Towards Cross-Blockchain Smart Contracts

Marten Sigwart*, Philipp Frauenthaler*, Christof Spanring†, Stefan Schulte*

* Distributed Systems Group
TU Wien, Vienna, Austria
{m.sigwart, p.frauenthaler,
s.schulte}@dsg.tuwien.ac.at

† Pantos GmbH
Vienna, Austria
contact@pantos.io

***Abstract*—Today, the development of cross-blockchain applications involves a lot of complexity regarding the underlying cross-blockchain communication. In an ideal world, developers are able to completely focus on writing application logic instead of dealing with passing data between blockchains.**

In this white paper, we present a framework enabling cross-blockchain smart contract invocations between Ethereum-based blockchains. With just two lines of code, developers can invoke smart contracts deployed on other blockchains, e.g., they can deploy a contract on the Ethereum blockchain that calls smart contracts on the Ethereum Classic blockchain.

I. Introduction

The Token Atomic Swap Technology (TAST) research project[1] investigates possible means of interconnecting multiple blockchains [2]. In particular, we explore different kinds of cross-blockchain applications and protocols, e.g., cross-blockchain token transfers [11, 12], or atomic swaps [9]. Typically, such applications involve various invocations of smart contracts deployed on different blockchains. That is, some action on blockchain A triggers another action on blockchain B.

Today, when devising cross-blockchain applications and protocols, developers usually have to deal with a lot of complexity regarding the underlying cross-blockchain communication. As different blockchains cannot natively communicate with each other [4], developers have to rely on off-chain clients passing information between blockchains. Ideally, off-chain clients are able to participate at any time without having to request permission from a centralized authority [4]. While this preserves the decentralized nature of blockchains, developers have to account for malicious clients joining and corrupting the system [7].

Furthermore, off-chain clients passing information from one blockchain to another incur cost, e.g., transaction fees. To compensate off-chain clients for their services, developers need to come up with an incentive structure. The incentives must reward honest and penalize dishonest behavior.

Hence, before being able to focus on writing the application logic, developers have to solve a variety of challenging tasks. As developers typically come up with a specific solution for their applications, various systems for cross-blockchain communication exist with many of them basically providing the same service [1, 3, 6, 12]. As operating such systems may be expensive [7], it may be more profitable to operate a generic system that is utilized by many different applications.

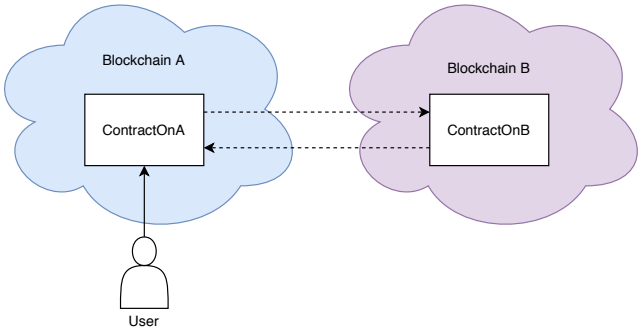

Figure 1: Interaction between two contracts residing on two different blockchains.

Hence, a generic framework is desirable that enables smart contracts on some blockchain A to communicate with contracts on some other blockchain B and vice versa. Ideally, developers can call smart contracts deployed on other blockchains the same way they would call contracts residing on the same blockchain without taking care of the underlying cross-blockchain communication. In the next section, we present a framework for such cross-blockchain smart contract invocations.

II. Cross-Blockchain Smart Contracts

As mentioned in Section I, a framework facilitating cross-blockchain smart contracts enables a contract on some blockchain A to invoke smart contracts on some other blockchain B and vice versa. Figure 1 shows the basic interaction of a cross-blockchain smart contract call. A user calls ContractOnA which initiates a call of ContractOnB (i.e., the *remote* contract). After the execution of ContractOnB, the result is reported back to ContractOnA for further processing. As a first step, we provide a proof of concept implementation of the framework for Ethereum-based blockchains which is available as open-source software on GitHub[2]. In the following, we discuss the inner workings of the framework.

A. Architecture

The framework is based on Remote Procedure Calls (RPCs). RPCs follow the request-response paradigm. A client initiates

White Paper, TU Wien; May 2020, version 1.0.

[1] http://www.dsg.tuwien.ac.at/projects/tast/

[2] https://github.com/pantos-io/x-chain-smartcontracts

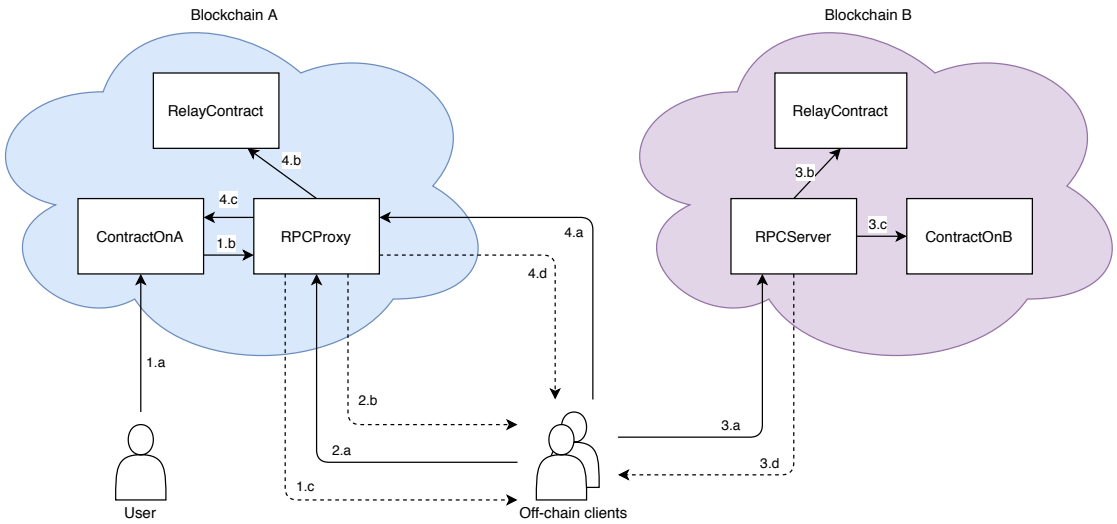

Figure 2: Interaction between two contracts residing on two different blockchains.

an RPC by sending a request to a known remote server. The server executes the specified procedure and returns a response to the client where the application continues to process.

Similarly, our framework consists of two main components provided as smart contracts: RPCProxy and RPCServer, two smart contracts that are deployed on the calling blockchain and the remote blockchain, respectively. Consider Figure 2. ContractOnA wants to call some function of ContractOnB. ContractOnA initiates the function call by calling contract RPCProxy which also resides on blockchain A. The RPCProxy contract encodes the request and sends it via a set of off-chain clients to a specialized contract on blockchain B, the RPCServer. The RPCServer decodes the request and calls the specified function of ContractOnB (the remote contract). After execution, the response is provided back to the RPCProxy on blockchain A which forwards the result back to ContractOnA (the calling contract).

As mentioned above, the actual cross-blockchain communication is handled by a set of off-chain clients since blockchains cannot natively communicate with each other [4]. The call requests and responses are encoded as Ethereum transactions and the off-chain clients just relay these transactions between the involved blockchains. For instance, the transaction that initiated the call request from ContractOnA is relayed to the RPCServer on blockchain B, which routes the request to the requested smart contract (i.e., ContractOnB).

Of course, ContractOnB should only execute the call, if the request is valid. That is, if the transaction containing the request is actually included and confirmed on blockchain A. For that, a further component is required, a so-called blockchain relay [7]. Blockchain relays provide the ability for one blockchain to verify whether certain transactions are included and confirmed on some other blockchain without relying on trust in a single party. As such, before forwarding the call request to ContractOnB, the RPCServer calls the respective relay to check that the request is indeed valid. Similarly, the RPCProxy calls a relay to verify that the provided response is a valid response that was returned by ContractOnB (i.e., the respective transaction is included and confirmed in blockchain B). Note that we treat the relay contract as a black box. Given a transaction, the relay contract provides an answer, *true* or *false*, whether the transaction is included and confirmed on the other blockchain or not. Details of how blockchain relays operate under the hood can be found in [7].

B. Behind the Scenes

With the different components of the framework introduced, this section discusses what happens behind the scenes when a remote contract is invoked.

1) Cross-chain Call Preparation: Consider Figure 2 again. Some user calls a function of ContractOnA on blockchain A which contains a cross-blockchain call, i.e., a call of some function of ContractOnB on blockchain B (Step 1.a). ContractOnA "prepares" the cross-blockchain call by calling the RPCProxy (Step 1.b). When ContractOnA makes a cross-chain call using the RPCProxy, the RPCProxy emits a *Call-Prepared* event which is recognized by the off-chain clients (Step 1.c).

2) Cross-chain Call Request: We need to make sure that cross-chain calls are only successful if they are initiated by the correct RPCProxy contract. In Ethereum, transactions only contain the address of the contract that has been invoked first. If this contract calls other contracts as part of its execution, these subsequent calls are not encoded in the transaction. Hence, the RPCServer would not be able to know whether the correct RPCProxy prepared the call request when parsing transactions from blockchain A as the transactions only contain ContractOnA's address and not the address of the RPCProxy. The RPCServer contract would have no way of knowing whether ContractOnA subsequently called the right RPCProxy or just a fake one. Thus, two further steps are necessary.

After the off-chain client is informed about a pending cross-chain call via the CallPrepared event on blockchain A, it can create a cross-chain call request by directly calling the RPCProxy (Step 2.a). This time, the address of RPCProxy is encoded in the corresponding transaction. Thus, when the transaction is forwarded to the RPCServer on blockchain B, the RPCServer can verify that it was the correct RPCProxy which requested the call and not a fake one. If this verification is successful, the RPCServer can trust the encoded call request data. Otherwise, it would be possible to invoke contracts on blockchain B which have never been called by some contract on blockchain A.

When the off-chain client calls the RPCProxy, it emits a *CallRequested* event (Step 2.b). This indicates that the call can now be relayed to the RPCServer on blockchain B.

3) Cross-chain Call Execution: Again, an off-chain client receives the CallRequested event. It now relays the call to the RPCServer on blockchain B by forwarding the transaction encoding the contract call (Step 3.a). When the RPCServer receives the transaction, the following two steps are executed.

Before the call encoded in the transaction can be executed, RPCServer needs to check that the received transaction is actually included and confirmed in blockchain A. For that, it forwards the transaction to the RelayContract (Step 3.b). Further, the RPCServer needs to verify that the received transaction called the correct RPCProxy contract on blockchain A to be sure that the call request can be trusted.

After the RPCServer is sure that the call request actually exists on blockchain A (RelayContract returned *true*), it can now forward the call to ContractOnB as indicated by the request (Step 3.c). After the call is executed by ContractOnB, the RPCServer emits a *CallExecuted* event encoding the result value from the call execution. When an off-chain client recognizes the CallExecuted event, it knows that the call request has been executed (Step 3.d).

4) Cross-chain Call Acknowledgment: After recognizing the event, the off-chain client forwards the transaction encoding the response to the RPCProxy on blockchain A (Step 4.a). Before the call response can be extracted from the transaction and returned to ContractOnA (call acknowledgment), the RPCProxy needs to check that the received transaction is included and confirmed in blockchain B. For that, it forwards the transaction to the RelayContract on blockchain A (Step 4.b). Further, RPCProxy verifies that the received transaction was submitted to the correct RPCServer on blockchain B to prevent fake responses.

After the RPCProxy is sure that the call response can be trusted (corresponding transaction is confirmed on blockchain B and correct RPCServer contract was called on B), it can now execute the acknowledgment (step 4.c). The RPCProxy looks up the stored callback function for the call and calls this function of ContractOnA. After the callback is executed by ContractOnA, the RPCProxy contract emits a *CallAcknowledged* event. When the off-chain clients receive the CallAcknowledged event, they know that the remote contract call has been completed.

C. Calling a Remote Contract

The provided framework hides the complexity involved in making cross-blockchain contract calls. Developers of applications do not have to take care of the underlying cross-blockchain communication anymore. Instead, they can concentrate on the application logic.

Listing 1 shows how a Solidity smart contract calls a remote contract deployed on another blockchain. The function of ContractOnB is called *remoteFunction*, takes two parameters and returns an unsigned integer. Lines 5-6 show how to call the function from within ContractOnA. The function signature is packed together with the parameter values into a byte array and passed via the *callContract* function to the RPCProxy. Along with the call data, the caller has to pass the address of the remote contract (*contractOnB*), a call identifier, and the name of the callback function ("callback") that will receive the call acknowledgment. The callback function takes the call identifier, the result of the remote call, and a boolean indicating whether the call has been successful or not as arguments.

As you can see, the developer only needs to know about the RPCProxy and can call a remote contract in just two lines of code. The details of the underlying cross-blockchain communication remain hidden.

III. Discussion & Future Work

The presented framework for cross-blockchain smart contracts can be beneficial for developers implementing cross-blockchain applications as the framework hides a lot of complexity involved in building cross-blockchain applications such as the underlying means of cross-blockchain communication, encoding and decoding of transactions, and security aspects. The framework in its current state can be considered a proof of concept. To provide a fully-fledged solution, several aspects still have to be explored in further detail. We discuss these aspects in the following subsections.

A. Incentive Structure

The framework relies on a set of off-chain clients to continuously monitor the RPCProxy and RPCServer for new events to transfer the necessary transactions between blockchains. Submitting transactions causes cost which off-chain clients are currently not compensated for. An incentive structure is necessary to encourage participation. Furthermore, incentives should be aligned so that honest behavior is rewarded while malicious behavior is penalized.

Such an incentive structure likely involves callers of cross-blockchain smart contracts to pay a small fee that is then paid out to the off-chain clients that handle the underlying cross-blockchain communication. The fee can be paid out in the native currency of the invoking blockchain or in a currency that exists on multiple blockchains contemporaneously, e.g., [5]. Furthermore, off-chain clients might have to provide a stake before being able to participate. In case malicious behaviour by an off-chain client is detected, its corresponding stake is

Listing 1: Calling a remote contract in Solidity

```
contract ContractOnA {
  ...

  function callRemoteFunction(address param1, uint param2) public {
    bytes memory callData = abi.encodeWithSignature("remoteFunction(address,uint256)", param1, param2);
    rpcProxy.callContract(contractOnB, callIdentifier, callData, "callback");
  }

  function callback(uint callIdentifier, bytes memory result, bool success) public {
    ...
  }
}

contract ContractOnB {
  ...

  function remoteFunction(address param1, uint param2) public returns (uint) {
    ...
  }
}
```

slashed or rewarded to the client that detected the wrongdoing. This way, off-chain clients are disincentivized to behave dishonestly and incentivized to report malicious activities.

B. Support for Reads

In its current form, the framework supports cross-blockchain invocations of functions that alter the state of the remote contract. Calling functions that merely read a value from the remote contract and provide it back to the calling contract is only supported by means of wrappers where reading the value is wrapped in a function that alters the remote contract's state in some way, e.g., by emitting an event. Hence, calling remote functions that only read a value causes some overhead for the framework as the caller has to make sure a corresponding wrapper exists. In future work, it needs to be examined if and how read functions can be natively supported by the framework.

C. Synchronous Calls

At the moment, the framework provides *asynchronous* cross-blockchain smart contract invocations. When the remote contract is called, its result is not immediately provided back to the calling contract. Instead the result of the remote call is provided back to the calling contract later on via a callback function. In this paradigm, cross-blockchain application logic is spread out over a series of calling and callback functions where the next remote call is triggered in the previous call's callback function and so on. Depending on the number of remote calls, this can lead to a deeply nested and complex call structure.

In future work, it may be of interest to explore the possibility of providing synchronous remote calls where the result of the remote call is provided back to the calling function immediately, eliminating the need for a callback function. While first attempts at synchronous cross-blockchain smart contract invocations exist, so far they are restricted to permissioned blockchains and require modifications of the blockchain core [10]. However, modifications of the core of a public, permissionless blockchain are not easily implemented as each change has to be accepted and approved by the blockchain's community. Therefore, it remains unclear whether synchronous remote calls are possible in public blockchains without requiring modifications of the involved blockchains' core implementations.

D. Nested Calls

Similarly to synchronous calls, the proof of concept only provides limited support for nested calls, i.e., a remote contract call that itself calls a remote contract. When the callback for a remote call is invoked it does not contain any information about the success of subsequent remote calls. This creates a challenge for smart contracts that rely on the results of subsequent remote calls.

In future work, we will investigate whether it is possible to guarantee that before the callback of the first remote call is invoked, all subsequent remote calls were executed as well.

E. Integration of further Blockchains

The proof of concept presented in this work enables cross-blockchain smart contracts for Ethereum-based blockchains. The given framework, for instance, provides the ability for a smart contract on the Ethereum blockchain to call smart contracts deployed on the Ethereum Classic blockchain. In future work, other blockchains should be analyzed and integrated. The existing framework will likely have to be adjusted to account for the intricacies of the other blockchains. For instance, Ethereum-based blockchains require the encoding of the remote call request in two separate transactions due to the fact that an Ethereum transaction does not contain the complete call chain but only the first contract that is invoked by the transaction. If a blockchain provides the complete call chain, these two transactions can be merged.

IV. Conclusion

The approach presented in this paper provides cross-blockchain smart contract invocations for Ethereum-based

blockchains. For instance, the framework enables smart contracts on Ethereum to invoke contracts deployed on Ethereum Classic and vice versa. By taking away the complexity of the underlying cross-blockchain communication such as encoding, decoding, security, and data transfer between blockchains, developers can focus completely on writing application logic. As we have shown, developers only need to write two lines of code to initiate a cross-blockchain smart contract call. Apart from reducing complexity, the proposed approach may also make blockchain interoperability economically more viable as the cost of maintaining such an interoperability platform can be spread across all applications relying on it.

Disclaimer

Information provided in this paper is the result of research, partly based on publicly available resources of varying quality. Popular use of cryptocurrencies includes investment and speculation on price developments of currencies and assets. The goal of this paper is to describe technical aspects relevant for the TAST research project. Economic considerations or future price developments are therefore not discussed. Technologies are described from a purely technical point of view. Therefore, the information in this paper is provided for general information purposes only and is not intended to provide advice, information, predictions, or recommendations for any investment. We do not accept any responsibility and expressly disclaim liability with respect to reliance on information or opinions published in this paper and from actions taken or not taken on the basis of its contents.

Acknowledgment

The work presented in this paper has received funding from Pantos GmbH within the TAST research project.

References

[1] Autonomous Software. *Metronome: Owner's Manual*. Version 0.99 (Last Updated 2019-08-15). Accessed 2020-07-31. 2018. URL: https://github.com/autonomoussoftware/documentation/blob/master/owners_manual/owners_manual.md.

[2] M. Borkowski et al. *Cross Blockchain Technologies: Review, State of the Art, and Outlook*. 2019. URL: http://dsg.tuwien.ac.at/projects/tast/pub/tast-white-paper-4.pdf. White Paper, Technische Universität Wien. Version 1.0. Accessed 2020-02-06.

[3] M. Borkowski et al. "DeXTT: Deterministic Cross-Blockchain Token Transfers". In: *IEEE Access* 7 (2019), pp. 111030–111042.

[4] V. Buterin. *Chain Interoperability*. https://allquantor.at/blockchainbib/pdf/vitalik2016chain.pdf. Accessed 2020-02-10.

[5] P. Frauenthaler et al. *Advanced Cross-Chain Tokens Transfers*. 2020. URL: https://www.dsg.tuwien.ac.at/projects/tast/pub/tast-white-paper-9.pdf. White Paper, Technische Universität Wien. Version 1.0. Accessed 2020-07-29.

[6] P. Frauenthaler et al. *Leveraging Blockchain Relays for Cross-Chain Token Transfers*. 2020. URL: https://www.dsg.tuwien.ac.at/projects/tast/pub/tast-white-paper-8.pdf. White Paper, Technische Universität Wien. Version 1.0. Accessed 2020-04-20.

[7] P. Frauenthaler et al. *Testimonium: A Cost-Efficient Blockchain Relay*. 2020. URL: https://arxiv.org/abs/2002.12837.

[8] J. Heiss et al. "From oracles to trustworthy data on-chaining systems". In: *2019 IEEE International Conference on Blockchain (Blockchain)*. IEEE. 2019, pp. 496–503.

[9] M. Herlihy. "Atomic cross-chain swaps". In: *Proceedings of the 2018 ACM symposium on Principles of Distributed Computing*. 2018, pp. 245–254.

[10] P. Robinson et al. "Atomic Crosschain Transactions for Ethereum Private Sidechains". In: (2019). URL: https://arxiv.org/abs/1904.12079.

[11] M. Sigwart et al. *Decentralized Cross-Blockchain Asset Transfers*. 2020. URL: https://arxiv.org/abs/2004.10488.

[12] A. Zamyatin et al. "XCLAIM: Trustless, Interoperable, Cryptocurrency-Backed Assets". In: *IEEE Symposium on Security and Privacy (S&P)*. 2019, pp. 193–210.