Off-chain Execution and Verification of Computationally Intensive Smart Contracts

Emrah Sariboz, Kartick Kolachala, Gaurav Panwar, Roopa Vishwanathan, and Satyajayant Misra
Department of Computer Science
New Mexico State University
Las Cruces, NM, USA
{emrah, kart1712, gpanwar, roopav, misra}@nmsu.edu

***Abstract*—We propose a novel framework for off-chain execution and verification of computationally-intensive smart contracts. Our framework is the first solution that avoids duplication of computing effort across multiple contractors, does not require trusted execution environments, supports computations that do not have deterministic results, and supports general-purpose computations written in a high-level language. Our experiments reveal that some intensive applications may require as much as 141 million gas, approximately 71x more than the current block gas limit for computation in Ethereum today, and can be avoided by utilizing the proposed framework.**

***Index Terms*—smart contract verification, verifiable computation**

I. Introduction

A smart contract is a computer program that resides on the Ethereum blockchain and gets executed automatically when predetermined conditions are met. Depending on the complexity, every transaction that modifies a smart contract's state consumes a certain amount of gas (the unit of cost in the Ethereum blockchain). As a result of this, it becomes infeasible to use smart contracts for computationally intensive applications such as image recognition and zero-knowledge proofs. In this paper, we refer to such contracts as *computationally intensive smart contracts* (CICs).

Recent studies have explored alternative solutions to eliminate the cost and make CIC execution scalable. Proposed solutions to this end either replicate the CIC's execution across a small subset of nodes or require a Trusted Execution Environment (TEE), which engenders greater trust. An alternative to the aforementioned methods is to outsource the CIC computation to a third party that does the computation and generates a proof of correctness for the same, that can be verified in polynomial time. Using this approach, the client can verify the returned computation's correctness in a much more efficient manner than re-executing it. Our work falls into verifiable computation category where we propose a solution that is scalable, avoids duplicating computations, and does not require tamper-resistant hardware or trusted execution environments.

Research supported by NSF awards #1800088, #2028797, #1914635, Intel Labs, and the Federal Aviation Administration (FAA). Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the NSF, FAA, and Intel Inc.

978-0-7381-1420-0/21/$31.00 ©2021 IEEE

II. Related Work

Trusted Hardware: TEE has been adopted to alleviate scalability and confidentiality obstructions of smart contracts in [1]–[3]. However, recent studies have identified several attack on SGX — we avoid the impact as we do not need SGX [4]–[10]. **Replicated Computation:** Outsourcing CIC execution to a set of delegators has been proposed by [11]; however, this model suffers from the large overhead of replicated computation and lacks support for randomized computations, which we address in our work. **Verifiable Computation:** Interactive proofs (IPs) [12] and probabilistically checkable proofs (PCPs) [13] laid the foundations of provable verifiable computation which has been studied in [14]–[21]. Despite promising asymptotics, these proof systems are highly impractical and may take inordinately long to verify instances with small input sizes [22]. Another line of work applies the above theoretical foundations to practice on cloud computing settings studied in [23]–[25]; however, they are far from being scalable for general-purpose computation. Adoption of zk-SNARKs [26] to verify smart contracts was proposed by [27]; however, their solution requires that the application code be written in a domain-specific language that they designed. This differs from our work as our work supports computations that are written in a high-level language.

III. Construction

The components of our framework are as follows: a client (Alice) who wishes to outsource a computationally intensive job, a worker (Bob) who does the computation for the client in exchange for some monetary reward, a miner (Charlie) to validate the transactions, and *Broker contract*, a smart contract which acts as an intermediary between the client and the worker.

Client's Operations: Alice writes the details of the smart contract to be executed to her publicly accessible server Step 1 in Figure 1. The details contain the inputs needed for execution, the fee given to a worker, the collateral the worker needs to deposit to register for this job, and the maximum time she is willing to allot for the computation result to be

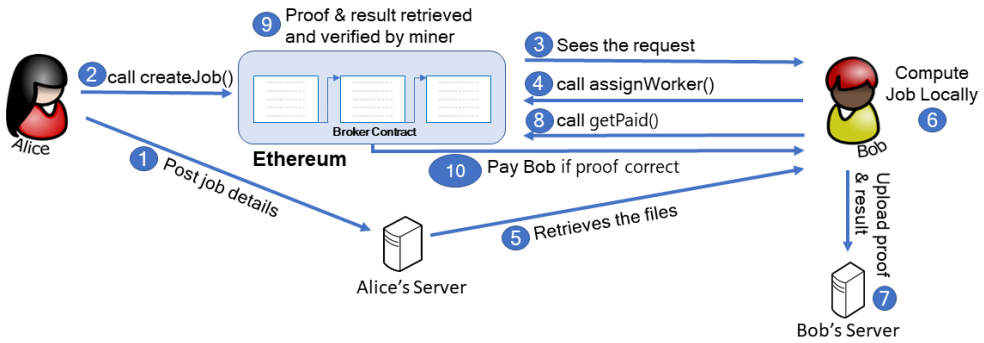

Fig. 1. Schematic diagram of interactions between the entities and the corresponding function calls in the framework.

delivered to her. She posts this job creation request to the blockchain by interacting with the *Broker Contract* in Step 2. This request contains the URL of her server, which has all the aforementioned details.

Worker's Operations: If Bob is interested in executing the computation, he goes to the specified server URL to check whether he has enough resources to complete the computation within the requested time interval in Step 3 and registers for the job by depositing the required collateral to *Broker Contract* in Step 4. He retrieves the inputs needed for the computation from Alice's server in Step 5. He performs the computation locally in Step 6, generates proof of correctness, and uploads them to his server in Step 7. He then submits the URL to *Broker Contract* for the verification by calling *getPaid()* function in Step 8 to get compensated for his work which internally starts the proof-verification mechanism.

Miner's Operations: Charlie picks up the transaction posted by Bob, executes the *Broker Contract*, and retrieves the result and proof from Bob's server in Step 9. The *Broker Contract* checks whether the proof was posted within a specified time limit, verifies the proof and result, and posts them to Alice's server. *Broker Contract* outputs a transaction paying Bob his fee and refunding his collateral if the verifications are successful in Step 10. However, Alice gets refunded her fee and also gets Bob's collateral if the verifications fail.

TABLE I
THE MEAN AND STANDARD DEVIATION FOR COMPUTATIONALLY INTENSIVE APPLICATIONS

Computation	Input	KeyGen (s)	ProofGen (s)	Verify (ms)
Matrix Mult.	70×70	40.25±1.48	117.91±4.24	2±2
	110×110	158.43±8.89	487.46±93.50	9±10
Image Match.	45×45	32.23±0.76	75.12±1.89	70±61
	85×85	115.89±3.32	317.88±8.68	9±2
MultiVar Poly	500040	36.23±2.58	140.81±7.12	8±2
	644170	65.70±3.60	1220.82±8.59	8±2
Floyd-Warshall	16×16	45.57±3.00	112.06±6.86	1±2
	25×25	166.40±4.36	514.99±13.35	3±7

IV. Results And Evaluation

The proposed framework's performance has been evaluated on four computationally intensive applications as in [22].

Matrix Multiplication takes two $n \times n$ matrices as an input, M_1 and M_2, and computes $M_1 \cdot M_2$. **Image Matching** takes a $k_w \times k_h$ ($k_w = k_h = 3$) sized image kernel and computes the point in an image where the minimum difference happens between the image and the kernel. **Multi-Variate polynomial evaluation** takes a polynomial of degree m, containing $(m+1)^k$ coefficients, and evaluates it over k ($k = 5$) variables taken as inputs. **Floyd-Warshall** algorithm takes an $n \times n$ matrix representing the adjacency matrix of an n-vertex graph. It computes the shortest paths among all the vertices.

According to our calculations, the gas required to implement these applications in smart contracts is infeasible given the current block gas limit of ≈12 million [28], e.g., 142 million gas units for image matching.

Evaluation: The above applications were written in C and first transformed into an arithmetic circuit. Next, the Pinocchio compiler is used to generate Quadratic Arithmetic Program (QAP), evaluation and verification keys [22]. In our experiments, the key generation phase is completed by the client and given to the worker along with the QAP. On receiving these parameters, the worker executes the code and posts the proof to the server he controls.

Our experimental results, detailed in Table I, show that our framework provides quick proof verification, for different sizes of input parameters for all applications. The framework also maintains a constant proof size of 288 bytes in all cases. As expected, we have noticed an increase in the proof generation time with an increase in the application parameters' size. This growth was linear for all except image matching, which was super linear due to an increase in the number of multiplication gates and equality comparisons in the equivalent arithmetic table.

V. Conclusion

We proposed a novel framework for execution and the verification of the CICs by offloading them to a computationally powerful entity using an incentive mechanism. Unlike other proposed solutions, our work prevents replicated computation, eliminates the need for TEEs, and supports computations with random results.

REFERENCES

[1] P. Das, L. Eckey, T. Frassetto, D. Gens, K. Hostáková, P. Jauernig, S. Faust, and A.-R. Sadeghi, "Fastkitten: practical smart contracts on bitcoin," in *28th USENIX Security Symposium (USENIX Security 19)*, 2019, pp. 801–818.

[2] R. Cheng, F. Zhang, J. Kos, W. He, N. Hynes, N. Johnson, A. Juels, A. Miller, and D. Song, "Ekiden: A platform for confidentiality-preserving, trustworthy, and performant smart contract execution," *arXiv preprint arXiv:1804.05141*, 2018.

[3] K. Wüst, S. Matetic, S. Egli, K. Kostiainen, and S. Capkun, "Ace: Asynchronous and concurrent execution of complex smart contracts," 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. 587–600.

[4] M. Hähnel, W. Cui, and M. Peinado, "High-resolution side channels for untrusted operating systems," in *2017 USENIX Annual Technical Conference (USENIX ATC) 17*, 2017, pp. 299–312.

[5] A. Moghimi, G. Irazoqui, and T. Eisenbarth, "Cachezoom: How SGX amplifies the power of cache attacks," in *International Conference on Cryptographic Hardware and Embedded Systems*. Springer, 2017, pp. 69–90.

[6] J. Van Bulck, N. Weichbrodt, R. Kapitza, F. Piessens, and R. Strackx, "Telling your secrets without page faults: Stealthy page table-based attacks on enclaved execution," in *26th USENIX Security Symposium (USENIX Security 17)*, 2017, pp. 1041–1056.

[7] S. Lee, M.-W. Shih, P. Gera, T. Kim, H. Kim, and M. Peinado, "Inferring fine-grained control flow inside SGX enclaves with branch shadowing," in *26th USENIX Security Symposium (USENIX Security 17)*, 2017, pp. 557–574.

[8] Y. Xu, W. Cui, and M. Peinado, "Controlled-channel attacks: Deterministic side channels for untrusted operating systems," in *2015 IEEE Symposium on Security and Privacy*. IEEE, 2015, pp. 640–656.

[9] A. Nilsson, P. N. Bideh, and J. Brorsson, "A Survey of Published Attacks on Intel SGX," Tech. Rep.

[10] J. Götzfried, M. Eckert, S. Schinzel, and T. Müller, "Cache attacks on Intel SGX," in *Proceedings of the 10th European Workshop on Systems Security*, 2017, pp. 1–6.

[11] S. Das, V. J. Ribeiro, and A. Anand, "Yoda: Enabling computationally intensive contracts on blockchains with byzantine and selfish nodes," *arXiv preprint arXiv:1811.03265*, 2018.

[12] U. Feige, S. Goldwasser, L. Lovász, S. Safra, and M. Szegedy, "Interactive proofs and the hardness of approximating cliques," *Journal of the ACM (JACM)*, vol. 43, no. 2, pp. 268–292, 1996.

[13] S. Arora and S. Safra, "Probabilistic checking of proofs: A new characterization of np," *Journal of the ACM (JACM)*, vol. 45, no. 1, pp. 70–122, 1998.

[14] P. Golle and S. G. Stubblebine, "Secure distributed computing in a commercial environment," in *Financial Cryptography, 5th International Conference, FC 2001, Grand Cayman, British West Indies, February 19-22, 2002, Proceedings*, 2001, pp. 279–294.

[15] W. Du and M. T. Goodrich, "Searching for high-value rare events with uncheatable grid computing," in *Applied Cryptography and Network Security, Third International Conference, ACNS*, 2005, pp. 122–137.

[16] R. Sion, "Query execution assurance for outsourced databases," in *Proceedings of the 31st International Conference on Very Large Data Bases VLDB*, 2005, pp. 601–612.

[17] P. Golle and I. Mironov, "Uncheatable distributed computations," in *Topics in Cryptology - CT-RSA 2001, The Cryptographer's Track at RSA Conference 2001, San Francisco, CA, USA, April 8-12, 2001, Proceedings*, 2001, pp. 425–440.

[18] G. Cormode, M. Mitzenmacher, and J. Thaler, "Practical verified computation with streaming interactive proofs," in *Innovations in Theoretical Computer Science ITCS*, 2012, pp. 90–112.

[19] S. Goldwasser, Y. T. Kalai, and G. N. Rothblum, "Delegating computation: interactive proofs for muggles," in *Proceedings of the 40th Annual ACM Symposium on Theory of Computing STOC*, 2008, pp. 113–122.

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

[21] K. Chung, Y. T. Kalai, and S. P. Vadhan, "Improved delegation of computation using fully homomorphic encryption," in *Advances in Cryptology - CRYPTO*, 2010, pp. 483–501.

[22] B. Parno, J. Howell, C. Gentry, and M. Raykova, "Pinocchio: Nearly practical verifiable computation," in *2013 IEEE Symposium on Security and Privacy*. IEEE, 2013, pp. 238–252.

[23] S. T. V. Setty, R. McPherson, A. J. Blumberg, and M. Walfish, "Making argument systems for outsourced computation practical (sometimes)," in *19th Annual Network and Distributed System Security Symposium, NDSS*. The Internet Society, 2012.

[24] S. T. V. Setty, V. Vu, N. Panpalia, B. Braun, A. J. Blumberg, and M. Walfish, "Taking proof-based verified computation a few steps closer to practicality," in *Proceedings of the 21th USENIX Security Symposium*. USENIX Association, 2012, pp. 253–268.

[25] V. Vu, S. T. V. Setty, A. J. Blumberg, and M. Walfish, "A hybrid architecture for interactive verifiable computation," in *IEEE Symposium on Security and Privacy, SP*. IEEE Computer Society, 2013, pp. 223–237.

[26] N. Bitansky, R. Canetti, A. Chiesa, and E. Tromer, "From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again," in *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, ser. ITCS '12. New York, NY, USA: Association for Computing Machinery, 2012, p. 326–349. [Online]. Available: https://doi.org/10.1145/2090236.2090263

[27] J. Eberhardt and S. Tai, "Zokrates - scalable privacy-preserving off-chain computations," *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, pp. 1084–1091, 2018.

[28] *Ethereum Stats*, 2020 (accessed December 16, 2020). [Online]. Available: https://ethstats.net/