Towards private Active Choreographies on public blockchain

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

[1] Hasso-Plattner-Institut, Prof.-Dr.-Helmert-Straße 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 was found to increase the performance of businesses. This collaboration is difficult if there is a lack of trust between the participating entities and between the entities and a third party. Active Choreographies (ACs) solve this problem, by making business processes enforceable. However they require that the business process and all exchanged messages are public. In this paper we provide an alternative approach towards Active Choreographies that keeps the business logic, the exchanged messages and the identity of the entities exchanging messages secret, while still being enforceable. The schema can also be used to function as an encrypted audit trail. In addition to providing the conceptual background, we also make a prototype available, which executes a simple, predefined business process.

Keywords: Business Processes · Active Choreographies · Privacy.

1 Introduction

Blockchain [12], invented by Saitoshi Nakamoto as a mean to solve the problem of double-spending in crypto-currencies, is a chain of digitally signed blocks of data. To sign a block, the hash of a block is calculated by applying a hash-function to the data of the block and the hash of the previous block. Nodes in the network have to agree on which block to include next, which is why there has to be a consensus-mechanism (e.g. proof of work). Because removing or altering blocks is very computationally expensive, there is no central authority needed to watch over the validity of submitted data. This is why blockchains can be used to serve as distributed, immutable data stores.

These capabilities (storing data immutably and having consent on a single state without the need of a central authority) are greatly increased by networks like Ethereum. The technology enables developers to write a smart contract (sc), a Turing complete program that is executed on the blockchain [7].

In our opinion, smart contracts are ideal for business process execution, since the entities collaborating with each other now have a single source of truth. Also, since the stored data is immutable, blockchains are feasible to serve as audit trails [**source?**].

As shown by Weber et al. in [19], 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. The execution of untrusted business processes is made possible by using Active Choreographies, which we will briefly discuss in section 2.2.

The goal of this paper is to extend the work of Weber et al.: We aim to make it possible to execute business processes in an untrusted environment and to enforce a business process without revealing its structure or purpose. The suggested schema keeps secret what was communicated during the process execution and who was communicating, while still making sure that the predefined procedure is being followed by all participants. The schema also records an immutable audit trail that is only visible by entities that have a right to see it.

2 Background

This section discusses the motivation, related work and the background which we took into account when designing our solution.

2.1 Motivation

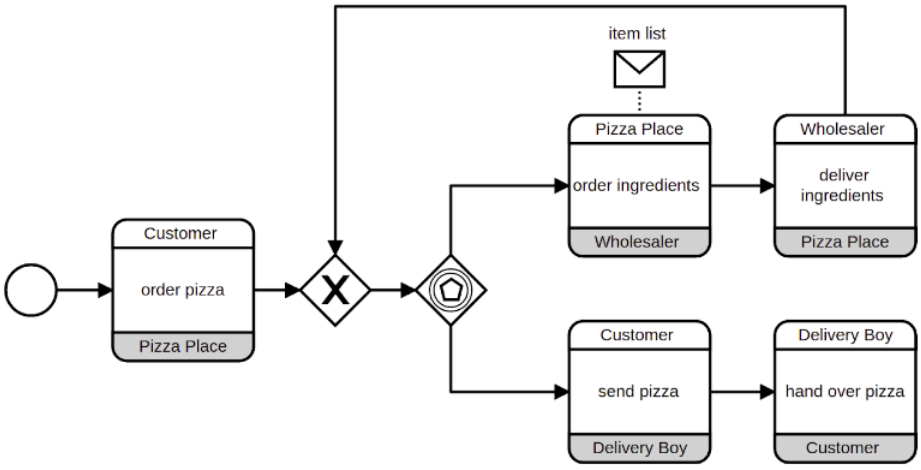

Fig. 1. A simple choreography diagram

Integrating business processes has been found to have a positive impact on the operational and business performance [8, 13]. However a lack of trust between business partners is hindering such progress [16]. By providing a solution

that needs neither trust between participants, nor in any central authority and keeps the business done secret, we hope to further increase collaboration between businesses.[3]

mention business process mining? [2]

Our motivation can be summarized in the following statements: (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 (d) furthermore want to keep their business process secret, (e) hide with whom they are collaborating and (f) hide what messages are exchanged 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.

Throughout this publication we will try to present our thoughts in more concrete fashion by applying them to the example shown in figure 1. Here the parties are (a) collaborating without trust ((b) and (c)). They also have an urge to keep as much as possible about their collaboration secret ((d), (e) and (f)). An example for (g) would be that the *Delivery Boy* should not know anything about the collaboration between *Wholesaler* and *Pizza Place*, even though he is part of the overall business process.

2.2 Related work

Visibility levels.

Our aim is to restrict the visibility of ACs and thus provide privacy for the participating entities in an untrusted business process. To structure our approach we would like to subdivide the visibility of ACs, an approach introduced by Ladleif in [10].

- **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.
- **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.

Active Choreography. In [19] 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, 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

[3] Phrasing and sources inspired by [19]

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 [19] 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. There is no point in trying to encrypt messages that trigger a state-change, since these messages have to be processed with the smart contract and all calculations done on the smart contract are public. We aim to change this and try to hide what the business process, who is participating in it and what messages are exchanged, while still providing that (I), (II) and (IV) hold. **Explain why (III) does not hold?**

For the scope of this publication we define an Active Choreography as a decentralized collaboration between nodes where no node has to orchestrate the process and the choreography is set up in a way that invalid state-changes can be rejected, thus making sure that the predefined procedure is always followed.

Privacy Enhancing Technologies on the Blockchain. Our implementation aims to provide privacy for all three levels. We considered multiple other technologies that are worth mentioning and might be used to protect one or more levels.

One of them is *obfuscation*. This is a technology that "aims to make a computer program 'unintelligible' while preserving its functionality" [9]. If we could make use of it, it might be possible to hide the functionality of our business process on a public blockchain in "plain sight" (model level). Since it was shown in [4, 5] that there are 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 have been obfuscated. These ideas seem to be very powerful, but come with a big drawback: Being extremely computationally expensive [3].

Homomorphic encryption might be used to do calculations on encrypted values **source**. This might be useful in the context of processing communication with the sc, without having to reveal the actual messages (content level). Similar to obfusaction, homomorphic encryption suffers from heavy computational overhead **source**.

A further technology considered is *secure multi-party computation (SMPC)* [15]. With SMPC, there are are multiple parties involved in computing the output of a function. These parties do not have to trust each other. They thus 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. An algorithm with these properties could provide us with a way to compute the current state of our business logic without "broadcasting" the input that triggered that state (content level).

A technology already in use in blockchain applications are zero-knowledge Succinct Non-interactive ARguments of Knowledge (zk-SNARKs). An application using this is Zerocoin, which aims to provide "fully anonymous currency transactions" [11]. zk-SNARKS provide the possibility to have an untrusted prover verify a statement made by a different party, without that party having to reveal their statement [6]. It furthermore does so in a non-interactive manner. If we think of our business model as a state-machine, we might use zk-SNARKs to prove that a state-change is valid without actually communicating the respective message(s) (message- and content level).

Also already applied in the context of blockchain applications (e.g. in Monero [14]) are *Ring Signatures*. This is a signature created by multiple signers. It can be used to verify that a message was send by one of the signers, without revealing who in the group was the sender. For this the sender "does not need the knowledge, consent, or assistance" of anyone else in the group [18]. If applied in the similar way as in Monero, Ring Signatures might be useful to protect the identity of the collaborating entities(communication level) and possibility the exchanged messages (content level).

An attentive reader might wonder why we are not suggesting to use a private blockchain. An application where it might be possible to penalize cheating participants or only include honest entities in the network (**sources needed?**). We try to avoid private networks, since in our opinion they are in need of of trust. Either in authorities within the network or in the organization(s) running the network. Executing untrusted business processes in a network in need of trust seems like a contradictory to us.

We would also like to mention *parity ethereum* [1]. With parity, users are able to hide private contracts in public contracts (model level) and to exchange private messages (content level, **communication level?**). However for private contract execution parity is relying on validators, "account[s] that can allow a private contract's state [to] change". In our understanding this makes parity insufficient for untrusted business execution, since, like private blockchains, it requires trust in one or multiple nodes.

3 out of scope things

I will at some point mention some things that are out of scope of this publication. How to exchange symmetric keys using asymmetric key encryption (compare OpenPGP and give broken down example of how that works. Mention that we did not implement it, to focus on the important stuff). How to agree on who should send it. We could also just have one node always send it to the chain. That would work, since an outsider would not know what is going on (no extra information, like who is the originator of the message). But since the participants do not trust each other using a random node to send the message is better. Selecting a random node can be done by using a distributed consensus protocol (like electing a leader in distributed databases. Just mention a possible algorithmn here).

4 some section

5 Approach

5.1 Assumptions

The following assumptions are made:

- the blockchain is public
- everyone can see what is posted on the blockchain
- an adversary does not track the traffic from individual participants from and to the blockchain. Therefore if the name of sender of a message is not stored in plain text on the chain, nobody knows who send the message.
- the participants of a business process (BP) do not trust each other, but would like to cooperate with each other
- it is possible to express any BP with a program and this program can enforce the corresponding business process
- it is possible for a program on the chain to notify a party (avoid busy-wait)
- This program can be viewed as a state-machine
- an entity that is part of a message exchange is allowed to see that message in the audit trail
- All nodes/entities are always online and node failure does not happen. This is unrealistic, but it is kept out of scope on purpose, to be able to focus on the privacy stuff.
- Clients are honest but curious: Everyone follows protocol, no one can forget anything they learned. Whatever info they can get they will get. Secure if a party only gets as much knowledge as they should (Defined in [20] [17])

5.2 Ideal solution

The ideal solution would look something like this:

- process uses a public blockchain
- A participant can enable a third trusted party to view all communication (audit trail) if (and only if!) one of the participants of that communication grants that third party permission to do so. Example: A is a supplier, B a seller, C the logistics guy, sending stuff from A to B, E the buyer, who buys stuff from B and F the logistics guy sending stuff from B to E. Something went wrong when ordering the supplies. A is unhappy with this and calls a lawyer. A can provide the lawyer with the audit trail of the communication he was involved in (e.g. between A and B, A and C and between A, B and C) but he can not give the lawyer any other communication, since he himself can not see that.
- parties agree on a process and that process has to be executed exactly as specified
- the parties only have to communicate via the blockchain
- execution of a BP is cheap (cost of gas on Ethereum)

– an observer, who read everything on the public blockchain that is used as a mediator between the parties can not know derive any conclusion about how the business logic looks like (logic), who send what messages when (message) and what messages where send (content).

5.3 Where should the logic of the business process be kept?

As mentioned before we try to enforce a business process (BP) with an Active Choreography while still keeping the content, communication and (in part) the model secret. Since we are in favor of using a public blockchain, we can consider all logic (model level) and all communication that is done with a smart contract running on such a blockchain as public. We could encrypt the messages exchanged between smart contact and the participants of the Business Process. This would keep the content and possibly the communication secret, but if we keep the logic of the BP public that would be of no use. A simple example to illustrate this point:

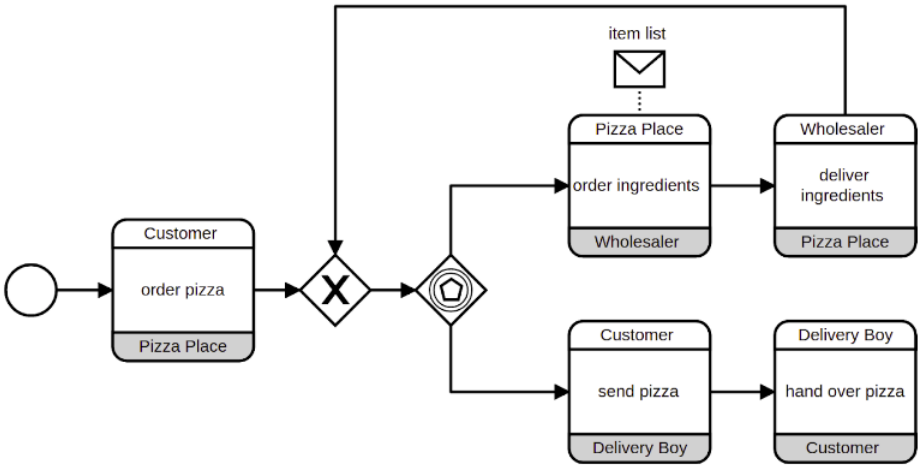

Fig. 2. A simple BPMN

If we implement the BP shown in 2 as an Active Choreography and upload it to a public blockchain there is no point in trying to keep any of the communication secret. What do we mean by that? Let's assume an observer tracks all communication done with the Active Choreography, but has no understanding what is being communicated and who is sending which messages. The observer does not need this information to be able to understand the communication. She can just watch the state-changes in the BP. If, for example, the state changes to *ordered pizza*, the observer knows that a customer just ordered a pizza. If it then changes to *order ingredients*, the observer understands that *Pizza Place*

does not have enough ingredients to process that order. Even if we try our best to keep content and communication secret, if we keep the state of the BP public, an outsider might be able to understand who is communicating and what is being communicated. The only thing we might be able to keep secret are mere details about the communication. For this reason we propose to keep the model off-chain.

5.4 Implementing an Active Choreography off-chain

An Active Choreography will make sure that a business process is executed as defined. Any attempt by a participant to evoke an illegal state-change will be rejected, since the business process knows what state-changes are legal and under what conditions they are carried out. This approach is no longer possible, if the Active Choreography is not aware of the business logic. Instead the clients would have to agree on what BP to execute and in which state they are. To keep a record of what business process is executed, the clients could share a hash of that BP (or rather the code representing it) and store it on the blockchain.

If a client now wants to execute a state-change in the BP, there should be a consensus among all clients that this state change is legal. If we use a "normal" consensus protocol this would not keep a client from cheating, since the choreography has no knowledge of a consensus reached between the clients. An example: *Customer* wants to order a Margarita pizza. For that he would like to do trigger a state change. For that he drafts the message *Order Margarita for 5 EUR*. If we have a consensus protocol, e.g. based on voting the other clients could now accept or reject that message and with it the state-change. Let's assume they accept it. *Customer* now sends the message *Order Margarita for 1 EUR* to the Active Choreography, which, with a client-based consensus protocol, has no way of rejecting this message as illegal.

To prevent this from happening a feasible consensus protocol might look like 3, with S_x being the secret key of each participant. In our example *customer* drafts the message *Order Margarita for 5 EUR* and encrypts it with his secret key S_c. The message is then send of to all other participants. Each of whom encrypt it with their secret key.

At the end the four times encrypted message reaches *customer* again. He than sends it to the Active Choreography together with the order they were encrypted in and the actual message. The message might look something like this *cipher, message, [C, D, W, P]*. The Active Choreography is in possession of all public keys. It applies them in order to the cipher and compares the result with the message. If they are the same and all public keys were applied to the message it knows that there was a consensus between all clients.

What happens in the case of a disagreement? Let's assume that the actual cost of the Margarita pizza is 6 EUR, not 5 EUR. The *pizza place* on receiving and encrypting $S_W(S_D(S_C(message)))$ will simply not encrypt it with its own private key. *Customer* has no way to still do a state-change, because the Active Choreography will reject the message, if it was not encrypted by all participants.

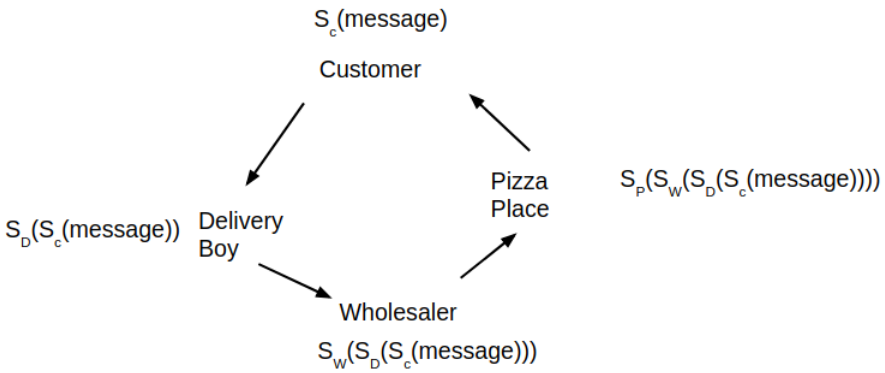

Fig. 3. The simple version of the consensus protocol

What happens if *customer* or any other participant tries to cheat? Customer will receive $S_P(S_W(S_D(S_C(message))))$ and can encrypt it. If he replaces *message* with something else, he will fail to encrypt it again, since he is not in possession of the other secret keys. The Active Choreography will therefore reject the message. The same is true for all other participants.

5.5 Advancing the idea of the simple consensus protocol

There are multiple drawbacks from the simple consensus protocol displayed in -3:

1. In order for the Active Choreography to be able to reject an invalid message, we have to provide it with the original message. We would like to avoid that to keep the content of the message secret (content level).
2. Every participant of the BP will know all messages exchanged, even messages which do not affect him.
3. An observer will always know who the originator of the message was, since it is always the first client in the provided order. In our example the order was *[C, D, W, P]*, which in this protocol, makes it obvious that this is a message from the owner of the C key-pair.
4. A client which goes rouge could always block further progress in the BP, by no longer encrypting messages.

The first and the second drawback can be by using symmetric keys to encrypt messages per "interest group". An interest group (we call them "circle") is the group of clients which are allowed to read a message, because they participate in the part of the business process that this particular message is relevant for. An example:

As shown in 4, *customer* is in a circle with *pizza place* and *delivery boy*. *wholesaler* is not in that group. This is because *wholesaler* should not know

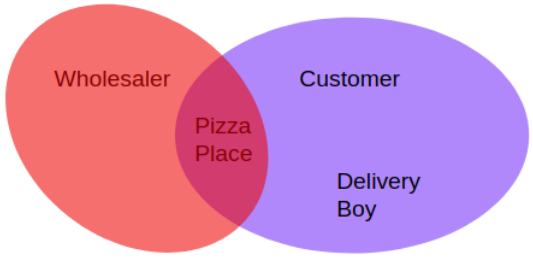

Fig. 4. The circles in the example BPMN

what e.g. *customer* and *delivery boy* are communicating. To enforce the circles, both groups will exchange symmetric keys, prior to the execution of the BP. This symmetric key exchange can be done safely, since they participants can exchange symmetric keys using asymmetric key encryption (similar to OpenPGP **citation needed!**). To exchange messages, the clients will now first encrypt them with the symmetric key of their circle and then send it on to the other clients (compare 5). A client which is in a circle will be able to encrypt the message. All other clients will not be able to read it. Like before one client from the group will have to send the encrypted message and the order of the applied keys to the Active Choreography. The example from above will now look as follows: *cipher1, cipher2, [C, D, W, P]*, with $cipher1 = S_P(S_W(S_D(S_C(cipher2))))$, $cipher2 = Sym_{blue}(message)$, $message = Order\ Margarita\ for\ 5\ EUR$ (Sym_{blue} is the symmetric key of the blue circle).

Like before the smart contract is now able to reject any message, which was send without the consent of all clients participating in the BP. Also, since a message has to be signed by all clients, not only by the clients of the relevant circle, an outsider will not be able to know who is collaborating with whom. The total number of circles, the size and the members of any circle will remain secret.

To solve the third drawback (observer is able to determine the originator of a message), we can propose the the following schema, depicted in 5. Among all clients there one client will be chosen, sending the above message for someone else. This participant could either be chosen randomly for each message or once for the entire BP. The former approach would require less trust among all clients, the latter would make it possible to only "register" one communicator at the Active Choreography and thus keeping secret who is participating in the BP (communication level). 5 shows how the example from above could look like in the complex version of the consensus protocol.

We assigned each client a color, depending on their circle membership. We assigned green to *wholesaler* to symbolize that this client will send the message to the chain, even though the *customer* is still the originator of the message. The

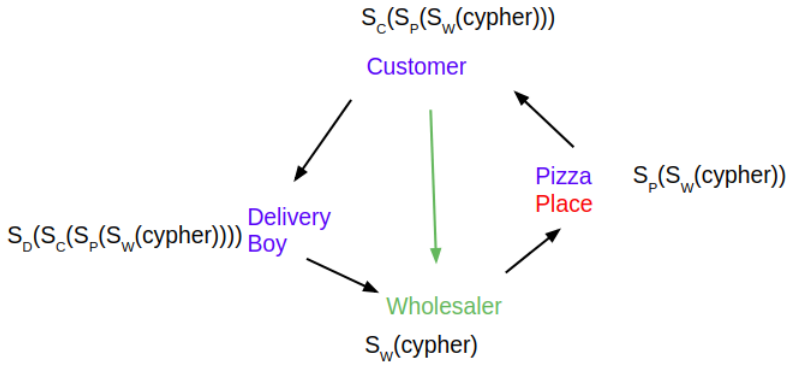

Fig. 5. The complex version of the consensus protocol

schema works as follows: The originator of the message (*customer*) will send cipher2 to the client responsible for communicating with the Active Choreography (*wholesaler*, exchange of cipher2 is the green arrow). That client will than send it on to the next client, who will send it on to the next... Like before all clients will encrypt it with their secret key. The message that *wholesaler* will send to the Active Choreography will be *cipher1, cipher2, [W, P, C, D]*, with *cipher1* $= S_D(S_C(S_P(S_W(cipher2))))$, $cipher2 = Sym_{blue}(message)$, *message = Order Margarita for 5 EUR.*

What happens if the client responsible to communicate with the Active Choreography tries to cheat? After all, if he is part of the same circle as the originator of the message, he can read cipher2. Still this should not matter, since all clients have to encrypt the message. If *wholesaler* cheats, *customer* will detect that and refuse to encrypt the message. Furthermore a message will always be rejected if not encrypted by all members of the BP.

6 Architecture Alternatives

We have discussed a range of different alternatives of how the architecture can look like. Each architecture has its advantages and disadvantages.

Where to put the code associated with the business logic? The blockchain acts like the source of truth for the actors. One main question is what is best to keep on the blockchain and what is better off. To maximize uniforminess between the actors it would make sense to have all logic on the blockchain. The obvious downside with the blockchain is that it is expensive to use and that everything is public. Depending on the business choreography and its priorities the better choice might be to leave parts of the business logic off-chain. The following table highlight the pros and cons about having the code on or off chain.

On-Chain	off-Chain
expensive	cheap
harder to scale	easier to scale
more transparent	more secret
enforceable	not directly enforceable

Table 1. Explain table here

How to make enforce an off-chain choreography If we decide to have the code code off-chain, to ensure that the business logic is kept secret, the question is how to ensure that a agreed upon business process will still be executed as specified. Since the smart contract in that case would only save the current state of the business process, it would have no way of knowing which state change would be legal or illegal to execute next.

An example: Alice A, Bob B and evil Eve E would like to participate in a business process. E intends to be cheating (meaning not executing the business process in the pre-defined order). All three parties agree on a business process that should be executed. This business process is compiled to executable code. To consolidate this, they save a hash of this code on-chain. All parties now start to execute the business process. There are therefore state-changes to the business process. This state changes are reflected on-chain, without any outsider being aware of the underlying business process. The smart contract thus acts as an interface between the clients and the blockchain, which here plays the role of an audit trail, not an Active Choreography. If E now sends an illegal state change to be recorded in the audit trail, the smart contract has no way of rejecting that, since it itself does not know if the proposed state change is legal or illegal.

We propose that there exists exactly one ring, which includes all participants (regardless of their circle). If a participant now wants to have a state-change confirmed he/she has it signed by all other participants in the ring. A participant who is in the same circle as the sender can read the message. If he she disagrees with the content of the message (the state-change) he can communicate this back to the sender and prevent the state change. Since the smart contract can check if the message was signed with the ring signature E has no way of submitting an invalid state-change. We now moved from an audit trail to an Active Choreography.

There is still an obvious flaw in the proposed implementation: How do we prevent E from halting any state change at any moment? She could just interrupt the continuity of the business process. So far we do not have any good answer to this. We could hope that it is still in the best interest of E that the business process is fully executed and she therefore would not want to interrupt it. A more technical option would be to have the clients generate random messages to confuse E, so that it at least would be very hard for her to halt the business process at a certain state. However this is just a suggestion and topic to further research.

How can a client prove to the chain that the proposed state-change is legal? zero-knowledge? confirmation of other clients

References

1. Private transactions - wiki. https://wiki.parity.io/Private-Transactions, accessed: 2019-01-19
2. van der Aalst, W.M., Reijers, H.A., Weijters, A.J., van Dongen, B.F., De Medeiros, A.A., Song, M., Verbeek, H.: Business process mining: An industrial application. Information Systems **32**(5), 713–732 (2007)
3. 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)
4. 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)
5. 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)
6. 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)
7. Buterin, V., et al.: A next-generation smart contract and decentralized application platform. white paper (2014)
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. Ladleif, J.: Active Choreographies. Master's thesis, Hasso Plattner Institut, Prof.-Dr.-Helmert-Straße 2-3, 14482 Potsdam, Germany (2018)
11. 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)
12. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system (2008)
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. Noether, S., Mackenzie, A., et al.: Ring confidential transactions. Ledger **1**, 1–18 (2016)
15. 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)
16. 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)
17. Paverd, A., Martin, A., Brown, I.: Modelling and automatically analysing privacy properties for honest-but-curious adversaries. Tech. Rep. (2014)

18. 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)
19. 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)
20. Xiong, H., Zhang, X., Zhu, W., Yao, D.: Cloudseal: End-to-end content protection in cloud-based storage and delivery services. In: International Conference on Security and Privacy in Communication Systems. pp. 491–500. Springer (2011)