Verified-Compute Protocol for Minimally-Trusted Execution Environments

bill.gleim@consensys.net

mario.alvarez@consensys.net

Abstract

Presented below is an introduction to a protocol specification for verified-compute in compute marketplaces containing possibly malicious worker nodes. An interactive verifiable computation system architecture is demonstrated to provide trustful operations to users in a compute environment containing malicious nodes and nodes of questionable trust.

Keywords: Verifiable Compute, DLT, Blockchain, Ethereum

1. Introduction

In Interactive Verifiable Computation (IVC), a powerful machine (the Prover) helps a weaker machine (the Verifier) calculate the result of running a computation, in such a way that the Verifier can check the validity of the result with high confidence without needing to run the full computation.

CoVe is a framework and reference implementation for integrating IVC protocols with Ethereum, in such a way that Verifiers can contract the services of Provers in a marketplace, that honest Provers are rewarded for their efforts, and that dishonest Verifiers and Provers are punished. This ensures that Provers and Verifiers both have an incentive to participate in a mutually-beneficial market for trustworthy computation. CoVe achieves limited data sharing between the Prover and Verifier actors, but complete data and compute privacy is not provided.

IVC takes the form of a protocol, in which the Prover and Verifier take turns transmitting and computing on messages to each other. We can think of these interactions as having a transcript, which looks something like Table 1 (where blocks labeled Px are sent from Prover to Verifier, and vice versa for blocks labeled Vx. At the end of the protocol, V obtains a final result, and

t0	**t1**	**t2**	**t3**	**t4**	**t5**	**tn**	**tn+1**
P_1	V_1	P_2	V_2	P_3	V_3	P_n	V Accepts or Rejects

Table 1: Prover-Verifier message transmission sequence

either accepts the result (i.e., assuming the Verifier ran its code correctly, there is not more than a vanishingly small chance that the result V has obtained is wrong) or rejects it (signifying that the protocol was not run correctly.)

In practice, rejection is also possible before the final stage, as the Verifier may be able to detect errors on the part of the Prover earlier on. Later, we will see how adding a notion of rejection for the Prover can help eliminate fraud on the part of the Verifier.

2. Interactive Verifiable Computation (IVC) Details

2.1. Interactive more performant than Non-Interactive

There has been a lot of excitement in the distributed ledger technology (including but not limited to blockchain) community about non-interactive proofs, especially non-interactive zero-knowledge proofs of knowledge (ZK-SNARKS). While techniques such as the Fiat-Shamir Heuristic exist for translating interactive proofs into non-interactive proofs, this can come at the cost of performance, depending on the domain. Thus, supporting interactive proofs in a distributed ledger context has independent value.

2.2. Limitations of IVC without Distributed Ledger Technology

Interactive Verifiable Compute (IVC) has the following pain points:

1. Overhead imposed on Prover and Verifier
    (a) Distributed ledger technology can't solve this, but choosing a specific domain wisely can (see, for example, SafetyNets [2])
2. If the Prover and Verifier are not mutually trusting - our use case - there are the following additional pain points:
    (a) How can the Verifier pay the Prover for the Prover's work? If the Verifier pays the Prover up front, how can the Verifier have recourse if the computation fails because of an error on the Prover's part?

   (b) If the Verifier pays the Prover after the computation is complete and only if it is successful, what is to prevent the Verifier from disputing correct computations performed by the Prover - thus receiving computation at the Prover's expense and without compensating the Prover?

The thesis behind the IVC protocol is that all of the pain points under (2) above can be resolved or at least greatly ameliorated using (in our case) an Ethereum-based incentive mechanism facilitating correct computation in distributed ledger technology environments.

2.3. IVC Protocol Walkthrough

There are three roles in the CoVe framework. These are the following:

1. Verifier, the lightweight consumer of the computation.
2. Prover, the heavyweight performer of the computation.
3. Judge, a trustworthy machine comparable in power to the Verifier that is part of an incentive system, described below, to deter dishonest Provers and Verifiers.

Here is an example of what a typical use of CoVe might look like:

Vernon (V) owns a lightweight node that wants to perform a particular inference in a particular neural network, but lacks the computational power to perform this inference on its own. Vernon is willing to stake an amount of Ether or token that is large in value relative to the value of the computations he wants performed. Then, Vernon goes to an Ethereum-based decentralized marketplace and puts out a work order, precisely describing the computation to be performed (including the data to perform it on). These orders can be directly encoded on-chain on the Ethereum public main network, or stored off-chain with only hashes left on-chain; the details of implementing a decentralized purchase-order system like this efficiently are beyond the scope of this document. Vernon must put into escrow an amount equal to the requested price of the computation he is ordering.

Priscilla (P) owns a heavyweight compute node that can fulfill Vernon's order. As she finds Vernon's price acceptable, she signs a commitment to perform this computation on his behalf.

Priscilla and Vernon then begin a modified version of the IVC protocol associated with the computation they want to perform:

- Priscilla generates the first message she will send Vernon in the IVC protocol, and publishes it irrevocably.
- Vernon runs his end of the computation, and submits his message back to Priscilla. He also irrevocably publishes an attestation regarding the message he sent.
- Priscilla runs the Verifier algorithm on her own machine, and then signs and publishes a commitment attesting to the validity of the result Vernon sent her.
    - If Priscilla gets a different output from Vernon, at this point she needs to publish a commitment contesting Vernon's response. Adjudication, as described below, will need to be involved.
- Based on this input from Vernon, Priscilla then sends Vernon her second IVC message, publishing it irrevocably.
- this process repeats...
- ...until Priscilla's last message to Vernon, which she publishes irrevocably.
- Vernon then has the output of the computation, and can decide whether to accept or reject (publishing a commitment saying one or the other).

If Vernon accepts, he publishes an irrevocable attestation certifying that the IVC protocol has informed him that the computation is correct. In this case, the funds held in escrow will be immediately released to Priscilla for withdrawal via an Ethereum smart contract transaction.

If Vernon rejects, adjudication is necessary to determine the final result.

2.4. CoVe Adjudication Walkthrough

If, as described above, Priscilla rejects one of Vernon's responses, or Vernon rejects the final result of the computation, some adjudication is necessary to ensure fairness.

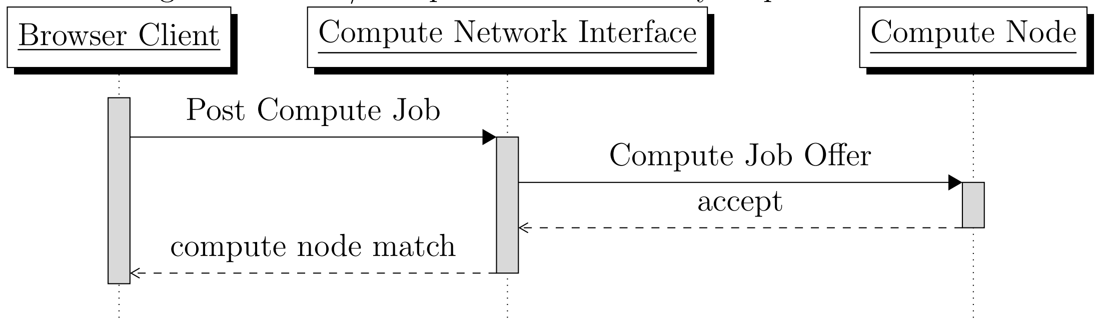
Suppose **John** (J) runs a trustworthy node at least equal in power to Vernon's machine. Setting aside the issue of how such trust is established and how John is incentivized, John can become a Judge in the network. He can be used to handle such cases, by running the following algorithm:

- John takes Priscilla's first committed output, and runs the Verifier algorithm against it.
- If the result he obtains is not equal to Vernon's first response, Vernon is incorrect and the algorithm ends.
- Otherwise, John takes Priscilla's second committed output, and runs the Verifier algorithm against it.
- Again, if the result he obtains is not equal to Vernon's response, Vernon is incorrect and the algorithm ends.
- This process continues until...
- ...John takes Priscilla's final committed output, and runs the Verifier algorithm against it.
- If John's Verifier accepts, then the computation is correct.
  - That is, if Vernon was the one who disputed the computation, he is incorrect.
  - Otherwise, Priscilla is the one who is incorrect.
- If John's Verifier rejects, then the computation is incorrect.
  - If Priscilla had a legitimate reason to dispute the computation (i.e., Vernon lying about an intermediate result) it would have been caught before this point.
  - Thus, Priscilla must be the one who is incorrect.

At the end, John will decide that either Vernon is correct - in which case his funds are returned and Priscilla's stake, if any, is slashed - or that Priscilla is correct, in which case Vernon's stake is slashed. (Note that, to prevent spam/denial of service, Priscilla probably also needs to provide sufficient stake to be eligible as a Prover).

The role of the Judge (John) could potentially be filled by some other trustworthy off-chain compute mechanism (e.g. TrueBit, iExec). If the verifier's cost is low enough this could even be done on-chain, although that is unlikely to be practical.

Figure 1. Client/Compute Node Discovery Sequence



2.5. CoVe IVC Implementation Design

CoVe IVC is implemented as a web application running on the decentralized Ethereum network (i.e., DApp). We provide two sequence diagrams illustrating the implementation design. The first sequence diagram (Figure 1) demonstrates the discovery, or client/compute-node matching, process. The second diagram (Figure 2) illustrates the client and compute-node sequential interactions with each other facilitated by the Ethereum network.

2.6. CoVe IVC Implementation Details

CoVe Interactive Verifiable Compute is implemented using a variant of Pepper [1]. CoVe provides a node.js API server and interfaces for encrypted IPFS storage facilitating the self-sovereign private distribution of data to selected compute nodes. CoVe represents important compute-network state transitions (e.g., posting of encrypted computed outputs to IPFS) on the Ethereum main net.

Applications from CoVe IVC include, for example, genetic sequence substring matching and high-dimensional data clustering. These CoVe IVC applications are available on the iExec Ethereum-based distributed compute marketplace.

Figure 2. Client/Compute Node/Ethereum Network Sequence

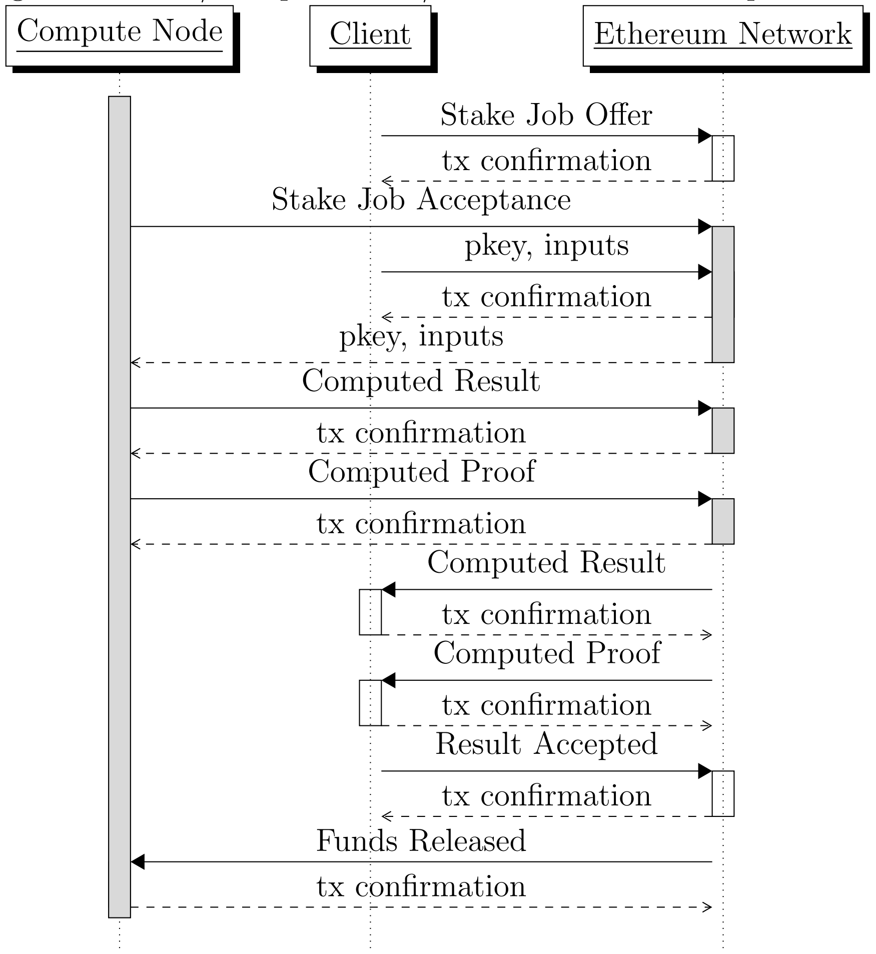


Figure 3. IVC Initialization Sequence (user provides inputs, prove key)

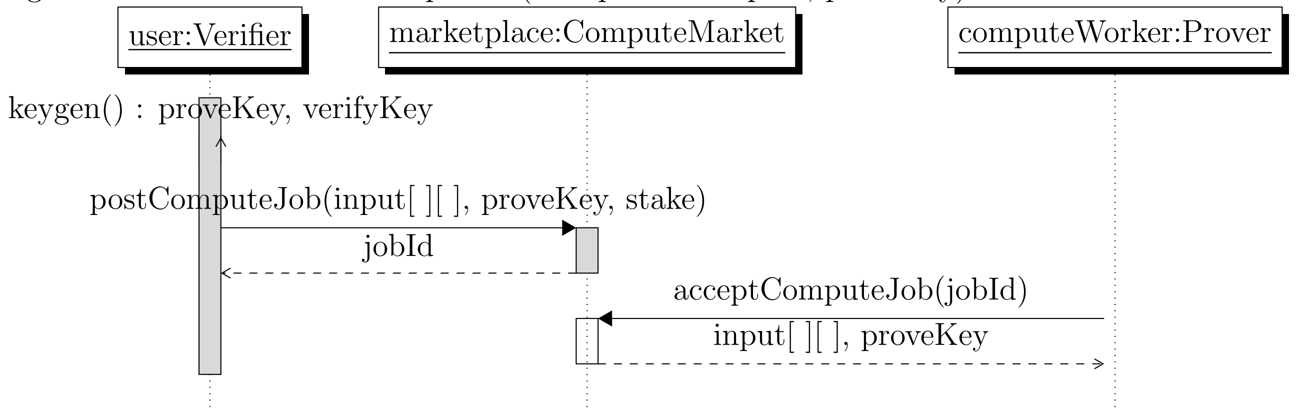
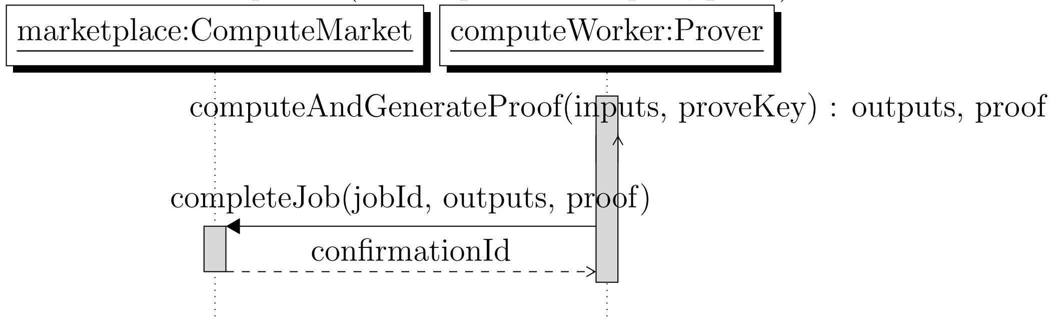

user:Verifier
marketplace:ComputeMarket
computeWorker:Prover
keygen() : proveKey, verifyKey
postComputeJob(input[ ][ ], proveKey, stake)
jobId
acceptComputeJob(jobId)
input[ ][ ], proveKey

Figure 4. IVC Invocation Sequence (worker provides outputs, proof)

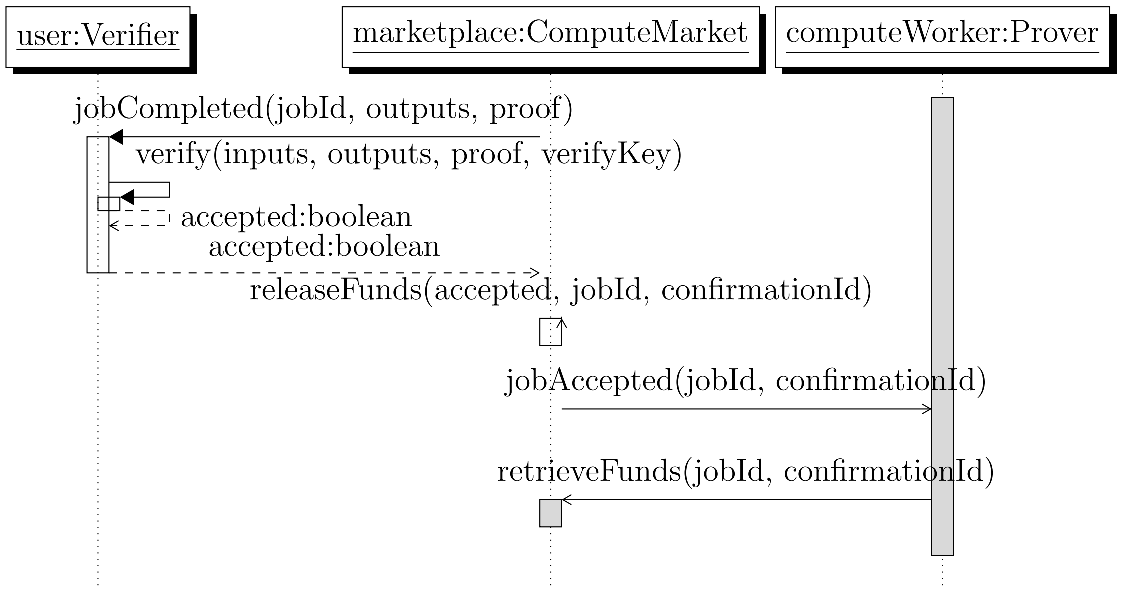

marketplace:ComputeMarket
computeWorker:Prover
computeAndGenerateProof(inputs, proveKey) : outputs, proof
completeJob(jobId, outputs, proof)
confirmationId

2.7. CoVe IVC in terms of EEA Trusted Compute specification v0.1

In terms of the Enterprise Ethereum Alliance early technical specifications, CoVe IVC and the IVC protocol fit early drafts of the Trusted Compute specification. Particularly, v0.1 Trusted Compute API framework separates component interaction into three stages: **initialization**, **invocation**, and **verification**.

Without directly addressing on-chain/off-chain interactions, Figures 3 through 5 illustrate CoVe IVC component interactions in the vocabulary and structure of EEA Trusted Compute API specification v0.1 **initialization**, **invocation**, and **verification** stages.

Figure 5. IVC Verification Sequence (user verifies outputs, accepts or rejects)


user:Verifier
marketplace:ComputeMarket
computeWorker:Prover
jobCompleted(jobId, outputs, proof)
verify(inputs, outputs, proof, verifyKey)
accepted:boolean
accepted:boolean
releaseFunds(accepted, jobId, confirmationId)
jobAccepted(jobId, confirmationId)
retrieveFunds(jobId, confirmationId)

3. References

[1] Pepper. `https://github.com/pepper-project/pepper`, 2017.

[2] Zahra Ghodsi, Tianyu Gu, and Siddharth Garg. Safetynets: Verifiable execution of deep neural networks on an untrusted cloud. *CoRR*, abs/1706.10268, 2017. URL `http://arxiv.org/abs/1706.10268`.