Verifiable External Blockchain Calls: Towards Removing Oracle Input Intermediaries

Joshua Ellul[1,2][0000−0002−4796−5665] and Gordon J. Pace[1,2][0000−0003−0743−6272]

[1] Centre for DLT, University of Malta, Malta
[2] Department of Computer Science, University of Malta, Malta
{joshua.ellul,gordon.pace}@um.edu.mt

Abstract. It is widely accepted that blockchain and other distributed ledgers cannot initiate requests for input from external systems and are reliant on oracles to provide such inputs. This belief is founded on the fact that each node has to reach a deterministic state. In this paper we show that this belief is a preconceived one by demonstrating a method that supports calls to external systems initiated from the blockchain itself.

Keywords: External calls · Oracle input · Blockchain architecture.

1 Introduction

Many have argued that decentralisation is a cure to many woes arising from issues of trust. By removing centralised points-of-trust, one can build solutions which empower participants. Blockchain and other distributed ledger technologies (DLTs) allow for the decentralisation of computational systems and services built on top of them. Whilst there is truth to such statements, the real world lies outside the blockchain, and although data and algorithms residing on the blockchain can be decentralised, any reference to the real world must necessarily break through the event-horizon of the blockchain and interact with the outside world — much of which is centralised out of physical or regulatory necessity. For instance, if one needs to access the temperature at a particular location at a particular time, one must interact with the real world and trust that the correct information has been provided.

Blockchain systems have traditionally addressed these issues through the use of oracles — channels providing information from the outside world into the blockchain. However, the nature of public blockchains allows only for a one way flow of information (from external entities into the blockchain) and any attempt to do this in the opposite direction (i.e. invoke an external entity from within the blockchain) causes problems due to the nature of consensus of such systems. The only alternative solutions available require trusted entities to perform such invocations, which simply delegates the problem one step away.

Blockchain systems require that the decentralised logic encoded within them reaches a deterministic state. It is often said that every node must execute the exact same logic in order to achieve consensus — and for this reason, it is the

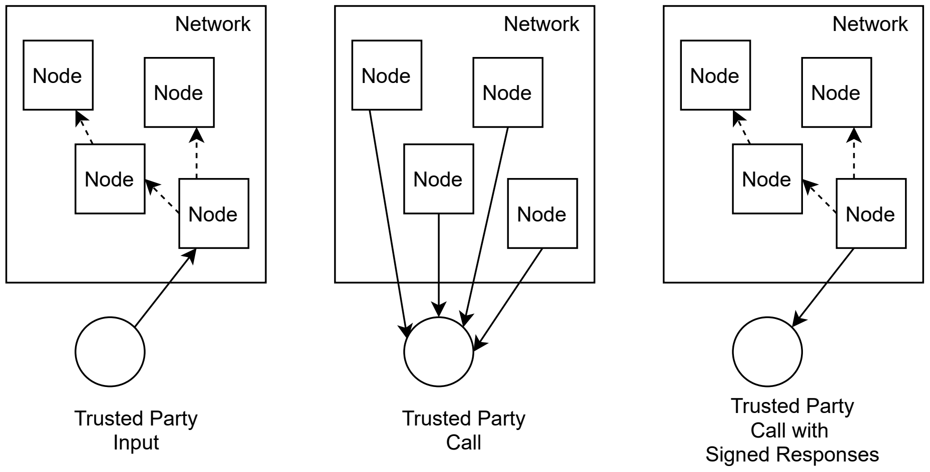

Fig. 1. Left: traditional trusted party input; middle: active calls requiring each node to undertake the external call that must return the same input; right: external calls enabled with verifiable signed responses.

general consensus in the community that Blockchain and DLT systems cannot make calls to external systems/oracles [14, 3, 12, 16, 1, 5, 9, 15, 7, 2, 8, 17, 6, 10]. However, we believe that the general consensus on this matter is not well-founded and is preconceived. Perhaps based upon the often cited statement that deterministic computation is required [13] — yet whilst this statement is true, it is important to highlight that it is the state that computation reaches that must be deterministic, and the computation performed can reach such a deterministic state in different ways.

In this paper, we present initial work on a technique that allows for the interaction with external parties directly in a feasible manner. Figure 1 provides an overview of an oracle input transaction/call flow for: (i) traditional oracle input (on the left); (ii) (inefficient) external calls requiring responses to always be the same — which does not scale up (in the middle); and (iii) the solution proposed herein which makes use of verifiable external calls (on the right).

We have implemented a prototype demonstrating the technique described in Section 2 and further present initial gas performance evaluation in Section 3. Initial thoughts, motivation and related work have been discussed in [4].

2 Design and Implementation

2.1 Verifiable External Calls

The solution proposed herein is to make use of verifiable external calls — i.e. a request (call) made to an external system that returns back a signed response which: (i) can be verified to truly be a response from the external party in question; (ii) which does not require any further communication (with the external party or other). This can be achieved in the same way how we provide such assurances in traditional applications and how trusted oracle input is verified, by checking whether the response was indeed digitally signed [11] by the external party. Knowledge of the trusted party's public key is required to be known (in the same way that oracle input requires knowledge of the trusted party's address) or can be retrieved from a trusted entity.

To allow for processes to make direct use of external services (in a feasible and efficient manner), which do not require explicit integration from the external parties themselves (with the specific platform), we propose to make use of verifiable external calls which provide a guarantee with respect to the veracity of the origin of the response both at the time of processing as well as for any point in future for which such verification may be required.

A verifiable external call is defined as the following tuple — a request, a public key and a signed response structured as follows:

$$\langle \mathit{request},\ \mathit{public\_key},\ \mathit{signed}(\mathit{response})\rangle$$

The *request* should point to the external system/service endpoint which is to be called (though this is an implementation design decision), and may also comprise of other input data. The *public_key* may be hard-coded into the application logic (e.g. into the smart contract), or it could even be retrieved by a trusted certificate provider. In either case it would need to be recorded by the time when the external call is executed — it will be used to verify the response originated from the respective external party. The *signed*(*response*) is the response that has been signed using the external party's private key (which is associated with *public_key*.

Indeed, this does require that the trusted data sources provide an end-point that responds back with a signed response which would likely require changes to existing data sources to implement signed responses — however, recent proposals indicate that such a standard may eventually be adopted[3], which if adopted would enable for this approach to integrate with all data sources (that are keeping up with standards).

Furthermore, to avoid old signed responses from being repeated, an incremental number, timestamp, block number or another challenge-response could be made use of which would ensure old responses cannot be repeated — however this is left as an implementation detail. Whilst, the challenge data sent to the external party will be part of the *request*, the verifiable external call's definition may be extended to include the challenge-response. For example, the request can be augmented by a request number to a fresh nonce ν, which is expected to be included unchanged in the response[4]:

$$\langle \mathit{request} \oplus \{\mathit{nonce} \mapsto \nu\},\ \mathit{public\_key},\ \mathit{signed}(\mathit{response} \oplus \{\mathit{nonce} \mapsto \nu\})\rangle$$

[3] `https://wicg.github.io/webpackage/draft-yasskin-http-origin-signed-responses.html`

[4] We use $\oplus$ to represent function overloading.

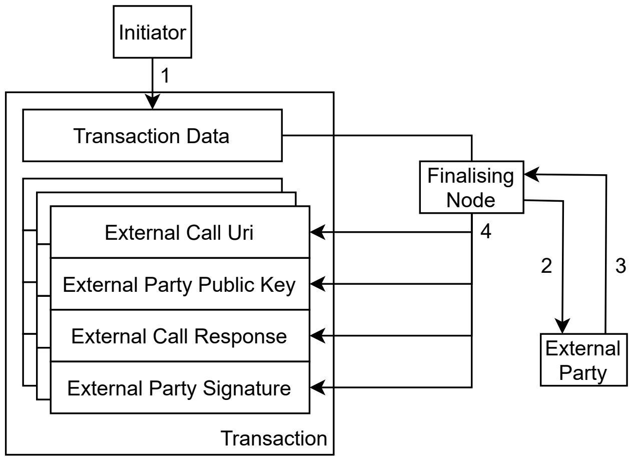

Fig. 2. Transaction finalisation process.

2.2 Transactions

When a transaction is initiated (be it by a user, another system, or the system itself if such a DLT allows this) and accepted for execution, the node which is processing the transaction will establish all external calls which need to be performed, execute them and record the responses received back from the trusted external parties along with associated digital signatures. Indeed, at this point the finalising node must ensure that the response is from the trusted party by verifying the response and signature against the trusted party's public key. Furthermore, if a unique number, date/time, or challenge-response mechanism was used to ensure old data is not repeated, then this would also be validated at this point. A depiction of how a transaction is initiated and attributed with the various data associated with external calls is depicted in Figure 2.

For responses that are not verified, associated transactions may be deemed to have failed, or depending upon reparation logic, the transaction may still be valid. This is a design decision that each platform would need to consider. The same goes for external calls for which no response is received.

To reiterate, to ensure that a finalising node does not repeat old responses from external parties, the response and signature could be accompanied with the date and time the response was generated and/or a unique response identifier associated with the response (and potentially request as well). One challenge is to ensure that participating nodes indeed execute such external calls rather than simply record failure, which we will delve into in a future paper (since it merits its own paper). However, verifiable external calls could also be undertaken by the transaction initiator (i.e. the party submitting the transaction provides this information as part of the transaction submission process) — however indeed this depends upon the architectural design of the blockchain, smart contract and wallet/dApp software submitting the transaction. By performing the verifiable external call at transaction submission time (on the initiator), the aforementioned problem pertaining to nodes potentially reporting back failed external calls would be eliminated.

2.3 Implementation Details

The Go Ethereum (geth)[5] node implementation (version 1.16.5) was modified to include support for the verifiable external call mechanism described above. The following salient modifications were implemented. A prototype of the approach has been implemented and available from `https://github.com/joshuaellul/excalls`.

EXCALL Transaction A new type of transaction, an EXCALL transaction (in `excall_tx.go`), was added (on top of the existing Legacy and Access List transactions) to facilitate storing the additional data associated with external calls (described in Section 2.1) in an EXCALL tuple — containing the external call response, signature, and the external party's known public key.

`EXCALL` instruction A new `EXCALL` virtual machine opcode which instructs the virtual machine to execute the external call was added. Rather than modify the whole programming tool-chain (including the Solidity programming language and Solidity compiler) to support the proposed `EXCALL` instruction, for the purpose of this prototype it was emulated by replacing `PUSH32` instructions (used for string assignments) whose associated data starts with "http" into `EXCALL` ones.[6]

A miner executes the emulated `EXCALL` instruction only when finalising a block, and will undertake an external call to the URL specified as a parameter to the instruction (pushed on the stack via the aforementioned `PUSH32` instruction). Upon receiving a response and a valid digital signature for the respective public key, the relevant data will be appended to an EXCALL transaction.

Following this, the transaction is stored in the block with the EXCALL transaction data filled in (as depicted in Figure 2). This then allows for other nodes to verify the external call based upon the stored data (without having to initiate an external call itself).

3 Evaluation

A gambling dApp is used to serve the purpose of a required use-case to evaluate gas performance of the proposed approach against a traditional approach. The evaluation discussed below would also apply to other smart contract use-cases

[5] `https://github.com/ethereum/go-ethereum`

[6] Indeed, this means that in the prototype it is not possible to make use of a `PUSH32` instruction for data that starts with the string "http", however this does not impact the prototype's purpose to evaluate the proposed technique.

that have similar protocol requirements where a party must first initiate a transaction to a smart contract prior to external oracle data being made available on the blockchain (which typically is due to not wanting to reveal that data prior to the initiating transaction).

The use-case requires that these steps are followed to complete a betting transaction: (i) a user initiates interaction with the smart contract by placing a bet; (ii) data from the oracle is retrieved and fed into the smart contract to determine whether the user won. The use-case has been implemented in Solidity and available from `https://github.com/joshuaellul/excalls` for: (i) a standard Ethereum network that makes use of an external oracle to feed in data; and (ii) a modified Ethereum implementation which supports external calls to directly fetch the oracle input.

Comparing gas costs associated with the two approaches, the standard approach requires 67,599 and 48,222 gas units to execute `beginBetOracle` and `continueBetOracle` respectively. The total gas cost for the standard approach, $standard_{gas}$, amounts to 115,821. Whilst, the total gas computed for the external call approach, $computed\_excall_{gas}$, is 89,071 — however, this does not include additional gas associated with actually undertaking the external call.

A gas cost associated with an external call would need to be decided upon for the respective blockchain system. It is not the scope of this work to decide upon an exact value, yet we can make an estimate by breaking down the external call process into: (i) the actual external call undertaken only on the node adding the associated block; and (ii) verifying the external call response which takes place on every node. Based on this the total gas consumption for the approach can be defined as:

$$total\_excall_{gas} = computed\_excall_{gas} + excall_{gas} + verify\_sig_{gas}$$

where $excall_{gas}$ is the gas associated with making an external call; and $verify\_sig_{gas}$ is the gas associated with verifying an external call's response (i.e. verifying an ECDSA signature).

The cost to verify an ECDSA signature ($verify\_sig_{gas}$) was evaluated to be 3,903 gas[7]. Therefore, the total gas required for the external call approach is:

$$total\_excall_{gas} = 89,071 + excall_{gas} + 3,903$$

If the gas costs of an external call approach is equal to or less than a traditional approach, then the external call approach will not be introducing any negative consequences with respect to gas. Therefore, we can identify an upper-bound limit for $excall_{gas}$ to be:

$$standard_{gas} - (computed\_excall_{gas} + verify\_sig_{gas})$$

[7] Code from `https://solidity-by-example.org/signature/` to verify an ECDSA signature was executed in order to retrieve gas costs. The cost of the verification only was calculated by first executing a function call and then adding in a call to verify a signature, and the difference between the two was used to calculate the signature verification gas cost.

This results in 22,847 gas — which given that the costs of $excall_{gas}$ are only incurred on the node that is adding a block, this amount should be more than justifiable. Based on this, we claim that the external call approach proposed herein should consume equivalent or less gas to that required for the standard approach. However, we leave a full investigation to define appropriate gas costs for such an operation for future work — which may include evaluating differing gas costs according to HTTP Request and Response payload sizes.

4 Conclusions

It is a widely accepted belief that blockchain systems cannot execute calls to external systems due to the requirement for computation to reach a deterministic state. It is often said that blockchain-based computation needs to be deterministic [13], however it is important to highlight that determinism of output can be achieved in different ways. Contrary to the general consensus, in this paper, we have demonstrated a method for Blockchain and DLT systems that allows for direct external calls to be initiated from the Blockchain/DLT itself. We have implemented a prototype and undertaken initial evaluation of gas overheads of a typical dApp requiring external oracle input.

This is only initial work in this direction for which we believe will pave the way for extensive future work in the following directions: (i) investigating novel consensus protocols that better support external calls; (ii) language design for smart contract external calls; (iii) development of novel blockchains that support external calls; (iv) development of novel dApps supported through active external calls; (v) miner/validator incentive mechanisms for external calls; (vi) further performance and evaluation of active external call techniques.

A prototype demonstrating verifiable external calls has been implemented and available from `https://github.com/joshuaellul/excalls`.

References

1. Adler, J., Berryhill, R., Veneris, A., Poulos, Z., Veira, N., Kastania, A.: Astraea: A decentralized blockchain oracle. In: 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. 1145–1152. IEEE (2018)
2. Caldarelli, G.: Real-world blockchain applications under the lens of the oracle problem. a systematic literature review. In: 2020 IEEE International Conference on Technology Management, Operations and Decisions (ICTMOD). pp. 1–6 (2020). https://doi.org/10.1109/ICTMOD49425.2020.9380598
3. Ellis, S., Juels, A., Nazarov, S.: Chainlink: A decentralized oracle network.(2017). White paper (2017)
4. Ellul, J., Pace, G.J.: Towards external calls for blockchain and distributed ledger technology. arXiv preprint arXiv:2105.10399 (2021)
5. Gatteschi, V., Lamberti, F., Demartini, C., Pranteda, C., Santamaría, V.: To blockchain or not to blockchain: That is the question. IT Professional **20**(2), 62–74 (2018). https://doi.org/10.1109/MITP.2018.021921652

6. Lin, S.Y., Zhang, L., Li, J., Ji, L.l., Sun, Y.: A survey of application research based on blockchain smart contract. Wireless Networks **28**(2), 635–690 (2022)
7. Liu, X., Muhammad, K., Lloret, J., Chen, Y.W., Yuan, S.M.: Elastic and cost-effective data carrier architecture for smart contract in blockchain. Future Generation Computer Systems **100**, 590–599 (2019). https://doi.org/https://doi.org/10.1016/j.future.2019.05.042, `https://www.sciencedirect.com/science/article/pii/S0167739X18328334`
8. Marchesi, L., Marchesi, M., Tonelli, R.: Abcde—agile block chain dapp engineering. Blockchain: Research and Applications **1**(1), 100002 (2020). https://doi.org/https://doi.org/10.1016/j.bcra.2020.100002, `https://www.sciencedirect.com/science/article/pii/S2096720920300026`
9. Marchesi, M., Marchesi, L., Tonelli, R.: An agile software engineering method to design blockchain applications. In: Proceedings of the 14th Central and Eastern European Software Engineering Conference Russia. pp. 1–8 (2018)
10. Mekouar, L., Iraqi, Y., Damaj, I., Naous, T.: A survey on blockchain-based recommender systems: Integration architecture and taxonomy. Computer Communications **187**, 1–19 (2022)
11. Merkle, R.C.: A certified digital signature. In: Conference on the Theory and Application of Cryptology. pp. 218–238. Springer (1989)
12. Rimba, P., Tran, A.B., Weber, I., Staples, M., Ponomarev, A., Xu, X.: Comparing blockchain and cloud services for business process execution. In: 2017 IEEE International Conference on Software Architecture, ICSA 2017, Gothenburg, Sweden, April 3-7, 2017. pp. 257–260. IEEE Computer Society (2017). https://doi.org/10.1109/ICSA.2017.44, `https://doi.org/10.1109/ICSA.2017.44`
13. Sankar, L.S., Sindhu, M., Sethumadhavan, M.: Survey of consensus protocols on blockchain applications. In: 2017 4th International Conference on Advanced Computing and Communication Systems (ICACCS). pp. 1–5. IEEE (2017)
14. Weber, I., Xu, X., Riveret, R., Governatori, G., Ponomarev, A., Mendling, J.: Untrusted business process monitoring and execution using blockchain. In: Rosa, M.L., Loos, P., Pastor, O. (eds.) Business Process Management - 14th International Conference, BPM 2016, Rio de Janeiro, Brazil, September 18-22, 2016. Proceedings. Lecture Notes in Computer Science, vol. 9850, pp. 329–347. Springer (2016). https://doi.org/10.1007/978-3-319-45348-4_19, `https://doi.org/10.1007/978-3-319-45348-4\_19`
15. Xu, X., Pautasso, C., Zhu, L., Lu, Q., Weber, I.: A pattern collection for blockchain-based applications. In: Proceedings of the 23rd European Conference on Pattern Languages of Programs, EuroPLoP 2018, Irsee, Germany, July 04-08, 2018. pp. 3:1–3:20. ACM (2018). https://doi.org/10.1145/3282308.3282312, `https://doi.org/10.1145/3282308.3282312`
16. Xu, X., Weber, I., Staples, M., Zhu, L., Bosch, J., Bass, L., Pautasso, C., Rimba, P.: A taxonomy of blockchain-based systems for architecture design. In: 2017 IEEE International Conference on Software Architecture, ICSA 2017, Gothenburg, Sweden, April 3-7, 2017. pp. 243–252. IEEE Computer Society (2017). https://doi.org/10.1109/ICSA.2017.33, `https://doi.org/10.1109/ICSA.2017.33`
17. Zhao, Y., Kang, X., Li, T., Chu, C.K., Wang, H.: Towards trustworthy defi oracles: Past, present and future. IEEE Access (2022)