is defined as a postfix of the state transition path in a part that is different from the normal state transition path $\Lambda^{\mathsf{S}} \xrightarrow{\mathsf{C}} \Lambda^{\mathsf{C}} \xrightarrow{\pi} \Lambda^{\mathsf{V}} \xrightarrow{r=1}$. The postcondition is a set of conditions which have to be satisfied by the computing channel state M.

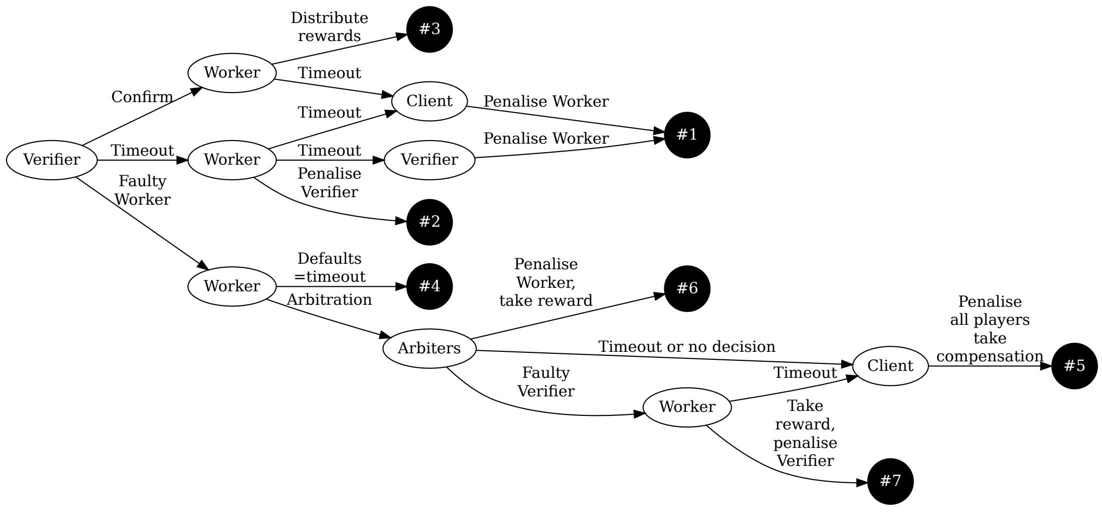


Figure 4: Possible game strategies

scheme.

Thus, by analysing both finite state machine and game theory models we have shown deterministic nature of the settlement algorithm leading to the Byzantine tolerance for the rational set of participants resulting in verified computation and proper reward distribution.

4 Transaction model

4.1 Prometheus contracts

Prometheus contracts $\{\mathsf{P}_i\}$ are defined in the global scope using on-chain transactions $\{\mathsf{T}_{\mathsf{P}_i}\} \mid \Gamma^\Omega$. Each transaction stores a tuple of contract parameters $\mathsf{P} = \langle w, v, n, a, F, \alpha, \beta, \gamma, \delta, \varepsilon, \mu, \sigma, t_1, t_2, t_3, t_4 \rangle$ in explicit form as part of its single `OP_RETURN` output. Explicit storage is required for faster achievement of global state G syncronisation without requirements to run multiple P2P requests and data keeping. The cost of on-chain storage is low, since each of the Prometheus contracts will define some set of parameters matching well-known best practice and there are just few possible options that can satisfy conditions for the Nash equilibrium.

Prometheus contracts are committed to by arbitration offer and computational contract originating transactions by their transaction hash (`txid`, see sections below for the details).

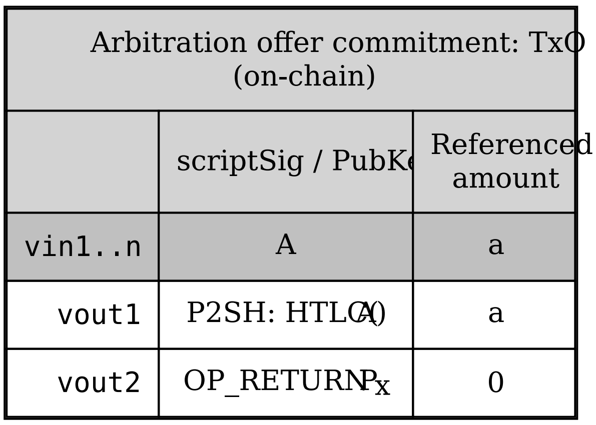
4.2 Arbitration offers

Arbitration offers are contracts created by the parties wishing to participate in the future Arbitrations started by Worker appeals under computational contracts (see section Arbitration below).

Arbitration offer O_i by a party A_i consists of a commitment to a particular Prometheus contract P_x and stake proof a_p that corresponds by its value to the parameter a withing the committed contract: $\mathsf{O} = \langle \mathsf{P}_x, a_p \rangle$. This information is packed into a transaction T_O with a_p amount stored under HTLC P2SH contract in `scriptPubKey` of its first output and with second output referencing corresponding $\mathsf{T}_{\mathsf{P}_x}$ `txid` via `OP_RETURN` code (see Fig. 5). HTLC type of the contract for the stake lock is required to protect from lateral stake spending during the *computational contract* execution (see 4.3 below) and from committing to multiple *computational contracts*.

The purposes of such transaction is to:

- Put information into the global state G about party willing to arbitrate under some specific contract conditions
- Prove availability of the funds required for creation of arbitration stake
- Prove time of the arbitration offer (important for achieving Nash equilibrium for the Prometheus settlement algorithms)



Arbitration offer commitment: TxO (on-chain)		
	scriptSig / PubKe	Referenced amount
`vin1..n`	A	a
`vout1`	P2SH: HTLC(A)	a
`vout2`	OP_RETURN P_X	0

Figure 5: Structure of the arbitration offer commitment transaction T_O

Only well-formed transactions with unspent output are taken as a part of global state of the system: $(\mathbb{A} \equiv \{\mathsf{T}_{\mathsf{O}_i}\} \equiv \{\mathsf{O}_i \mid \Gamma^\Omega\}) \subset \mathsf{P} \subset \mathsf{G}$.

4.3 Computational contracts and stage channel lifecycle

4.3.1 Setup phase overview

Channel M is initiated during its setup phase $\mathsf{M} \mid \Lambda^\mathsf{S}$ off-chain by establishing *computational contract* C – multi-party economically-binding agreement between the Client, Worker node and Verifier node. The agreement is established via P2P communication protocol, described in the next chapter of the specification. Initially, contract in its off-chain version $\mathsf{C} \mid \Lambda^\mathsf{S}$ consists of some selected Prometheus contract P_x, defining main parameters for the contract execution, and initial set of unpublished transactions, namely *commitment transaction* T_0, *arbitration commitment transaction* T_A and *initial set of settlement transactions* consisting of 8 members $\mathsf{T}_{1\ldots8}$: $\mathsf{C} \mid \Lambda^\mathsf{S} = \langle \mathsf{T}_0, \mathsf{T}^\star_\mathsf{A}, \{\mathsf{T}^\star_{1\ldots8}\} \rangle$ ($\mathsf{T}^\star$ denotes partially-signed transaction). All of these transactions are created and signed by utilising Prometheus P2P protocol.

Commitment transaction (notation T_0, txid `Tx0`) locks deposit consisting of reward c from the Client and stakes w, v from the Worker and Verifier nodes stored under a single output (see Fig. 6 for the details on transaction structure). As the second output, the transaction contains `OP_RETURN` scriptPubKey referencing the hash of the corresponding Prometheus contract P transaction the participants are committing to.

Arbitration commitment transaction (notation $\mathsf{T}^{\star}_{\mathsf{A}}$, txid `TxA`) serves the purpose of pre-agreed dispute resolution with economically-binding commitments from all of the participants. Thus, it is fully-signed during the setup phase, and the failure of creating such transaction implies the failure for establishing computational contract: the channel will proceed to the closed state not being published into the global scope. Due to its keystone importance the transaction is created and signed with a special protocol (ceremony), described in the following section.

Settlement transactions (notation $\{\mathsf{T}^{\star}_{i} \mid 1 \leq i \leq 8\}$, txid `Tx1-Tx8`) partially-signed transactions, covering all settlement schemes outside Arbitration. They remain unpublished during the channel existence, i.e. reside off-chain in the channel scope as a part of multiparty channel state $\mathsf{T}^{\star}_{1\ldots5} \in \mathsf{M}(\mathsf{C}_i) \mid \Lambda$. Details on these transactions will be covered in the part related to the channel lifecycle.

All prepared transactions are organised into partially-signed transaction tree Υ originating from the *commitment transaction* (Fig.. 6).

4.3.2 Arbitration commitment ceremony

The ceremony must guarantee:

1. Random selection of Arbiters, that were existed before the creation of the *computation contract* and have committed with some stake. This prevents occupation of the network with pre-registered Arbiters and excludes possibility of the participants to change the existing Arbiters depending on the strategy of contract execution.

2. Commitment from the selected Arbiters to participate in the contract execution.

3. Absence of censorship from one party or any kind of coalition from the contract participants.

These criteria are satisfied by the following procedure:

1. A global set of arbiters $\mathbb{A} \mid \Gamma^{\Omega}$ is known in global on-chain scope before the initiation of the contract and fixed with a snapshot for the full time of contract execution.

2. All of its participants of $\mathbb{A}$ have committed to the same parameters of the contract (i.e. the same Prometheus contract P) and have locked their stake with a proper *arbitration offer* transaction: $\forall A_i \in \mathbb{A} \mid \mathsf{P}_x : \exists \mathsf{O}_i \mid \Gamma^{\Omega} = \langle \mathsf{P}_x \ldots \rangle$. For details on arbitration offers see the corresponding section in the current chapter.

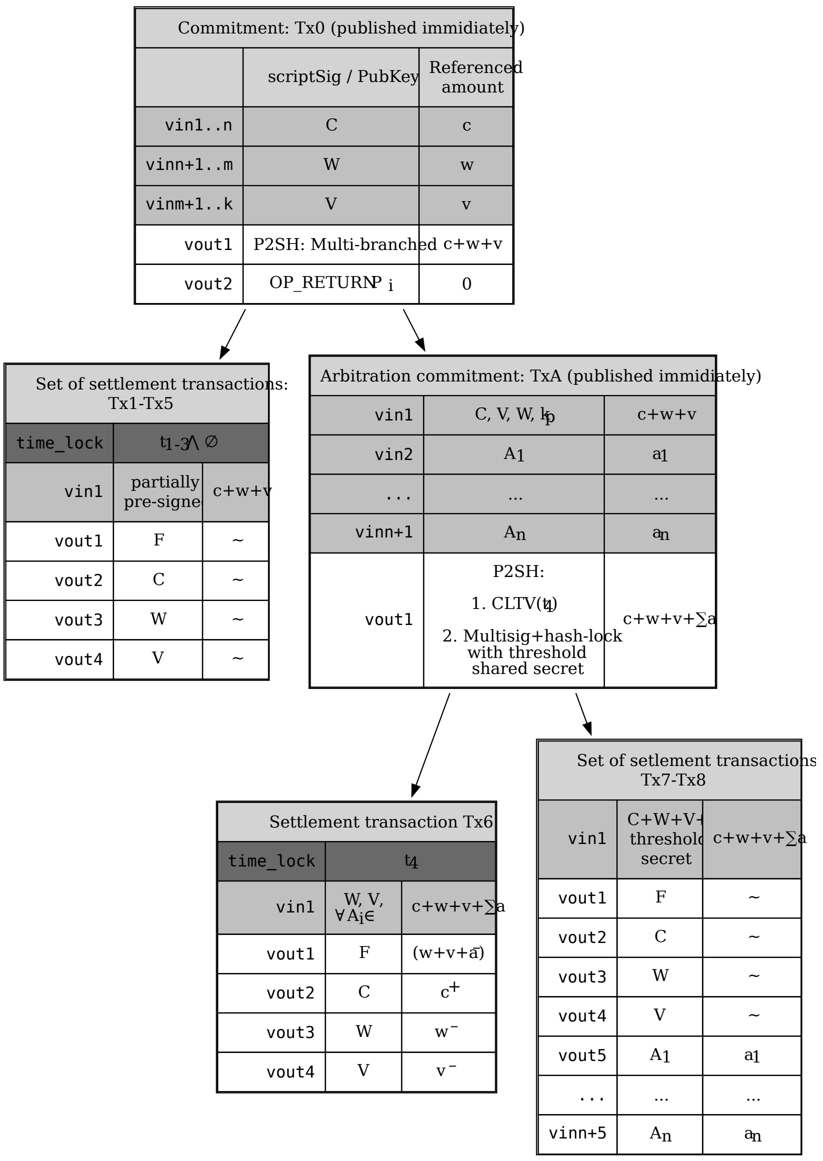


Figure 6: Details on the structure of the transactions inside Υ transaction tree

To be brief, we have simplified some details in the transaction fields: amounts with superscripts x^- and x^+ signify that the recipient will receive original deposit/stake x either in decreased (slashed) or increased (with some additional payment) form. The exact sizes of the slashes and rewards are provided by the Table 2; tildes stands for "amount might vary", please check the Table 2 for specific amounts for different settlement schemes corresponding to the given transaction ids according to the Table 3

3. Deterministic random oracle is utilized by all contract participants to sample $\mathbb{A}$. The simplest form of the oracle is a hash function taken from con-

catenated string of all `txid` used by $\mathbb{A}$ participants in their arbitration offer transactions.

4. Each of C, V, W does the following in an independent manner:
   - Sample set $\mathbb{A}$ in an iterative manner by sending requests to the P2P network and collecting responses of those arbiters who will to participate in the contract. This continues until each of the participants will make a list $\mathcal{A}, |\mathcal{A}| = \mu$ of the parties that have responded to the request.
   - Present to others participants in Λ^{S} his own view on $\mathcal{A}$, namely $\mathcal{A}^W, \mathcal{A}^V, \mathcal{A}^C$.
   - Compute $\mathcal{A}' = \mathcal{A}^W \cup \mathcal{A}^V \cup \mathcal{A}^C$ and reorder it according to the original random sampling order, select subset $\mathcal{A} \subset \mathcal{A}' : \mathcal{A} = \{A_i \in \mathcal{A}' : i \leq \mu\}$.
   - Prepare $\mathsf{T}^{\star}_{\mathsf{A}}$ with all necessary fields (see Fig. 6) including all of the Arbiters from $\mathcal{A}$ and sign $\mathsf{T}^{\star}_{\mathsf{A}}$ with each of the $\mathcal{A}$ participants (via P2P communications)
   - Run Shamir's secret sharing protocol extended with collective secret generation procedure as described in [21] across all selected arbiters $\mathcal{A}$
5. If there were no Byzantine parties, at the end of this ceremony all of participants will have the same version of $\mathsf{T}^{\star}_{\mathsf{A}}$.
6. If the version of $\mathsf{T}^{\star}_{\mathsf{A}}$ does not match between the participants, the channel and computation contract must be closed via $\Lambda^{\mathsf{S}} \to \Lambda^{\times}$ procedure. The Client may try to set up another computing contract by opening new channel and running the same procedure with different W and V.

4.3.3 Completion of channel creation

Transactions, constituting Υ of *computational contract* C during the setup phase Λ^{S} must be (partially)signed in the foolowing order: $\mathsf{T}^{\star}_{\mathsf{A}} \parallel \mathsf{T}^{\star}_{6} \prec \mathsf{T}^{\star}_{7,8} \prec \mathsf{T}^{\star}_{1\ldots5} \prec \mathsf{T}_0$. This order is important so that inability to successfully accomplish the arbitration commitment ceremony will result in proper channel termination without option for possible Byzantine parties to publish the rest of prepared transactions on-chain and failure channel closure.

Upon the completion of the Υ construction the original fully-signed *commitment transaction* must be brought into the global space by the Client, putting its last signature upon it. Following publishing of the commitment transaction and its inclusion into the blockchain the transaction becomes *computing contract* as it is seen by everybody in the global on-chain scope $\mathsf{T}_0 \equiv \mathsf{C} \mid \Gamma^{\Omega} = \langle \mathsf{P}_i, C, W, V, c, w_p, v_p \rangle$ (pls. note that the payload data part of the C is kept off-chain within the channel scope Λ, being known only to channel participants). The transaction tree Υ from now till the end of computation and validation stages persists in the state shown on Fig. 7.

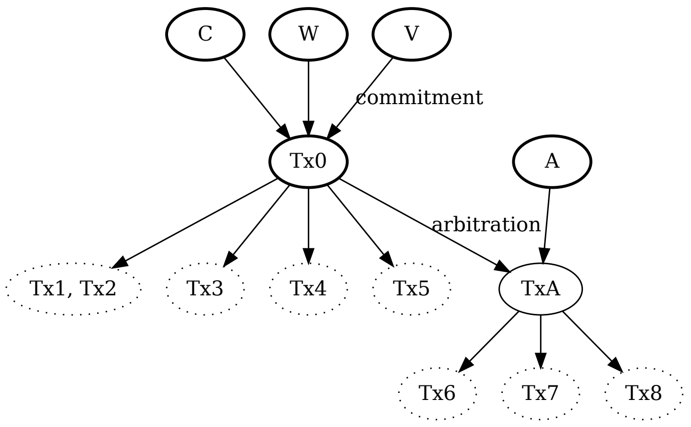


Figure 7: Transaction tree $\Upsilon \mid (\Lambda^{\mathsf{C} \vee \mathsf{V}} \cup \Gamma)$ during computation and verification channel states

On-chain/published transactions (thus, defined in Γ^{Ω} scope) are shown in bold, partially-signed transactions – with dotted line.

4.3.4 Normal channel operation and closure

Prometheus settlement algorithm is implemented utilising partially-signed transactions from the Υ (Fig. 7), prepared at the setup phase by all involved parties. They are consist of two subsets: *pre-arbitration* and *post-arbitration transactions.*

Pre-arbitration transactions are specified in the Table 3 and Fig. 6-7 by their ids `Tx1-Tx5`. These transactions cover all settlement schemes outside Arbitration and remain unpublished, i.e. residing off-chain in the channel scope as a part of multiparty channel state $\mathsf{T}_{1 \ldots 5} \in \mathsf{M}(\mathsf{C}_i) \mid \Lambda$. Upon corresponding state transitions from the Fig. 2 they will be signed by the proper party (see "Post-signed" column in the Table 3), and published into the global scope to be included on-chain. This completes computing contract C, invalidates local scope Λ and closes the state channel $\mathsf{M}(\mathsf{C})$ according to the corresponding settlement scheme from the Table 2:

$(\mathsf{T}_j^\star \mid \Lambda) \xrightarrow{C \vee W \vee V} (\mathsf{T}_j \mid \Gamma^{\Omega});\ \exists j : 1 \leq j \leq 5 \quad \Longrightarrow \quad \mathsf{M}(\mathsf{C}) \mid (\Lambda^{\mathsf{S} \vee \mathsf{C} \vee \mathsf{V}} \rightarrow \Lambda^{\times}_{1 \ldots 4})$

For the settlement schemes that have to reveal computation results to the client (#2 and #3), the signing Worker node must additionally include into the signed transaction *preimage* h_p that was originally used for the results encryption. This is required by unlocking script of `Tx0` which contains hash-lock conditions in branches related to these settlement schemes (see below section on the unlocking scripts details).

4.3.5 Arbitration

In order to transition the channel state into the Arbitration, $\mathsf{M}(\mathsf{C}) \mid \Lambda^{\mathsf{C}} \rightarrow \Lambda^{\mathsf{A}}$, Worker must submit appeal by including its *arbitration-unlocking preimage* k_p into `scriptSig` of `TxA` (which is already signed by the Verifier as part of its *verdict* on the Worker fault) and publishing the transaction:

Table 3: Settlement transactions and their finalisation conditions

Id	Payout Schedule	Pre-signed	Post-signed	Hashlock	nTimelock
`Tx1`	#1	W, C	V	-	t_3
`Tx2`	#1	W, V	C	-	t_3
`Tx3`	#2	V, C	W	h_p	t_2
`Tx4`	#3	C, W	V	h_p	-
`Tx5`	#4	C, W	V	-	t_1
`TxA`	#5-7	C, W, A	V	k_p	-
`Tx6`	#5	V, W, A	C	-	t_4
`Tx7`	#6	C, W, V	-	$T^k_n(\mathcal{A}^+)$	-
`Tx8`	#7	C, W, V	-	$T^k_n(\mathcal{A}^+)$, h_p	w

$(\mathsf{T}^\star_A \mid \Lambda^\mathsf{V}) \xrightarrow{k_p} (\mathsf{T}_A \mid \Gamma^\Omega) \quad \Longrightarrow \quad \mathsf{M}(\mathsf{C}) \mid (\Lambda^\mathsf{V} \xrightarrow{\texttt{appeal}} \Lambda^\mathsf{A})$.

This signals to the arbiters in $\mathcal{A}$ to initiate their verification procedure on the data from M, mainly PCP proofs π.

Arbitration is completed in one of the ways defined in the previous chapter. Each of them has a corresponding pre-signed transaction `Tx6-8` (Table 3), which can be finalised by the interested party and published into the global scope, which will close the channel and complete settlement:

$(\mathsf{T}^\star_j \mid \Lambda) \xrightarrow{C \vee \mathcal{A}^+} (\mathsf{T}_j \mid \Gamma^\Omega);\ \exists j : 6 \leq j \leq 8 \quad \Longrightarrow \quad \mathsf{M}(\mathsf{C}) \mid (\Lambda^\mathsf{A} \to \Lambda^\times_{5\ldots7})$

Arbiters that came to the *coherent decision* sign transactions `Tx7-Tx8` by revealing their parts of the collectively shared secret with a threshold signature T^k_n, the corresponding revealed secret is a *preimage* for the hashlock condition in `scriptSig` part of the transactions.

5 Conclusion

In the present work we have formally defined Prometheus: a protocol for running high-load & machine learning computing inside a trustless decentralized environment in Byzantine-tolerant way, utilizing zero knowledge computation integrity arguments, economic incentives and p2p networking. The paper have covered protocol design rationale, architecture, formal protocol definition and specified protocol operations with demonstrable Nash equilibrium, as well as transaction model for implementation with Bitcoin blockchain and script functionality.

References

[1] The specs for libp2p and associated submodules. https://github.com/libp2p/specs.

[2] Webassembly. http://webassembly.org/, 2016.

[3] The #freeai manifesto. https://manifesto.ai/, 2018.

[4] A. Acquisti, R. Dingledine, and P. Syverson. On the economics of anonymity. In R. N. Wright, editor, *Financial Cryptography*, pages 84–102, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[5] A. Back, M. Corallo, L. Dashjr, M. Friedenbach, G. Maxwell, A. Miller, A. Poelstra, J. Timon, and P. Wuille. Enabling blockchain innovations with pegged sidechains. 2014.

[6] C. Badertscher, J. Garay, U. Maurer, D. Tschudi, and V. Zikas. But why does it work? a rational protocol design treatment of bitcoin. In J. B. Nielsen and V. Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2018*, pages 34–65, Cham, 2018. Springer International Publishing.

[7] E. Ben-Sasson, A. Chiesa, and N. Spooner. Interactive oracle proofs. Cryptology ePrint Archive, Report 2016/116, 2016. https://eprint.iacr.org/2016/116.

[8] J. Benet. Ipfs - content addressed, versioned, p2p file system. 2016.

[9] K. Bennett, C. Grothoff, T. Horozov, I. Patrascu, and T. Stef. Gnunet – a truly anonymous networking infrastructure. In *In: Proc. Privacy Enhancing Technologies Workshop (PET*. Citeseer, 2002.

[10] B. Cohen. Incentives build robustness in bittorrent. In *Workshop on Economics of Peer-to-Peer systems*, volume 6, pages 68–72, 2003.

[11] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The second-generation onion router. https://svn.torproject.org/svn/projects/design-paper/tor-design.pdf.

[12] D. Fudenberg and J. Tirole. *Game Theory*. MIT Press, Cambridge, MA, 1991.

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

[14] N. Houy. The bitcoin mining game. *Ledger*, 1:53–68, 2016.

[15] J. H. Howard, M. L. Kazar, S. G. Menees, D. A. Nichols, M. Satyanarayanan, R. N. Sidebotham, and M. J. West. Scale and performance in a distributed file system. *ACM Transactions on Computer Systems (TOCS)*, 6(1):51–81, 1988.

[16] N. Kshetri. Privacy and security issues in cloud computing: The role of institutions and institutional evolution. *Telecommunications Policy*, 37(4):372 – 386, 2013.

[17] R. Kumaresan and I. Bentov. How to use bitcoin to incentivize correct computations. In *ACM Conference on Computer and Communications Security*, 2014.

[18] J.-J. Laffont and D. Martimort. *The theory of incentives: the principal-agent model*. Princeton University Press Princeton, N.J. ; Oxford, 2002.

[19] P. Maymounkov and D. Mazières. Kademlia: A peer-to-peer information system based on the xor metric. In *IPTPS '01 Revised Papers from the First International Workshop on Peer-to-Peer Systems*, pages 53–65, 2002.

[20] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2008.

[21] M. Orlovsky. Typhon: trustless bitcoin sidechain protocol. https://github.com/dr-orlovsky/typhon-spec, 2019.

[22] M. Orlovsky and S. Sachtachtinskagia. Incentive-based interactive settlement scheme for computational integrity proofs (ibisscip). https://github.com/pandoracore/ibiss-spec/blob/master/ibiss-spec.pdf, 2019.

[23] J. Teutsch and C. Reitwießner. A scalable verification solution for blockchains. 2017.

[24] B. T. Ward and J. C. Sipior. The internet jurisdiction risk of cloud computing. *Information Systems Management*, 27(4):334–339, 2010.

[25] Q. Zhang, L. Cheng, and R. Boutaba. Cloud computing: state-of-the-art and research challenges. *Journal of Internet Services and Applications*, 1(1):7–18, May 2010.