Towards private Active Choreographies on public blockchain

Henry Bergstroem[1] and Jan Mensch[1,2]

[1] Hasso-Plattner-Institut, Prof.-Dr.-Helmert-Strae 2-3, 14482 Potsdam, Germany
`bergstroem@uni-potsdam.de`
[2] University of Potsdam, Am Neuen Palais 10, 14469 Potsdam, Germany
`jan.mensch@uni-potsdam.de`

Abstract. Integrating business processes has been found to have a positive impact on operational and business performance [8, 13]. However if business partners have no trust in each other, integrating processes is challenging [15]. In this paper we present our approach of working towards a realization of Active Choreographies that is in no need of trust and can enforce a business process, while still implementing strong visibility constraints that would usually be seen in real-world applications. These constraints keep the business logic, the identity of a messages originator and the content of their communication secret. This is done while still being able to record an encrypted audit trail, where messages are only readable by parties that are allowed to read them. We also make a prototype available, which showcases our proposed schema.

1 Introduction

Integrating business processes has been found to have a positive impact on operational and business performance [8, 13]. However a lack of trust between business partners is hindering such progress [15]. By proposing a solution that needs neither trust between participants, nor trust in any central authority and keeps the business done secret, we hope to further increase collaboration between businesses. As shown by Weber et al. in [18], there are ways to extend the use cases of business process execution and auditing to an untrusted environment, thus making it possible for entities to collaborate with each other that would usually not be willing to do so. This execution of untrusted business processes is made possible by using an blockchain-based implementation of Active Choreographies, which we will briefly discuss in Section 2.1.[3]

The goal of this paper is to explore the notions of privacy and visibility with regards to untrusted execution of choreographies on the blockchain and to introduce a schema which aims to keep untrusted business process execution private, while still being enforceable. The structure of the paper is as follows: Introduction (Section 1). Background (Section 2), where we will give a short overview of the work we considered when creating our solution. Approach (Section 3).

[3] Phrasing and sources of this paragraph where in part taken from [18]

A brief explanation of the architecture of the proposed solution. Implementation (Section 4) and the discussion (Section 5). A summary over limitations, alternative implementations and future work. Conclusion (Section 6).

2 Background

2.1 Blockchain-Based choreographies

The following section provides a short introduction to blockchain-based choreographies.

Active Mediators. In [18] Weber et al. explain how a business process can be enforced in an environment where the participants do not trust each other. By using a smart contract[4], acting as an active mediator between participants. The term active mediator implies that the component is able to accept or reject messages and makes sure that the business process is executed as specified. With the help of this and other components they are able to (I) execute collaborative processes over a network of untrusted nodes, (II) enforce that only conforming messages can trigger state changes, (III) payments and escrows can be coded in the business process and (IV) an immutable audit trail keeps track of all transaction. Furthermore none of the participants in the business process has to organize the collaboration, which is what differentiates a choreography from a orchestration. The approach introduced in [18] is different from ours in the sense that in their implementation the business process is included in the smart contract, which makes it public. It displays which participants are collaborating with each other, how they are doing it and what messages are exchanged. We aim to provide privacy and try to hide the logic of the business process, who the originator of a message is and what messages are exchanged, while still providing that (I), (II) and (IV) hold.

Active Choreographies. In [11] Ladleif builds on the work of Weber et al. and creates an extension of the Business Process Model and Notation (BPMN) 2.0 choreography diagrams in combination with Ethereum. The introduced addition to the BPMN standard is capable of modeling events, scripts and data, which are vital to the modeling of smart contract driven choreographies. This extension is called Active Choreographies. Like Weber et al., Ladleif uses smart contracts as a central link between the collaborating parties. Messages are not exchanged directly between the entities, but through the smart contract. This schema allows for business process execution without the need of trust between the participating businesses.

In this paper we work towards realizing Active Choreographies, which are in no need of trust and can enforce a specified business logic. Our addition to the work of Weber et al. and Ladleif is that we additionally aim to provide strong visibility constraints. We are therefore working towards private Active Choreographies on public Blockchain infrastructure.

[4] A program which is being executed on a blockchain

2.2 Visibility levels

While the issue of privacy was mentioned by Weber et al. and Ladleif, they did not take any measures to actually implement them. However, Latleif did introduce different levels of visibility in the context of modeling business contracts with ACs [11]. The levels are as follows:

- *Model Level*: The logic of the business process. If you would model a business process using BPMN, the BPMN model would be the model layer. If you would implement it using a programming language, the source code would be the model layer.
- *Communication Level*: The knowledge of which messages were exchanged, who exchanged them and which entities are part of the business process. Latleif also includes the time at which messages were send. For the scope of this publication, we would like to exclude the time of an exchanged message from the communication level.
- *Content Level*: The content of the exchanged messages.

We are referring to these levels in Section 3.1 when discussing the assumptions that our approach is based.

2.3 Privacy Enhancing Technologies

Privacy Enhancing Technologies on the Blockchain. Our implementation aims to provide privacy for all three layers. We considered multiple other technologies that are worth mentioning and could be a viable alternative to our approach.

One of them is *obfuscation*. This is a technology that "aims to make a computer program 'unintelligible' while preserving its functionality" [9]. Since it was shown in [5, 6] that there is a set of functions that are impossible to obfuscate, the idea of indistinguishable obfuscation was introduced. This is a weaker constraint, guaranteeing that it is impossible to distinguish two equivalent programs of a similar size that have been obfuscated. These ideas seem to be very powerful, but come with a big drawback: Being extremely computationally expensive [4].

Homomorphic encryption might be used to do calculations on encrypted values. Similar to obfusaction, homomorphic encryption suffers from heavy computational overhead [10].

A further technology considered is *secure multi-party computation (SMPC)* [14]. With SMPC, there are are multiple parties involved in computing the output of a function. These parties do not have to trust each other. Therefore they need a protocol that makes sure that the output is correct and that cheating participants will not be able to learn anything about the input of the honest parties.

A technology already in use in blockchain applications are *zero-knowledge Succinct Non-interactive ARguments of Knowledge (zk-SNARKs)*. An application using it is Zerocoin, which aims to provide "fully anonymous currency transactions" [12]. zk-SNARKS provide the possibility to have an untrusted prover verify a statement (or secret) made by a different party, without that party having to reveal their secret [7]. It furthermore does so in a non-interactive manner.

Today there exist already multiple projects that focus on business process execution (e.g. Hyperledger Grid [1]). Some of these implementations provide mechanisms aiming to create trust between participants. One of them is *parity ethereum*. With parity, users are able to hide private contracts in public contracts and to exchange private messages [3].

3 Approach

3.1 Assumptions and Scenario

Before introducing our schema, we would like to mention the assumptions on which our proposal is based on. Each assumption is marked with a letter for later reference.

Assumptions about the participating entities: a: there are several parties which want to collaborate. These parties have b neither trust in each other nor c trust in any third party. They furthermore d want to keep their business processes secret, e hide with whom they are collaborating and f hide what is being communicated during the collaboration. Since the parties distrust each other they also g only want to share messages with the entities that have to see them and keep them secret from the others. Each party h is "honest-but-curious"[5]. These assumptions justify that we aim to keep all levels mentioned in Section 2.2 private. To be more specific, d requires a private *Model Level*, e a private *Communication Level* and f a private *Content Level*.

Further assumptions: i: all data that is processed on the blockchain is considered public. It is j possible to convert a business process into a program or state-machine in order for parties to determine the validity of a state change. The process of converting a business process into a program or state-machine is out of the scope of this paper. k: in terms of trust, a public blockchain is superior to a private one, since it does not require trust in the organisation that runs the network. l: a network that employs "policing nodes"[6] is considered inferior to a network without such nodes, since such a network requires "trust in the authorities". Communication between nodes is done m on a message based system where the metadata of a message does not contain the address of the message sender.

Throughout this paper we will try to present our thoughts in more concrete fashion by applying them to the example shown in Figure 1. Here the parties

[5] Definition honest-but-curious: "The honest-but-curious (HBC) adversary is a legitimate participant in a communication protocol who will not deviate from the defined protocol but will attempt to learn all possible information from legitimately received messages." [16]

[6] Nodes that have the special role of enforcing rules.

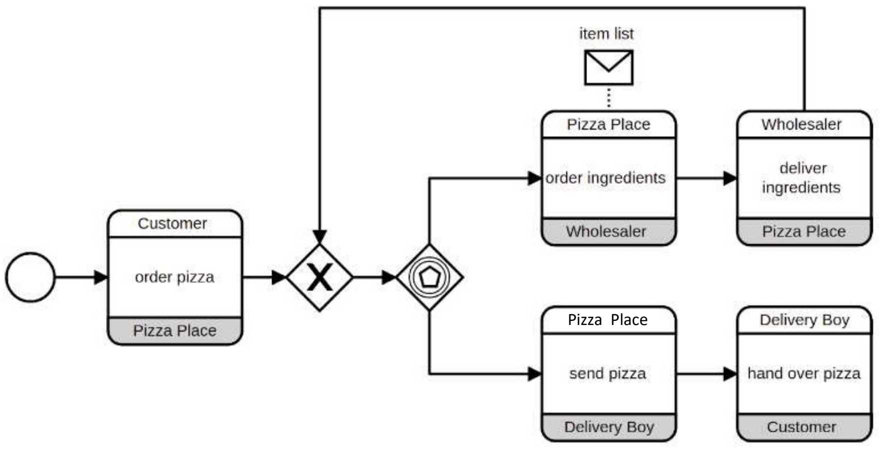

Fig. 1. Choreography diagram of a pizza ordering process

are collaborating without trust (a, b and c). They also have an urge to keep as much as possible about their collaboration secret (d, e and f). An example of g and h would be that the *Delivery Boy* (D) should not know anything about the collaboration between the *Customer* (C) and the *Pizza Place* (P), even though D is part of the overall business process.

3.2 Proposed Schema

In this chapter we discuss our proposed schema. We will do so by introducing mechanisms which each solve one or more of the assumptions of Section 3.1.

Circles. In order to solve assumption f, g and (in part) h, we introduce the notion of circles. Circles are visibility constraints for a group of entities. A message exchanged within a circle can only be read by other participants of a circle. We achieve this by using symmetric keys which are exchanged between the members of a circle before the execution of the process. This can be done by using well established processes like the session key exchange in openPGP [2]. Which parties are within which circle should be established by the parties themselves and is therefore out of scope of this paper. Figure 2 shows an example of possible circles of the process defined in Figure 1. Each circle has its own associated symmetric key and each message is encrypted with the respective key. This way C cannot read the communication between the *Wholesaler* (W) and P, because this communication is encrypted with sym_2 and C is only in possession of sym_1. Also the message can still be included in the audit trail, since only members that are in possession of sym_2 can read it.[7]

[7] Credit: key-icons in Figure 2 made by Yannick (http://yanlu.de) from www.flaticon.com

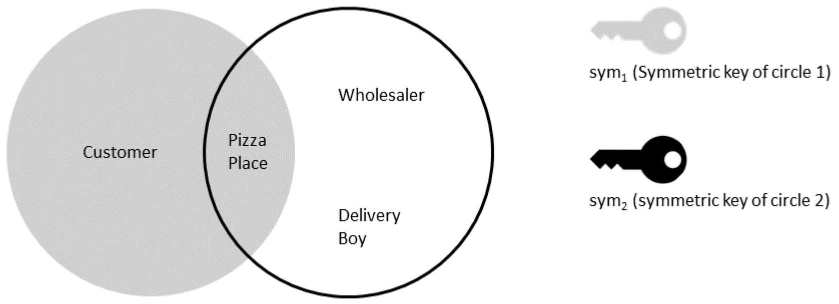
Customer
Pizza Place
Wholesaler
Delivery Boy
sym$_1$ (Symmetric key of circle 1)
sym$_2$ (symmetric key of circle 2)

Fig. 2. Circles for of the process described in Figure 2. The circles indicate who is communicating with whom and who should see the content of the exchanged communication.

Protecting the Model Layer. Since we consider all communication and all code execution on the blockchain as public (assumption i), we decided to store and execute the logic off-chain. This is in order to keep the *Model Layer* secret (assumption d). The representation of the business process is shared amongst all entities. This could be a graph or executable code (assumption j). This representation should vary in detail, hiding the details of the process that are out of the own circle. How this representation looks like exactly and how it is shared is out of scope of this paper.

Consensus mechanism. Since we store the logic of the business process off-chain, a smart contract has no way of knowing if a proposed state change is valid or not. This is why we came up with a simple consensus mechanism for the nodes and created a sc that is able to reject messages that did not result out of this consensus mechanism. If both of these things are given a smart contract is able to enforce a business process without being aware of its logic.

In our opinion a business process can be viewed as a state-machine. In it, states are actions that are taken by the entities (e.g. "send pizza") and state changes are triggered by messages (e.g. "W is now hands over 1kg of flour, ordered by P" changes the state from "order ingredients" to "deliver ingredients"). If the state machine is automated on each client, invalid messages can be rejected and the business process enforced.[8]

Figure 3 shows such a mechanism[9]. In it all nodes are placed in a ring[10]. In this example communication is done counterclockwise. The order in which communication is done is fix. Just the start- and endpoint vary. As already

[8] We will briefly discuss limitations of this though in Section 5.

[9] contract-icon in the Figure 3 made by Freepik (http://www.freepik.com) from www.flaticon.com

[10] not to be confused with a ring-signature [17]

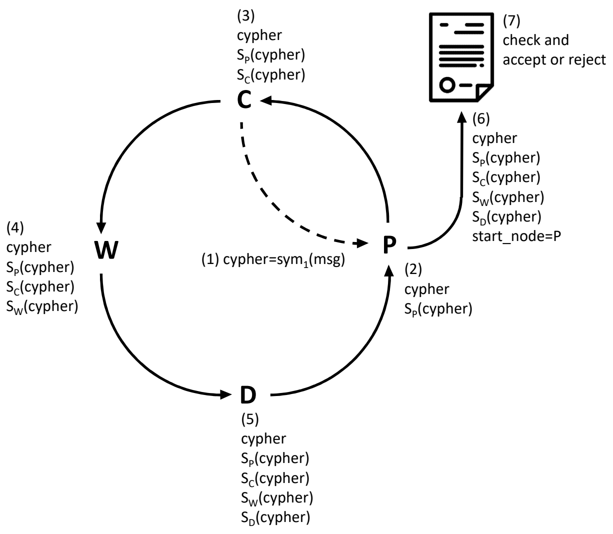

Fig. 3. The proposed schema. 1: initial sending of cypher. 2-5: other nodes give consent. 6: cypher and signatures are send to smart contract. 7: validation

discussed, all nodes are in possession of the symmetric keys of their associated circles. In addition to that they also own a asynchronous key-pair of which their public key is published on the smart contract.

It is displayed how the *Customer (C)* would like to propose a state change (e.g. from the "start-state" to "order pizza"). For that, C creates the message *msg* (e.g. "C: order pizza margarita for 8 EUR."). This message is encrypted using the symmetric key sym_1 of his circle. The encrypted message is denoted as *cypher*. As a first step, C is then sending *cypher* to a random node, e.g. P (we explain the necessity of this step in in the next paragraph). Since P happens to be in the same circle as C, he can read *cypher* by decrypting it with sym_1.

P can now signal his consent by applying a hash function to *cypher* and then encrypting this hashed value with his private key. The combination of these functions for the signing process is denoted as S_X. X is the name of respective node. A node that is not in the circle does not receive any new information since it is not able to read the message. It is also unable to know who the originator of the message is, since the originator sends it to a random node to start the procedure. If this was not the case and C would send *cypher* the next node in line (D), D would know that C is the message sender, since the order of nodes is known to all entities. We do this to counter assumption b and h and with the premise that assumption m holds.

If all nodes "agreed"[11] to *cypher*, *cypher* and the signatures will reach P again. Since P has already seen *cypher* he will send it to the smart contract for

[11] We write "agreed" since nodes outside the circle cannot read the message and simply pass it on.

validation, together with all signatures and the information that he was the first node to add a signature (*start_node*=P). If the validation passes, the nodes will know that they reached a new state in their business process, but only C and P will know what state this is.

3.3 Validation

The smart contract receives *cypher*, a list of signatures and the information that P started the signing process. The sc can now decrypt each signature using the public key of each of node in the circle. If all encrypted signatures match the hash of *cypher* the state change is valid. *cypher* is then included in the audit trail and all clients know that a new state was reached, but only C and P will know which state this is, since only they are in possession of sym_1. This way an audit trail is kept in which messages can only be read by entities that are supposed to read them.

4 Implementation

In order to evaluate our thoughts, we build a simplified version of the schema discussed in Section 3. To implement the clients we used Python 3.6. To implement the smart contract, we used Solidity running on a local Ethereum Blockchain on Ganache. Our implementation is able to simulate a ring with three nodes. If all nodes confirm the encrypted message by signing it, the encrypted message and the signatures are send to the smart contract for validation. Out of the three clients, two are in the same circle. This is to showcase the visibility constraints of circles.

So far we did not implement the automatic accept- or reject-mechanism that is necessary to enforce a business process. With the current prototype a user has to accept or reject the proposed state change manually.

Also topic of future work is the algorithm deciding to which random node an encrypted message should be send to at the first step of the schema. With the current implementation the client just sends it to the next node in line. We avoided the the topic of choosing a random node, because in our opinion this should be done not by the message originator, but by the entire group. This is to avoid the danger that a node might choose a different node not at random but with the intention to somehow cheat on the system. Since distributed algorithms are difficult to develop and this is just a minor detail of the schema, we decided to implement it as future work.

5 Discussion

5.1 Limitations of the proposed Schema

There are a number of shortcomings in the approach we introduces in Section 3.2. In this chapter we would like to name the ones that we noticed.

Collusion of all entities within a circle. The goal of our approach was to enforce a business process, while keeping the *Model Layer*, the *Communication Layer* and the *Content Layer* secret. We try to achieve this by automatically rejecting illegal state changes. If an entity encounters a message proposing an impossible state change it should reject it (at least if it is able to read that message). A possible misuse of this system is that all entities within a circle collaborate and pass invalid state changes. It is debatable though, if this behaviour should be prevented. Messages passed within a circle should be only relevant for the parties within that circle. If the deviation from the defined business process is agreed upon by all parties within the circle, this would be equivalent of renegotiating the terms of the collaboration. For example, C might sends the message "send pizza margarita for 6 EUR", even thought the price might have been defined as 8 EUR in the business process. If P now accepts it, they just renegotiated their terms. The way the business process is enforced is therefore rather soft.

Knowledge of progress within the consensus mechanism. During the consensus mechanism every entity is adding their signature to the message which is passed around. Because the sc has to know all public keys in order to verify messages, the number of participants is known to all entities. Every node who receives this message will therefore be aware of how important their consent it (premise h). This knowledge might be abused and should be avoided. Solving this issue is the topic of future work.

Blocking progress. Any node can decide to not sign any more messages and block the progressing of the business process. We hope that this issue will not occur very often, because the nodes that are participating in the business process should have an interest in the successful execution of that progress (assumption a).

Powerful nodes. Depending on the business process and the participating parties, it might happen that there is an party that accumulates a lot of knowledge. In our case this is P, who can see all messages exchanged, because he is a member in both circles. This may or may not be seen as a problem by the other participants. This issue should be taken into account when designing the visibility constraints of the business process.

Spam. If assumption m holds and the nodes indeed do not know who the originator of a message is, this opens the door for spam messages. An outsider could send a message, encrypted with an unknown key to a member of the ring. That member would pass the message along, assuming that he can not read it because he is not part of the same circle as the originator of the message. So would all other nodes in the ring and the message would be varified by the sc and stored on the blockchain. Depending on which blockchain is used, storing data on it might be expensive. An outsider might therefore be able to increase the price of the business process execution. This should be avoided! A possible solution to this might be to have all nodes create a second key-pair and to only exchange those public keys within the ring. These public keys might then be used to create a ring signature [17], which would prove that the originator of a message is indeed part of the ring. The "original" public keys could not be used for this task, since they are encoded in the sc and are therefore public knowledge (assumption *i*). The exploration of this idea is the topic of future work.

5.2 Alternative approaches

Some of the technologies mentioned in Section 2.3 might be used to provide privacy. For example zk-SNARKs might be used to prove that a state change is legal. If that would be possible, we might be able to design a protocol where the communication overhead between the nodes could be drastically reduced.

We were only able to read about a few of the many blockchain-implementations that take on the issue of privacy. The one implementation that caught our attention was parity ethereum, which is why we mentioned it in Section 2.3. As already discussed, users are able to create private contracts and to exchange private messages. However one difference between our approach and parity is that parity needs specialized nodes in order to execute private contracts. These nodes are called validators, which are "account[s] that can allow a private contracts state change" [3]. In our opinion this trust in specialized nodes makes parity unfit for untrusted business process execution (assumption *l*).

6 Conclusion

In this paper we showed that it is possible to hide the logic of the business process, the communication and the content of the messages. This was achieved on public blockchain infrastructure to avoid the trust issues that are, in our opinion, inherent to private blockchains. We furthermore showed that a business process can be enforced without the smart contract being aware of its logic. Since this work is proprietary, there are multiple limitations to it. For some of these limitations solutions were proposed. We hope that this paper might be a first step towards private active choreographies on public blockchain.

References

1. Hyperledger grid. https://www.hyperledger.org/projects/grid, accessed: 2019-01-30
2. Openpgp message format - 2.1 confidentiality via encryption. https://tools.ietf.org/html/rfc4880section-2.1, accessed: 2019-01-30
3. Private transactions - wiki. https://wiki.parity.io/Private-Transactions, accessed: 2019-01-19
4. Banescu, S., Ochoa, M., Kunze, N., Pretschner, A.: Idea: Benchmarking indistinguishability obfuscation–a candidate implementation. In: International Symposium on Engineering Secure Software and Systems. pp. 149–156. Springer (2015)
5. Barak, B., Goldreich, O., Impagliazzo, R., Rudich, S., Sahai, A., Vadhan, S., Yang, K.: On the (im) possibility of obfuscating programs. In: Annual International Cryptology Conference. pp. 1–18. Springer (2001)

6. Barak, B., Goldreich, O., Impagliazzo, R., Rudich, S., Sahai, A., Vadhan, S., Yang, K.: On the (im) possibility of obfuscating programs. Journal of the ACM (JACM) **59**(2), 6 (2012)
7. Ben-Sasson, E., Chiesa, A., Genkin, D., Tromer, E., Virza, M.: Snarks for c: Verifying program executions succinctly and in zero knowledge. In: Advances in Cryptology–CRYPTO 2013, pp. 90–108. Springer (2013)
8. Flynn, B.B., Huo, B., Zhao, X.: The impact of supply chain integration on performance: A contingency and configuration approach. Journal of operations management **28**(1), 58–71 (2010)
9. Garg, S., Gentry, C., Halevi, S., Raykova, M., Sahai, A., Waters, B.: Candidate indistinguishability obfuscation and functional encryption for all circuits. SIAM Journal on Computing **45**(3), 882–929 (2016)
10. Gentry, C.: Computing arbitrary functions of encrypted data. Communications of the ACM **53**(3), 97–105 (2010)
11. Ladleif, J.: Active Choreographies. Master's thesis, Hasso Plattner Institut, Prof.-Dr.-Helmert-Strae 2-3, 14482 Potsdam, Germany (2018)
12. Miers, I., Garman, C., Green, M., Rubin, A.D.: Zerocoin: Anonymous distributed e-cash from bitcoin. In: Security and Privacy (SP), 2013 IEEE Symposium on. pp. 397–411. IEEE (2013)
13. Narayanan, S., Jayaraman, V., Luo, Y., Swaminathan, J.M.: The antecedents of process integration in business process outsourcing and its effect on firm performance. Journal of Operations Management **29**(1-2), 3–16 (2011)
14. Orlandi, C.: Is multiparty computation any good in practice? In: Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on. pp. 5848–5851. IEEE (2011)
15. Panayides, P.M., Lun, Y.V.: The impact of trust on innovativeness and supply chain performance. International Journal of Production Economics **122**(1), 35–46 (2009)
16. Paverd, A., Martin, A., Brown, I.: Modelling and automatically analysing privacy properties for honest-but-curious adversaries. Tech. Rep. (2014)
17. Rivest, R.L., Shamir, A., Tauman, Y.: How to leak a secret. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 552–565. Springer (2001)
18. Weber, I., Xu, X., Riveret, R., Governatori, G., Ponomarev, A., Mendling, J.: Untrusted business process monitoring and execution using blockchain. In: International Conference on Business Process Management. pp. 329–347. Springer (2016)