TU WIEN Informatics

Using Zero-Knowledge Proofs for the Confidential Execution of Collaborative Business Processes on Blockchain

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Software Engineering & Internet Computing

eingereicht von

Oskar Petto, BSc
Matrikelnummer 01627750

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Univ.Prof. Dipl.-Ing. Dr.techn. Wolfgang Kastner
Mitwirkung: Projektass. Dipl.-Ing. Thomas Preindl, BSc
Projektass. Dipl.-Ing. Martin Kjäer, BSc

Wien, 7. Dezember 2023

Oskar Petto   Wolfgang Kastner

TU WIEN Informatics

Using Zero-Knowledge Proofs for the Confidential Execution of Collaborative Business Processes on Blockchain

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Software Engineering & Internet Computing

by

Oskar Petto, BSc
Registration Number 01627750

to the Faculty of Informatics

at the TU Wien

Advisor:	Univ.Prof. Dipl.-Ing. Dr.techn. Wolfgang Kastner
Assistance:	Projektass. Dipl.-Ing. Thomas Preindl, BSc
	Projektass. Dipl.-Ing. Martin Kjäer, BSc

Vienna, 7th December, 2023

Oskar Petto   Wolfgang Kastner

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

Erklärung zur Verfassung der Arbeit

Oskar Petto, BSc

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 7. Dezember 2023

Oskar Petto

Danksagung

Zuerst möchte ich meinem Betreuer, Wolfgang Kastner, dafür danken, dass er mir die Möglichkeit geboten hat, Teil der Automation Systems Group zu sein. Diese Erfahrung hat es mir nicht nur ermöglicht, mit außergewöhnlich netten Kollegen zusammenzuarbeiten, sondern mir auch einen tiefen Einblick in den Alltag der wissenschaftlichen Forschung gegeben. Besonderer Dank gilt Thomas Preindl und Martin Kjäer für ihre außergewöhnliche Betreuung. Ihre Anleitung war von unschätzbarem Wert für den erfolgreichen Abschluss dieser Arbeit und trug maßgeblich dazu bei, dass mir das Arbeiten Spaß machte. Abschließend bin ich meiner Verlobten Lydia zutiefst dankbar für ihren Glauben an mich. In den Momenten, in denen ich daran zweifelte, diese Arbeit zu Ende zu bringen, waren es Lydias ermutigende Worte, die mein Vertrauen und meine Entschlossenheit wieder entfachten.

Acknowledgements

I first want to thank my advisor, Wolfgang Kastner, for offering me the opportunity to be a part of the Automation Systems Group. This experience has not only allowed me to engage with exceptionally nice colleagues but also provided me with a deep insight into the daily life of scientific research. Immense thanks are due to Thomas Preindl and Martin Kjäer for their extraordinary supervision. Their guidance was invaluable for the successful completion of this thesis and made the whole experience enjoyable. Finally, I am deeply grateful to my fiancée Lydia for her belief in me. In moments when I doubted my ability to complete this thesis, it was Lydia's encouraging words and reassurance that reignited my confidence and determination.

Kurzfassung

Blockchain-Technologie ermöglicht die Ausführung von zwischenbetrieblichen Geschäftsprozessen ohne die Notwendigkeit einer zentralen Vertrauensinstanz. Jedoch bringt die Transparenz öffentlicher Blockchains aufgrund der Sichtbarkeit aller enthaltenen Daten, Datenschutzherausforderungen mit sich. Um dies zu verhindern, findet sich in der Literatur ein neuartiger Ansatz, der nur Hashes von Prozessdaten in Smart Contracts speichert und die Übereinstimmung mit business process modeling notation (BPMN) Prozessdiagrammen mit Verwendung von zero-knowledge proofs (ZKPs) sicherstellt. Trotz aller Innovation, leidet ihre Implementierung unter hohen Ausführungskosten und langen Beweisgenerierungszeiten. Um diese Nachteile zu eleminieren, schlägt diese Masterarbeit ein System für die blockchain-basierte Ausführung von Prozesschoreographien vor, das BPMN-Choreographiediagramme zuerst in eine Zwischendarstellung umwandelt. Im Gegensatz zur Methode von Toldi und Kocsis, bei der jeden Prozess in zwei separate Softwarekomponenten transformiert wird, bestehend aus einer Off-Chain-Prover-Anwendung und einem Verifier-Smart-Contract, verwendet unser System nur ein einziges Paar von Prover- und Verifierartefakten. Beide Artefakte erhalten eine Zwischenrepräsentation einer Choreografie als Eingabe, wodurch es unserem System ermöglicht wird, alle Choreografien bis zu einer vordefinierten maximalen Größe mit nur einem Deployment zu unterstützen. Diese Methode verringert die Kosten für die Instantiierung neuer Prozesse um 90% im Vergleich zum Ansatz von Toldi und Kocsis. Darüber hinaus führt unsere Strategie, keine verschlüsselten Prozessdaten auf der Blockchain zu speichern, zu einer Reduzierung der mit Statusaktualisierungen verbundenen Kosten um 53%. Ergänzend dazu optimieren wir den Validierungsprozess von Choreographie-Instanzen, was wesentlich zur gemessenen Verringerung der Prover-Zeit um 99% beiträgt.

Abstract

Blockchain technology facilitates the execution of interorganizational business processes without the need for a central trusted authority. However, the transparency of public blockchains poses privacy challenges due to the visibility of all recorded data. To address this, Toldi and Kocsis propose a novel approach that stores only hashes of process data in smart contracts and ensures alignment with business process modeling notation (BPMN) process models using zero-knowledge proofs (ZKPs). Despite its innovation, their implementation faces high execution costs and lengthy proof generation times. Addressing these limitations, this thesis introduces a blockchain-based system for the execution of process choreographies, which first converts BPMN choreography diagrams into an intermediate representation. Unlike the method of Toldi and Kocsis, which transforms each process into two separate software artifacts comprising an off-chain prover application and a verifier smart contract, our system employs a single pair of prover and verifier artifacts. Both artifacts receive an intermediate representation of a choreography as input, enabling our system to support all choreographies up to a predefined maximum size with just a single deployment. This method reduces instantiation costs of new processes by 90% compared to the approach of Toldi and Kocsis. Moreover, our strategy of not storing encrypted process data on the blockchain results in a 53% reduction in the costs associated with state updates. Complementing this, we optimize the validation process of choreography instances, which contributes substantially to the measured 99% decrease in prover time.

Contents

CHAPTER 1

Introduction

1.1 Problem Statement and Motivation

Collaboration is key to enhancing productivity, and this is particularly true for collaborative business processes. Unlike processes confined within a single organization, collaborative ones often grapple with challenges like a lack of trust among participants. In scenarios where trust is minimal, reaching a consensus on who should oversee the execution of the process can be problematic [Web+16a].

Blockchain technology emerges as a robust solution to facilitate business processes in such low-trust environments. It creates a verifiable and immutable record of transactions, which is established through consensus between all participants in the network, thus serving as a common source of truth [Fri+18]. In combination with smart contracts, blockchains allow executing arbitrary code with transparency and can be used to enforce agreements between parties without relying on a trusted third-party [Kha+21].

In recent years, the research interest in deploying blockchain technology for business process management (BPM) has led to its application within most phases of the traditional BPM lifecycle [Gar+20]. A key phase within this lifecycle is the implementation phase, where a process model is converted into software components that manage the execution of the process [Dum+18]. In the context of blockchains, smart contracts serve to enforce the business process by validating and executing transactions only if they comply with the established process model [SW22]. The monitoring phase is concerned with collecting events of process executions in an event log to subsequently derive performance measures. Blockchains can aid with the monitoring phase by providing immutable storage for event logs, where the provenance of each event is known [DMP22].

Nevertheless, the inherent transparency of public blockchains, such as Ethereum, presents significant privacy concerns due to the public accessibility of all recorded process data[1].

[1]Private blockchains are not addressed in this thesis

This factor renders approaches that lack a strong emphasis on privacy particularly inappropriate for many organizations [LWW19]. Although encryption has been proposed as a means to hide sensitive process data on the blockchain [CRF18; MDW22], it also obscures the process execution from public verification.

ZKPs present a robust solution for validating statements about private data without revealing the data itself. A ZKP employs the interaction between a prover and a verifier to ascertain the correctness of a computation without revealing its inputs to the verifier. Zero-knowledge succinct argument of knowledges (zk-SNARKs), a non-interactive variant of ZKPs with greater practical efficiency, even allow a prover to prove their knowledge of a secret through a succinct, single-message proof. The combination of their efficiency and the ability to succinctly express complex conditions, makes zk-SNARKs suitable for applications such as digital signature schemes, electronic voting and user identification protocols [PNP20].

A notable area of research explores the use of zk-SNARKs to ensure the consistency of business process executions with their predefined models [Too20; TK23]. However, generating these zk-SNARK proofs can be computationally demanding, an issue this thesis seeks to address by proposing more efficient methods.

1.2 Aim of the Thesis and Expected Results

The primary objective of this research is to design a system architecture that facilitates multi-party execution of cross-organizational business processes via blockchain technology. This system ensures the confidentiality of sensitive information and allow the verification of conformity with the process model using zk-SNARKs. We seek to accommodate a broad spectrum of processes and achieve greater efficiency in proof generation than existing approaches.

The proposed system comprises smart contracts that manage the process state, alongside off-chain software artifacts tasked with submitting updates to these contracts. These updates undergo validation by the smart contracts to ensure they adhere to the established process model and are agreed upon by all relevant participants. For instance, a process step like message exchange requires approvals from both the sender and receiver to update the process state. The smart contracts also maintain a transaction history, serving as an event log that timestamps every authorized state update.

Off-chain software components play a crucial role in updating the process state. These components are designed to protect the confidentiality of sensitive process data, thus protecting participant privacy. In order to maintain public verifiability, they are responsible for generating zk-SNARK proofs to verify the authenticity of state updates. Furthermore, off-chain components facilitate the creation of update transactions on the blockchain and manage interactions among the process participants.

Our goal is to enhance the practicality of confidentially executing collaborative business processes on the blockchain. We plan to achieve this by reducing the prover time required for the employed zk-SNARK proofs compared to previous approaches, thereby ensuring process conformance more efficiently.

In this thesis, the following research questions (RQ) are answered:

RQ1 Which are the requirements of a system that enables the confidential execution of collaborative business processes on blockchain and uses zk-SNARKs to enable public verification of process executions?

RQ2 Which aspects of the state of the art must be adopted to create an architecture of a system that satisfies these requirements?

1.3 Methodology

This section outlines our selected methodology to acquire essential background knowledge and relevant literature, identify the necessary requirements, and design a system for these requirements.

1.3.1 Literature Review

To acquire essential background knowledge for this thesis, we conduct narrative literature reviews focusing on relevant subject areas. We collect scientific publications using web-based search engines, notably Google Scholar. This exploration of existing literature should lead to a comprehensive understanding of verification steps in blockchain-based process execution and the necessary knowledge for devising algorithms that can efficiently be verified using zk-SNARKs. Furthermore, insights gleaned from these studies contribute to defining the requirements of our proposed system.

1.3.2 Requirements Engineering

We utilize requirements engineering to identify and compile the yet unknown functional requirements of our proposed system. This process encompasses examining and adapting requirements from existing blockchain-based business process execution systems, as well as extracting requirements through literature review in this field. Examples of potential requirements for our system may include the enforcement of participant agreement on a process state or the verification of specific time constraints. These identified requirements are instrumental in addressing research question 1.

1.3.3 Design Science

Design Science is a methodology, where an artifact is created and applied to a problem to gain an understanding of the problem domain and how it can be solved [Hev+04]. In this thesis, the primary artifact is a system architecture utilizing zk-SNARKs to verify confidential process executions on a blockchain. This architecture is then applied in a prototype to assess performance metrics such as proving time and on-chain verification costs, evaluating the architecture's overall feasibility. Additionally, the prototype undergoes a qualitative analysis of its capabilities and is benchmarked against pre-established requirements and current state-of-the-art approaches, addressing research question 2.

1.4 Structure of the Thesis

The remaining chapters of this thesis are organized as follows: Chapter 2 provides essential background information. Chapter 3 explores literature on the use of zk-SNARKs in blockchain applications and the execution of business processes using blockchain technology. Building on these findings, Chapter 4 outlines the requirements for our system. Our proposed system is presented in Chapter 5, followed by an evaluation in Chapter 6. The thesis concludes with Chapter 7.

CHAPTER 2

Background

2.1 Business Process Management

A business process involves a defined set of actors that execute one or multiple activities in coordination to realize a business goal. BPM refers to the set of methods and techniques that improve the efficiency and effectiveness of business processes. Among these, process modeling is a fundamental BPM activity that provides structured diagrams that capture the sequence of tasks, their interdependencies, and the roles of those involved. These models are crucial for establishing a shared understanding of business processes and ensuring that operational practices align with the intended design.

2.1.1 Business Process Modeling Notation

A popular modeling language for business processes is the BPMN [OMG11], which is a graphical notation that easy to understand and at the same time has clear operational semantics. The BPMN defines four categories of diagram elements, which are depicted in Figure 2.1. Flow objects are the main building blocks of business processes and include events, activities and gateways. Events and activities represent their real life analogues, gateways describe specific control flow patterns. Activities can be atomic, in which case they are called tasks, or they can represent sub-processes. Swimlanes represent organizational boundaries within a diagram. Each pool corresponds to an organization and each lane within a pool to a department within the organization.

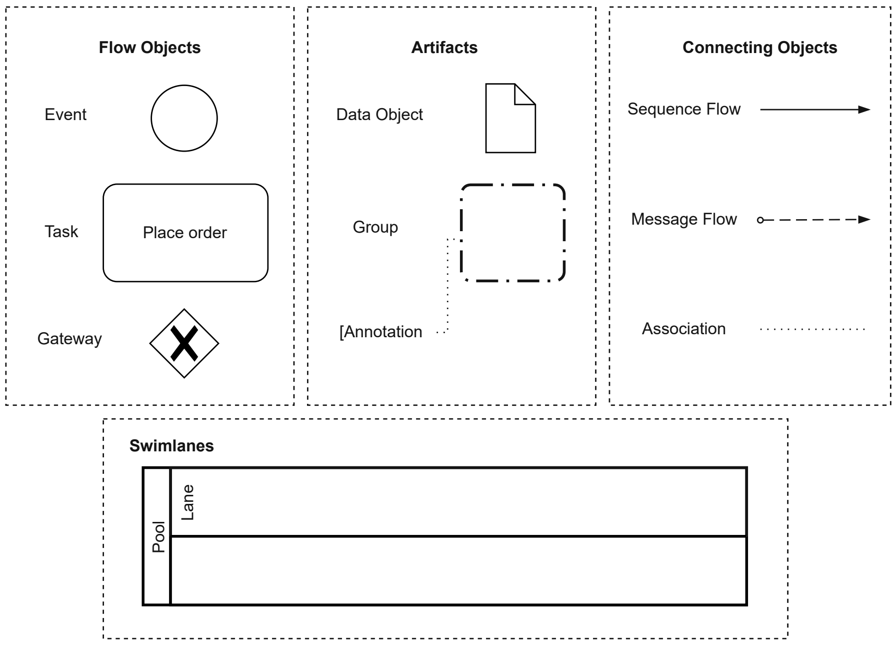

Figure 2.1: The categories of diagram elements of BPMN, taken from [Wes07]

Artifacts, such as groups, annotations and data objects enhance business process models with additional information. For instance, data objects are used to document the data associated with a business process. The last category of diagram elements are connecting objects, which connect the previously mentioned elements. Sequence flows specify the ordering of flow objects, message flows denote messages that are exchanged between organizations represented by pools, and associations link artifacts to other diagram elements [Wes07].

In BPMN, each gateway is either a split gateway with one incoming edge and at least two outgoing edges or a join gateway with at least two incoming edges and one outgoing edge. Split as well as join gateways are assigned a logical function that determines how many outgoing, respective incoming, edges may be used in the instances of the modelled process. There exist gateways for the XOR, AND, OR functions and complex gateways that realize arbitrary logical functions. The outgoing edges of XOR split gateways are labeled with logical expressions that use events or process data and one of these edges is selected based on which expression evaluates to true. The BPMN does not define an expression language for specifying these gateway conditions and these can range from an informal description in any human language to an expression in a programming language [Wes07].

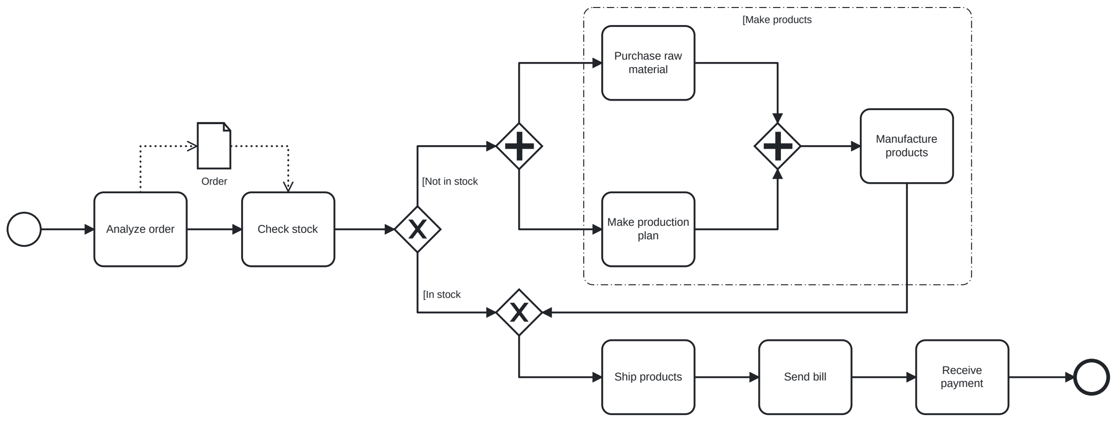

Figure 2.2: A model of an ordering process in BPMN, taken from [Wes07]

Figure 2.2 shows a model of an ordering process in BPMN, which was taken from [Wes07]. The process begins with a start event, which in this particular process represents the placement of an order. The seller then analyzes the order and checks the stock in sequence, both of these tasks use the order as a data object. After that, there is an XOR split gateway and the seller takes exactly one path based on whether they have the products in stock. If the products are not in stock, an AND split gateway splits the control flow into two separate threads and the seller purchases raw materials and makes a production plan in parallel. The following AND join gateway requires that both incoming edges are used by a process instance. The subsequent task of manufacturing products is grouped together with the two previous tasks and annotated to provide additional information. After the group of tasks, and also if the products are in stock, an XOR join gateway requires that exactly one of the incoming paths was taken. The seller then consecutively ships the products, sends the bill and receives the payment before finally arriving at the end event, which signals the end of the process.

Figure 2.3 shows a supply chain scenario that was adopted from Weber et al. [Web+16a] with four different collaborating organizations represented by pools. A Bulk Buyer initiates the business process and orders goods from a Manufacturer, which is represented by a message from the Bulk Buyer to the Manufacturer. The Manufacturer then calculates the demand and places an order for the required materials with the Middleman. The Middleman directly forwards the order to the Supplier and hires the Special Carrier for transporting the materials to the Manufacturer. Upon receiving the order from the Middleman, the Supplier produces the materials, prepares the transport and provides transport information to the Special Carrier. After the delivery of the materials, the Manufacturer starts production and ultimately delivers the product to the Bulk Buyer.

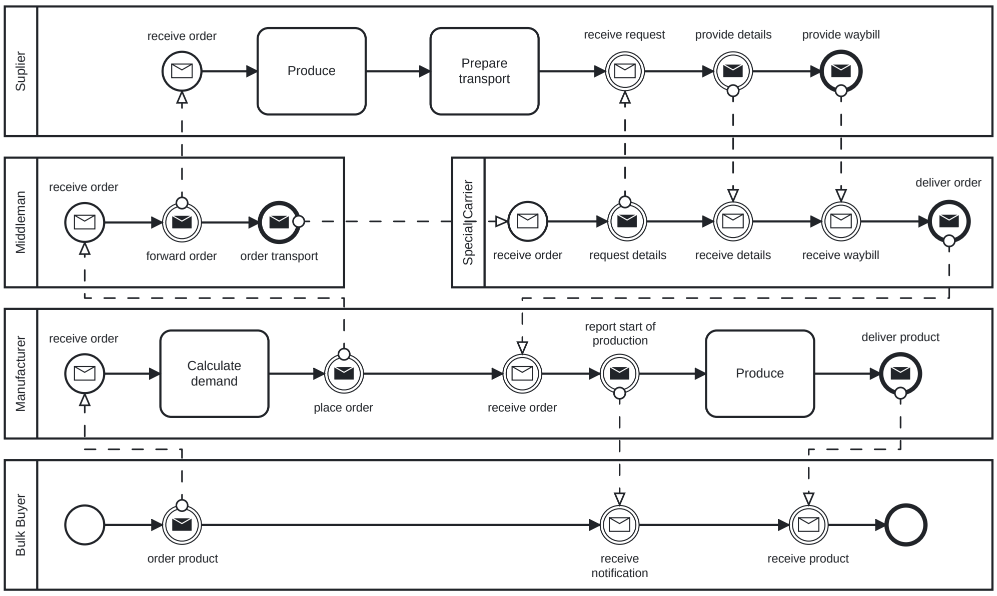

Figure 2.3: A supply chain scenario in BPMN, taken from [Web+16a]

The supply chain scenario illustrated in Figure 2.3 exemplifies a process choreography, as there is no single party that has visibility into all the exchanged messages [Web+16a]. The BPMN includes a choreography diagram type for process choreographies with task elements shown Figure 2.4, emphasizing on the messages exchanged between organizations. Each choreography task specifies the initiating and responding participants along with the initial message and any subsequent response.

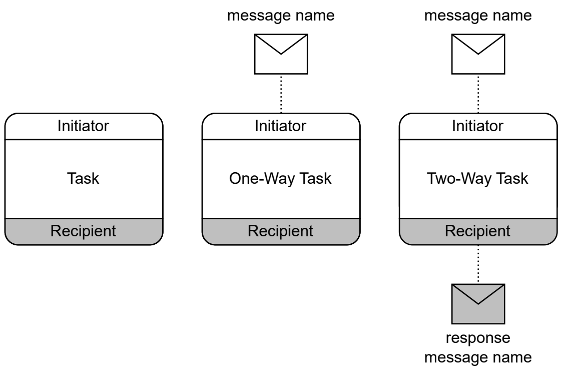

Figure 2.4: Task elements in BPMN choreography diagrams

Figure 2.5 depicts the aforementioned supply chain scenario as a BPMN choreography diagram. The choreography diagram, in contrast to the process diagram, abstracts away the internal process details of the participants, resulting in a diagram with a reduced number of elements.

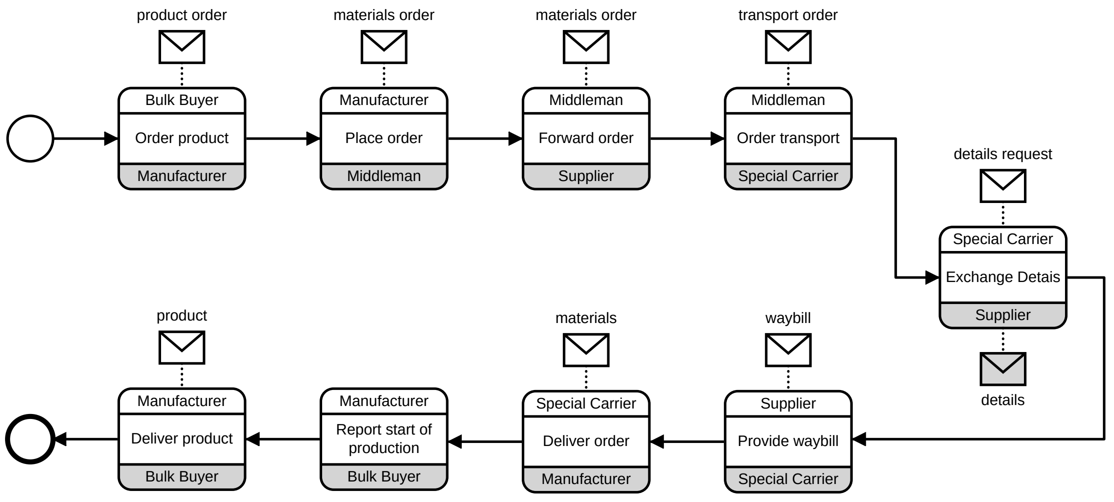

Figure 2.5: The supply chain scenario from [Web+16a] as BPMN choreography diagram

2.1.2 The BPM Lifecycle

The BPM lifecycle [Dum+18] is a strategic approach for the ongoing management and iterative enhancement of an organization's business processes. It is structured as a recurring series of six different phases, which are shown in Figure 2.6.

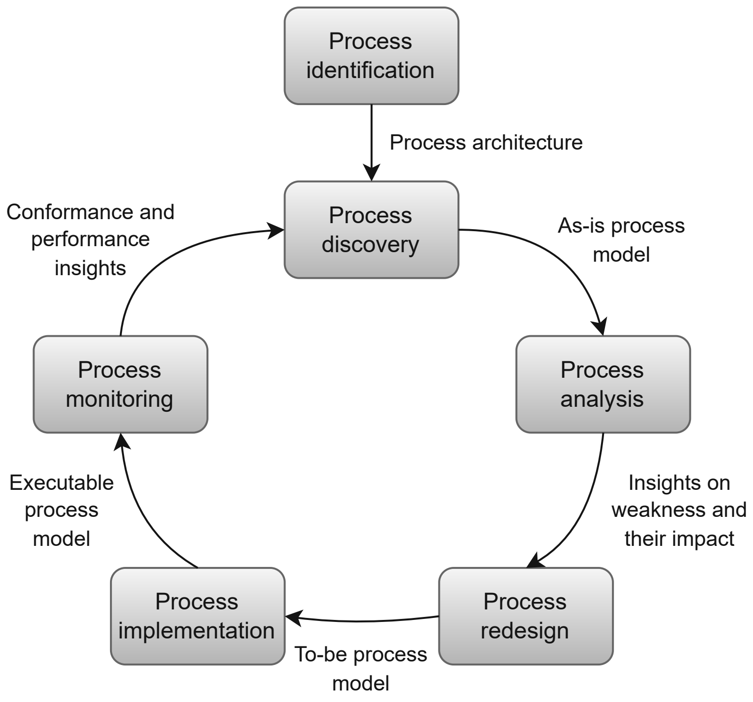

Figure 2.6: BPM lifecycle, taken from [Dum+18]

(1) Process identification. In this phase, we aim to get an overview of the employed business processes and derive methodologies for measuring their outcomes. (2) Process discovery. The goal of this phase is to obtain an as-is process model of one employed process. (3) Process analysis. Here, we identify issues with the as-is process, quantify them whenever that is possible and document them as a structured collection of issues. (4) Process redesign. The previously identified issues are addressed by introducing improvements to the as-is process, which results in a to-be process model. (5) Process implementation. In this phase, the as-is process is modified to align with the to-be process, which involves changing existing IT systems that handle process automation. (6) Process monitoring. Once the migration to the to-be process is complete, we analyze the performance of the running process and check whether it conforms to the to-be process model. These insights are used to derive a new as-is process model and the cycle repeats.

2.2 Conformance Checking

An event log is a collection of events that document the tasks carried out in a business process, including the specific time and case of execution. Grouping these events by case generates sequences known as traces, which provide the recorded behavior for specific instances of the process. Conformance checking evaluates the discrepancies between these recorded behaviors and the intended behaviors outlined in a process. This enables the verification that executions of the process conforms to any regulations and standards, provided that the process model itself is compliant [Car+18].

The main focus of conformance checking lies on the correct ordering of tasks inside a process. Nonetheless, process models may also include business rules for aspects of processes besides the specified control flow. In such scenarios, conformance checking must take into account data, resource and time perspectives, which may require supplementary data in the event log. For instance, if a process involves rules regarding process data, then events must contain the corresponding data objects. Similarly, if the process involves rules concerning the eligibility of allocating resources, events need to include information on which participants are involved in the task [Car+18].

The difference between online and offline conformance checking is determined by the timing of when the checking is performed. Offline conformance checking, or simply conformance checking, examines the process after it has been completed and aims to identify issues or deviations from the expected behavior. On the other hand, online conformance checking is performed during the process execution to ensure that the process is being executed correctly. It operates on a stream of events and uses a different approach from offline conformance checking, as it doesn't require that the current state is the final state of the model, but rather that the final state can be reached from the current state [Car+18].

2.2.1 Fitness and Precision

Event logs and process models represent the recorded (L) and modelled (M) behavior of business processes, respectively. The relationship between these two sets can be evaluated based on two dimensions: fitness and precision.

Fitness measures how well the behavior in the event log is captured by the model and is defined as

$$\mathsf{fitness} = \frac{|L \cap M|}{|L|}. \tag{2.1}$$

Precision is used to quantify the behavior of the model compared to the event log and is calculated with

$$\mathsf{precision} = \frac{|L \cap M|}{|M|}. \tag{2.2}$$

The possible relations between recorded and modelled behavior are depicted in Figure 2.7 with the corresponding fitness and precision values in Table 2.1. Relation A connects a process model with an event log that has both extra and missing entries. Relation B pairs the model with a log that only has entries not in the model. Relation C shows a log that includes everything the model expects and some additional entries. Relation D pairs the model with a log that fits exactly. Lastly, Relation E links the model with a log that's short on some entries.

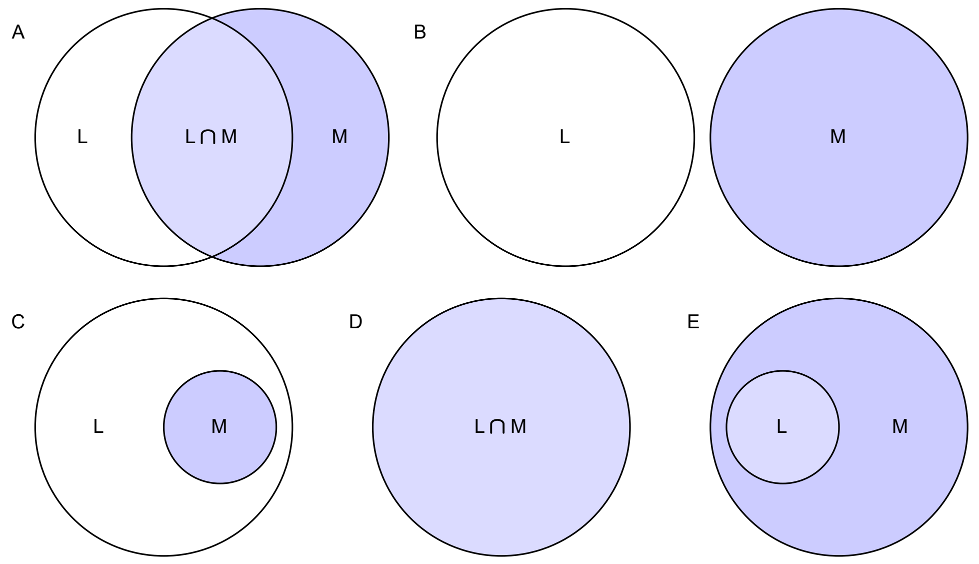

Figure 2.7: Venn diagrams of possible relations between recorded (L) and modelled (M) behaviors, taken from [Car+18]

Relation	Description	Fitness	Precision
A	$L \not\subseteq M$ and $M \not\subseteq L$	<1	<1
B	$L \cap M = \emptyset$	=0	=0
C	$M \subseteq L$	<1	=1
D	$L = M$	=1	=1
E	$L \subseteq M$	=1	<1

Table 2.1: Fitness and precision values for the relations in Figure 2.7

2.2.2 Petri nets

Despite its widespread use as a process modelling language, BPMN is less popular in conformance checking literature, where the most commonly used process modelling language by far is the Petri net [Dun+19]. However, there is a direct mapping from process models in BPMN to Petri nets, which was given by Dijkman, Dumas, and Ouyang [DDO08] under the condition that the process models are well-formed. For a definition of a well-formed BPMN process and the mapping from BPMN tasks, events and gateways to transitions and places within a Petri net, we refer to their paper.

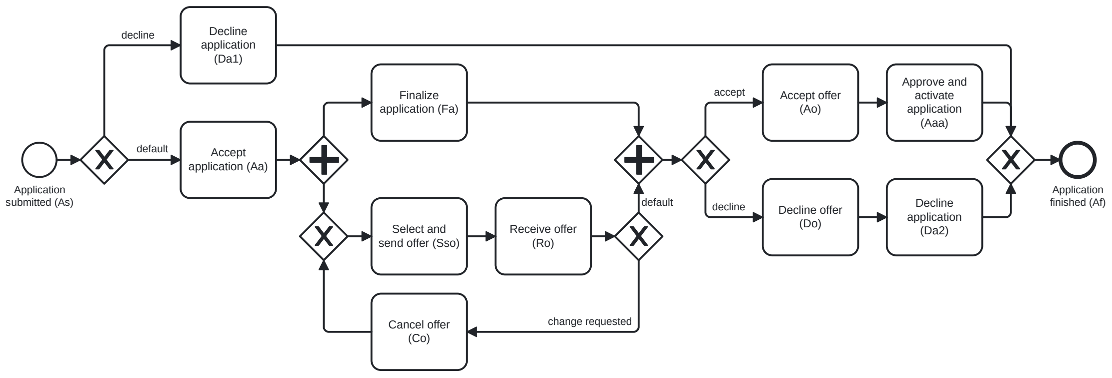
decline
Decline application (Da1)
default
Accept application (Aa)
Application submitted (As)
Finalize application (Fa)
Select and send offer (Sso)
Receive offer (Ro)
Cancel offer (Co)
change requested
default
accept
decline
Accept offer (Ao)
Approve and activate application (Aaa)
Decline offer (Do)
Decline application (Da2)
Application finished (Af)

Figure 2.8: Example of a loan application process, taken from [Car+18]

After applying this mapping to the example loan application process in Figure 2.8, we obtain the Petri net in Figure 2.9. Transitions t_0 through t_{12} are called silent transitions, because they do not correspond to tasks or events, but rather are introduced to capture the routing behavior of gateways.

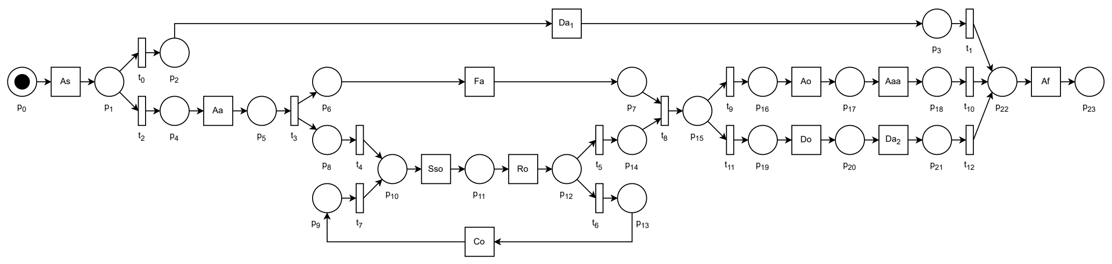
As
Aa
Fa
Sso
Ro
Co
Da$_1$
Ao
Aaa
Do
Da$_2$
Af
p$_0$ p$_1$ p$_2$ p$_3$ p$_4$ p$_5$ p$_6$ p$_7$ p$_8$ p$_9$ p$_{10}$ p$_{11}$ p$_{12}$ p$_{13}$ p$_{14}$ p$_{15}$ p$_{16}$ p$_{17}$ p$_{18}$ p$_{19}$ p$_{20}$ p$_{21}$ p$_{22}$ p$_{23}$
t$_0$ t$_1$ t$_2$ t$_3$ t$_4$ t$_5$ t$_6$ t$_7$ t$_8$ t$_9$ t$_{10}$ t$_{11}$ t$_{12}$

Figure 2.9: Petri net for loan application process

These silent transitions can be removed with the behavior-preserving reductions introduced by Murata [Mur89], which simplify the Petri net to the one in Figure 2.10.

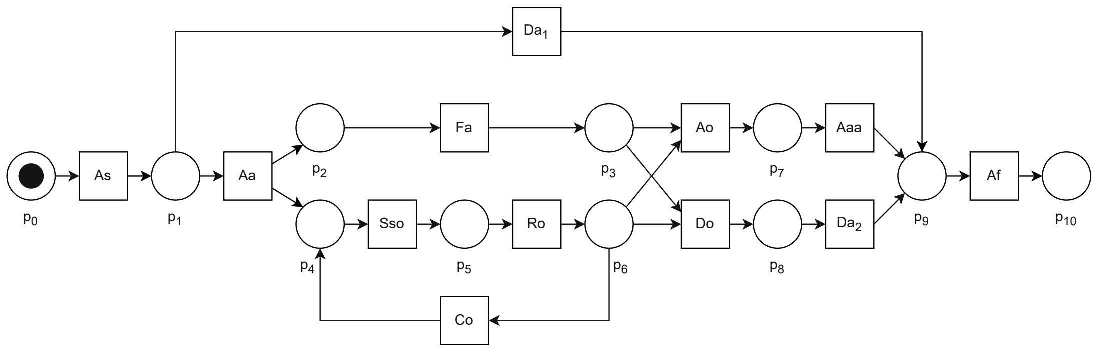
Da$_1$
As
Aa
Fa
Sso
Ro
Co
Ao
Aaa
Do
Da$_2$
Af
p$_0$ p$_1$ p$_2$ p$_3$ p$_4$ p$_5$ p$_6$ p$_7$ p$_8$ p$_9$ p$_{10}$

Figure 2.10: Petri net for loan application process without silent transitions

2.2.3 Algorithms

In the following section, we explore algorithms designed to evaluate the fitness and precision of an event log in relation to a process model. We progress through three principal methodologies utilized in conformance checking, presented in ascending order of complexity: rule-based checking, token-based replay, and alignments.

Rule checking

Rule checking algorithms derive a set of rules over the activities of a process model and verify these rules with respect to the traces of an event log. Unary rules specify how a single activity has to be executed, and binary rules describe the relationship between the executions of a pair of activities. A significant drawback of rule checking is that it becomes inefficient for larger processes as the number of necessary rules typically increases exponentially with the number of activities. Additionally, quantifying non-conformance can be challenging as it heavily depends on how the rules are derived [Car+18].

Token replay

Token replay algorithms require that processes are modelled as Petri nets, as shown in Section 2.2.2. The conformance of a single trace is then checked by replaying each event as transition in the Petri net and keeping track of consumed, produced, missing and remaining tokens. Let M be the model with the current number of token at each place and L be the event log, then replaying an event $\sigma \in L$ in M results in four token counts: $\mathsf{consumed}(\sigma, M)$, $\mathsf{produced}(\sigma, M)$, $\mathsf{missing}(\sigma, M)$, $\mathsf{remaining}(\sigma, M)$. Carmona et al. [Car+18] define the fitness of σ in M, by counting the number of missing and remaining token relative to the number of consumed and produced token, respectively, and then taking the average of both ratios

$$\mathsf{fitness}(\sigma, M) = \frac{1}{2}\left(1 - \frac{\mathsf{missing}(\sigma, M)}{\mathsf{consumed}(\sigma, M)}\right) + \frac{1}{2}\left(1 - \frac{\mathsf{remaining}(\sigma, M)}{\mathsf{produced}(\sigma, M)}\right). \tag{2.3}$$

The fitness of the trace L in M is then given by

$$\mathsf{fitness}(L, M) = \frac{1}{2}\left(1 - \frac{\sum_{\sigma\in L}\mathsf{missing}(\sigma, M)}{\sum_{\sigma\in L}\mathsf{consumed}(\sigma, M)}\right) + \frac{1}{2}\left(1 - \frac{\sum_{\sigma\in L}\mathsf{remaining}(\sigma, M)}{\sum_{\sigma\in L}\mathsf{produced}(\sigma, M)}\right). \tag{2.4}$$

A trace fits perfectly with fitness 1, if there are no missing or remaining tokens.

Token replay is based on the assumption that each event in the trace can be matched with one activity in the process model. If there are no matching activities, the event in question cannot be considered for the fitness of the trace and if multiple activities can be matched for an event, conformance checking becomes non-deterministic. In the latter case, there are multiple possibilities to replay a trace and one might search for the one with the highest fitness [Car+18]. Aside from this difficulty, token replay suffers from „token flooding“, where inserting a lot of missing tokens during the replay enables many unwanted transitions in the Petri net. Subsequently, the corresponding invalid events in the log then increase the fitness of the trace instead of decreasing it [BA19].

Alignments

Alignments aim to provide a better fitness measurement than token replay and capture the idea that there might be multiple possibilities of replaying a given trace in the model. An alignment connects a trace of the event log with a sequence of activities in a two-row matrix. Each column of an alignment is a pair (e_i, a_i) of an event e_i and an activity a_i, which together are called a move. Carmona et al. [Car+18] define three types of moves:

- Synchronous move ($e_i = a_i$ and $e_i \neq \gg$). The next event in the trace corresponds to the next activity in the execution sequence.
- Model move ($e_i = \gg$ and $a_i \neq \gg$). The model specifies that an activity should have been executed, but there is no corresponding event in the trace.

- Log move ($a_i = \gg$ and $e_i \neq \gg$). An activity has a corresponding event in the trace, even though it was not supposed to be executed according to the model.

Table 2.2 shows possible alignments for the process model in Figure 2.8. Alignment a shows three synchronous moves and the log move $(Af, \gg)$, because the Af event occurred twice. Alignments b and c show the same trace and execution sequence, but with a different ordering of the log move $(\gg, Da2)$ and the model move $(Aaa, \gg)$. Alignment d shows the same trace as alignments b and c, but with an execution sequence containing activities Ao and Aaa instead of Do and Da2.

a)

trace T_1	As	$Da1$	Af	Af
execution sequence E_1	As	$Da1$	Af	$\gg$

b)

trace T_2	As	Aa	Sso	Ro	$\gg$	Do	$\gg$	Aaa	Af
execution sequence E_2	As	Aa	Sso	Ro	Fa	Do	$Da2$	$\gg$	Af

c)

trace T_2	As	Aa	Sso	Ro	$\gg$	Do	Aaa	$\gg$	Af
execution sequence E_2	As	Aa	Sso	Ro	Fa	Do	$\gg$	$Da2$	Af

d)

trace T_2	As	Aa	$\gg$	Sso	Ro	Do	$\gg$	Aaa	Af
execution sequence E_3	As	Aa	Fa	Sso	Ro	$\gg$	Ao	Aaa	Af

Table 2.2: Possible alignments for the process model in Figure 2.8

Computing a fitness value of an event log and a model, requires computing multiple alignments and choosing one with the minimal cost, which is called an optimal alignment. To compute an optimal alignment, both the trace and the model must first be converted into Petri nets. The synchronous product model is created as a Petri net, incorporating places and transitions from both the model and trace Petri nets, along with additional synchronous transitions. One synchronous transition is added to the synchronous product model for every pair of transitions from the trace and model Petri nets that share the same label, connecting the inputs and outputs of both transitions. As a result, the synchronous product model encompasses all possible types of moves that may occur during an alignment, with each alignment corresponding to a specific execution sequence in the model. The cost of synchronous transitions is set to 0, while the cost of transitions corresponding to model and log moves is determined based on a defined cost function. The optimal alignment is then represented by the shortest path in the reachability graph of the synchronous product model, where each edge in the graph is labelled with the cost of the corresponding transition [Car+18].

Let M be a model, L be an event log, $\sigma \in L$ be a trace in the log, $\mathsf{cost}(\sigma, M)$ be the cost of the optimal alignment of σ and M, $\mathsf{cost}(\sigma, \langle\rangle)$ be the cost of aligning σ with an empty execution sequence and $\mathsf{cost}(\langle\rangle, x)$ be the cost of aligning an empty trace with some execution sequence $x \in M$. Then, Carmona et al. [Car+18] define the alignment-based fitness of σ in M as

$$\mathsf{fitness}(\sigma, M) = 1 - \left(\frac{\mathsf{cost}(\sigma, M)}{\mathsf{cost}(\sigma, \langle\rangle) + \min_{x \in M} \mathsf{cost}(\langle\rangle, x)}\right). \tag{2.5}$$

The fitness of L is given by

$$\mathsf{fitness}(L, M) = 1 - \left(\frac{\sum_{\sigma \in L} \mathsf{cost}(\sigma, M)}{\sum_{\sigma \in L} (\mathsf{cost}(\sigma, \langle\rangle)) + |L| \times \min_{x \in M} \mathsf{cost}(\langle\rangle, x)}\right). \tag{2.6}$$

The fitness metric of alignments is more robust than the fitness metrics of rule based conformance checking and token replay, leading to more accurate diagnostics. Additionally, alignments are configurable through the chosen cost function, which can take the likelihood and severity of deviations into account [vdAal16]. Because of these advantages, alignments represent the majority of conformance checking algorithms used in literature [Dun+19].

2.3 Zero-Knowledge Succinct Arguments of Knowledge

ZKPs and zk-SNARKs are protocols between a prover and a verifier, where the prover convinces the verifier of the correctness of some computation without revealing the secret inputs associated with it. This computation is defined as the decision circuit C of an NP language L, where any $x \in L$ is a true statement, while any $x \notin L$ is a false statement. The process of convincing of the verifier involves exchanging a sequence of messages. While ZKPs allow multiple message exchanges between the prover and verifier, zk-SNARKs are non-interactive, which means the prover only sends a single message, referred to as a proof, to the verifier. The security properties of these protocols can be summarized as:

- Correctness. When a statement evaluates to true, the prover can convince the verifier of its truth.
- Soundness. In case of a false statement, no malicious prover deceive the verifier into accepting it.
- Zero-Knowledge. For true statements, malicious verifiers learn nothing from executing the protocol with the prover, except the statement's validity.

For zk-SNARKs, the soundness requirement applies only to provers with limited computational power, and additionally demands that anybody convincing the verifier of a statement must have knowledge of a witness for that statement. Furthermore, for a non-interactive protocol to qualify as a zk-SNARK, the length of the proof must be either constant or logarithmic in relation to the size of the statement [Tha22].

Efficient zk-SNARKs also necessitate a trusted party to perform a one-time setup, which produces proving and verification keys for the prover and verifier, respectively. The three parties involved in a zk-SNARK are defined using the following algorithms [AN20]:

- $\mathsf{Gen}(1^\lambda, \mathsf{C}) \to (\mathsf{pk}, \mathsf{vk})$. Given a security parameter λ and a decision circuit C of an NP language L the Gen algorithm produces a proving key pk and a verification key vk, which are subsequently made public as protocol parameters.
- $\mathsf{Prove}(\mathsf{pk}, x, w) \to \pi$. When provided with a proving key pk, a true statement $x \in L$ and a corresponding witness w for x, the Prove algorithm generates a proof π that attests to the truth of $x \in L$.
- $\mathsf{Verify}(\mathsf{vk}, x, \pi) \to \{0, 1\}$. When given a verification key vk, a statement x and proof π, the Verify algorithm outputs 1 if $x \in L$ is true.

For zk-SNARKs that rely on a trusted party, the soundness property depends on the trusted party discarding the randomness used during the execution of the Gen algorithm. To increase the trust in this critical step, the Gen algorithm can be executed as a multi-party computation protocol. This enhancement ensures the soundness of the zk-SNARK as long as any one of the parties discards their randomness [Ben+15].

CHAPTER 3 Related Work

In this chapter, our objective is to provide a comprehensive overview of the scientific research relevant to our thesis. Our first area of focus is the realm of ZKPs in blockchain, because the work in this area explains mechanisms for enhancing blockchain privacy and can serve as inspiration for private blockchain applications. Moving on, we delve into ZKPs in blockchain applications. This topic is significant, because it explains how different aspects of blockchain applications can maintain privacy and which specific ZKP techniques are employed for achieving this goal. Furthermore, we survey the execution of business processes on the blockchain. Understanding this topic is essential to grasp how to efficiently execute these processes on the blockchain while considering the unique challenges and considerations involved. Lastly, we investigate the use of ZKPs for conformance checking. This area is most relevant to our thesis as it revolves around verifying the conformance of entire traces or individual process steps using ZKPs.

We conducted a narrative literature review for each of these topics and searched for scientific literature using the Scopus[1] and Scholar[2] databases in order to gain an understanding of the related work. We queried for scientific articles using boolean search queries, and only included results with at least 5 citations. In Scholar this query was directly used, while in Scopus we additionally used the `TITLE-ABS-KEY` operator to only include results, whose title, abstract or keywords match the query. We expanded upon the results by employing snowballing techniques, utilizing the related work sections of papers to discover additional relevant sources.

3.1 Zero-Knowledge Proofs in Blockchains

In this section we aim to capture scientific literature that covers the use of ZKPs for private transactions and, additionally, for private computations. Private transactions provide

[1]`https://www.elsevier.com/solutions/scopus`
[2]`https://scholar.google.com/intl/de/scholar/about.html`

the capability to obscure the transaction amount, ensure that the sender and receiver cannot be linked, and even hide the identities of both parties. Private computations are built on top of private transactions and encompass techniques for executing on-chain computations without revealing the input data and, in some cases, the computation itself. We initiated our search for articles in these domains using the following query.

```
 ( zero-knowledge  OR  snark )
AND  ( transaction  OR  payment  OR  "smart contract" )
```

3.1.1 Transactions

This section focuses on strategies for making blockchain transactions both secure and private. All strategies discussed involve invalidating sets of cryptocurrency coins and creating new sets, all the while implementing measures to ensure these sets cannot be linked. An inherent challenge in this context is the prevention of double spending, which all the covered methods address similarly.

„Zerocoin“ [Mie+13]

By concealing the identity of the transaction sender, the Zerocoin protocol became the first proposal to address the issue of privacy in Bitcoin transactions. To achieve this, the protocol introduces a token called Zerocoin with a fixed Bitcoin value and uses a cryptographic accumulator to maintain a record of all Zerocoins. In the minting phase, the user generates a random serial number S and computes a commitment C to S using randomness r. To mint C as a Zerocoin, the user sends its Bitcoin value to an escrow pool and includes C in the mint transaction. Once miners have successfully validated the transaction, C is added to a global accumulator. In the spending phase, the user broadcasts a spend transaction with a serial number S of a Zerocoin and a ZKP, which proves that the user knows r, such that a commitment to S with randomness r is accumulated in the global accumulator. Miners verify the proof and check whether S has been used in any previous spend transactions to prevent the double-spending of C. Upon successful validation, the user is able to redeem the value of a Zerocoin from the escrow pool. The Zerocoin protocol hides the origin of Zerocoins, because the mint and spend transactions cannot be linked together.

„Zerocash“ [Ben+14]

The Zerocash protocol improves upon the Zerocoin protocol by allowing coins to have arbitrary values and by hiding the values of coins. In Zerocash, addresses are used to identify the owners of Zerocash coins, and pseudorandom functions are employed to derive addresses and the serial numbers of coins. To mint a Zerocash coin of Bitcoin value v, the user selects an address a, generates a coin secret ρ, and then computes an intermediate commitment k to a and ρ using randomness r and a coin commitment cm to v and k using randomness s. The mint transaction of the Zerocash coin $\mathbf{c} := (a, v, \rho, r, s, \mathsf{cm})$

deposits v Bitcoin and includes the values v, k, s and cm so that anyone can verify that cm is a coin commitment of a Zerocash coin of value v.

Unlike the Zerocoin protocol, the Zerocash protocol only keeps track of the coin commitments of unspent coins, which are stored in a Merkle tree. A pour transaction is used to spend a set of input coins by „pouring“ their value into a set of output coins such that the total value of the input coins equals the total value of the output coins. The pour transaction includes the serial numbers of the input coins, coin commitments of the output coins and a zk-SNARK proof. As in the Zerocoin protocol, serial numbers of Zerocash coins are used to prevent double-spending attacks. The zk-SNARK proof in the pour transaction ensures that the user knows all input and output coins and a secret key for each input coin, such that:

- The output coins are well-formed, as specified by the minting process.
- The secret keys match the addresses of the input coins.
- The serial numbers of the input coins are computed correctly.
- The coin commitments of the input coins are leaf nodes in the Merkle tree.
- The total value of input coins equals the total value of output coins.

To transfer the secret components of the output coins to the transaction recipient, the protocol utilizes a public-key encryption scheme and includes the ciphertext in the pour transaction. Additionally, the protocol supports pour transactions with public outputs, enabling users to exchange their Zerocash coins for Bitcoin.

3.1.2 Programmability

This section covers results that facilitate the private execution of code on the blockchain. In contrast to the transparency of conventional blockchain computations, the results discussed here focus on concealing input data and, in some instances, even the computation process itself. These approaches have to tackle challenges, such as creating an expressive language for private blockchain computations, as well as allowing users to perform computations on private data to which they do not have direct access.

„Hawk“ [Kos+16]

Hawk is a privacy-preserving smart contract platform that provides a framework for building confidential applications on public blockchains. Hawk relies on a distributed ledger to facilitate confidential currency transfers, which is achieved by utilizing a modified version of Zerocash. It introduces primitives that extend the private ledger's functionality, enabling the execution of smart contracts while preserving transactional privacy.

One of the primitives introduced by Hawk is the freeze operation. This operation enables users to commit their coins and private inputs for a specific computation. These commitments, accompanied by a zk-SNARK proof proving a statement similar to Zerocash's pour operation, are then submitted to a smart contract. The smart contract subsequently places the received coins into a pool of frozen coins. During the compute operation, users are required to disclose their coins and private inputs to a trusted party called the manager, who is expected to maintain their confidentiality. Off-chain, the manager executes the desired computation to determine the payout for all users and generates a zk-SNARK proof that attests to the correctness of the computation. The process concludes with the finalize operation, where the manager pours the frozen coins into a set of new coins in the global coin pool and submits the zk-SNARK proof of correctness to the smart contract.

Hawk provides its own programming language for smart contracts, which are compiled into a public and a private contract. The public contract can be directly deployed to a blockchain, while the private contract is transformed into an arithmetic circuit. This circuit is then augmented with gates that handle commitments, encryption and ensure the balance of input and output coins.

„Zkay“ [Ste+19]

Another programming language for smart contracts is Zkay, which allows labelling data as private and associate it with corresponding accounts. These Zkay smart contracts can be compiled into deployable smart contracts on Ethereum, alongside arithmetic circuits specified in the ZoKrates [ET18] domain specific language (DSL). The ZoKrates framework then handles the generation and verification of zk-SNARK proofs during the execution of the deployed smart contracts. The main idea behind this transformation is to encrypt private values under the owner's public key and to utilize zk-SNARK proofs to ensure that state modifications align with the smart contract. Consequently, transactions in Zkay are unable to operate on values owned by parties other than the caller. However, ZeeStar [Ste+22] subsequently addressed this limitation by leveraging homomorphic encryption, enabling the addition of values from other parties.

„ZEXE“ [Bow+20]

Zexe also extends Zerocash to enable decentralized private computations, but takes a different approach from Hawk. Its approach involves substituting Zerocash coins with records that carry an arbitrary payload instead of the coin's value. These records also include a birth predicate and a death predicate. In order to exchange old records for new ones, both the death predicates of the input records and the birth predicates of the output records must evaluate as true. Zexe records serve the purpose of implementing Zerocash mint and pour operations, while also facilitating the implementation of custom programs in a manner similar to Bitcoin Script. Commitments and zk-SNARKs hide all information about the payload and the birth and death predicates, which means not only the data, but also the computations themselves remain confidential. However, Zexe lacks support for loops, which are typically available in smart contracts.

3.2 Zero-Knowledge Proofs in Blockchain Applications

Among the surveyed areas, the literature on employing ZKPs in various blockchain applications stands out as the most diverse. Some blockchain applications use general ZKPs, while others develop custom ZKP protocols. The use of ZKPs range from verifying the knowledge of a hash preimage to validating the correctness of a step within a complex multi-round protocol employing various cryptographic schemes. To obtain literature about the application of ZKPs in blockchain-based systems, we used the following boolean search query.

```
  ( zero-knowledge OR snark )
AND ( blockchain OR "smart contract" )
AND ( application OR system )
```

In order to provide a structured presentation of the results, we categorize them based on their primary use cases.

3.2.1 Auctions

In this section, we cover the utilization of blockchain technology for conducting sealed-bid auctions. Sealed-bid auctions involve bidders placing their bid in sealed envelopes and handing them to the auctioneer. The auctioneer subsequently opens the envelopes and declares the bidder with the highest bid as the winner of the auction. Sealed-bid auctions may be realized as smart contracts on a blockchain to increase the transparency of the process. However, this transparency may compromise the confidentiality of the bids.

„Verifiable Sealed-Bid Auction on the Ethereum Blockchain“ [GY19]

Galal and Youssef aim to introduce privacy to sealed-bid auctions on the blockchain with a system that utilizes homomorphic commitments and ZKPs. In a first step, the smart contract receives homomorphic commitments to the bids of all participating bidders. Then, the bidders privately reveal their bids to the auctioneer, who determines the winner of the auction and submits it to the smart contract. Finally, for each losing bid the auctioneer and the smart contract engage in an interactive zero-knowledge (ZK) protocol to convince the smart contract that the winning bid is in fact larger than the losing bid.

3.2.2 Authentication

This section discusses using ZKPs for anonymous user or data authentication in blockchain. Anonymous authentication verifies users or entities without disclosing personal information. ZKPs enable this by allowing users to prove their identity or authorization without revealing sensitive data. The methods introduced in this context primarily rely on ZKPs of knowledge of a shared secret to authenticate various entities, including users, Internet of Things (IoT) devices, and electric vehicles.

„Zk-AuthFeed“ [Wan+19]

Zk-AuthFeed is a system that ensures the authenticity and confidentiality of data used in smart contracts. A typical use case involves a decentralized application (DApp), users of the DApp and a data authenticator. The data authenticator generates authentic data for users and computes a hash using the data, the user's ID, the present timestamp, and a random value before signing it. Given the nature of DApps, any user can perform the desired computation itself to obtain the result as well as compute the aforementioned hash. After obtaining both values and assuming that both computations exist as arithmetic circuits, the user is able to generate a zk-SNARK proof that both values were computed by using the same data as input to both circuits. The user then submits the result of the computation, the hash, the signature from the data authenticator and the zk-SNARK proof to a smart contract. The smart contract is able to verify the authenticity of the input data as well validity of the result by verifying the signature and the zk-SNARK proof, respectively.

„PUF-derived IoT Identities in a Zero-Knowledge Protocol for Blockchain“ [Pra+20]

Prada-Delgado et al. employ blockchain as a platform for authenticating IoT devices using public-key cryptography based on the learning parity with noise problem. Each IoT device is capable of independently generating its own secret key through the utilization of a physical uncloneable function, which takes advantage of the variability introduced during the device's manufacturing process. This secret key and a similarly generated secret error are used to compute the corresponding public key, which can be seen as a commitment to the private key with the secret error used as randomness. An IoT device authenticates itself with a public key and a ZKP that proves knowledge of a valid opening of the commitment represented by the public key.

„Privacy-Preserving Authentication Scheme for Connected Electric Vehicles Using Blockchain and Zero Knowledge Proofs“ [GAC20]

An approach of authentication through knowledge of a shared secret was presented by Gabay, Akkaya, and Cebe to authenticate electric vehicles at charging stations. Initially, the service provider and authorized electric vehicles exchange a shared secret, parameters of the Pedersen commitment scheme [Ped92] and a ZoKrates [ET18] program that checks whether the hash of its input matches the hash of the secret. To authenticate itself, each electric vehicle must first compute a zk-SNARK proof for the ZoKrates program and a Pedersen commitment to its desired charging slot. The vehicle then submits both and the charging slot encrypted with the public key of the service provider to an authentication smart contract. If the submissions are valid, the smart contract publishes an event containing the encrypted charging slot, which is decrypted by the service provider to book the charging slot for the vehicle. Upon arriving at the charging station, the vehicle opens the Pedersen commitment to its assigned charging slot. Subsequently, the service provider can verify that it is the same vehicle that booked the current charging slot.

„Privacy-Preserving Blockchain-Based Systems for Car Sharing Leveraging Zero-Knowledge Protocols“ [Gud+20]

Gudymenko et al. use zk-SNARKs to grant authorization to users in a car sharing service. The service needs to verify if users possess the required attributes, such as a valid driving license for the appropriate vehicle class and the minimum duration of its validity. Prior to utilizing the service, users are required to authenticate themselves with the issuer of each attribute and provide the hash of a secret value. Upon successful authentication and if users have the required certification, they receive a certificate containing the issuer's signature of the attribute and the submitted hash. When a user then wants to rent a car, they provide the certificate and a zk-SNARK proof that attests to their knowledge of a preimage of the contained hash. The implementation leverages ZoKrates and Ethereum to verify zk-SNARK proofs within a smart contract, which convinces a car terminal to authorize the rental of the car.

„A Zero-Knowledge-Proof-Based Digital Identity Management Scheme in Blockchain“ [YL20]

Yang and Li propose a digital identity management scheme on blockchain, where an identity consists of a set of attributes that are bound to an identifier. Identity providers issue each attribute separately as a token that contains a hash of the recipient's public key concatenated with the attribute's identifier. Users then need to claim their tokens by providing a ZKP of knowledge of a preimage of this hash. After a token has been claimed, the owner of the recipient's secret can use the token to authenticate themselves. For this, they need to provide a ZKP of knowledge of a preimage of the hash contained in the token and the recipient's secret key. A hash of the secret key concatenated with a random value acts as a nullifier to ensure that this proof can only be submitted once.

3.2.3 Delivery

In this section, we cover blockchain applications that provide a proof of data delivery. To generate such proof, both the data sender and receiver are required to collaborate, while protective measures are in place to discourage any party from gaining an unfair advantage through non-compliance with the established protocol. The proof itself always includes the recipient's signature on the delivered data, which the sender subsequently presents to a smart contract.

„Incentivized Delivery Network of IoT Software Updates Based on Trustless Proof-of-Distribution“ [Lei+18]

Leiba et al. propose a peer-to-peer system for distributing firmware updates among IoT devices, which offers greater scalability than a vendor's centralized distribution system. In this system, vendors reward distributing nodes with digital currency for transmitting software updates to other devices. To release a new update, a vendor needs to compute a file wrapping package consisting of the update file, proving key and verification key of a zk-SNARK, the vendor's digital signature of the verification key and the hash of the update file. The vendor then deploys a smart contract with a hash of the file wrapping package and enough rewards to incentivize all IoT devices to distribute the update. The file wrapping package itself is uploaded to a decentralized storage network (DSN) and nodes initially download it from there until a sufficient number of nodes have registered themselves in the DSN peer discovery scheme and can start distributing the package themselves.

To create a proof-of-delivery between a distributing node and an IoT device, the distributing node generates a symmetric key, encrypts the update file and sends the encrypted file to the IoT device. The distributing node also includes the hash of the used symmetric key, the vendor's digital signature and a zk-SNARK proof. The zk-SNARK proof proves that the hashed symmetric key can be used to decrypt the encrypted file, such that the hash of the result matches the hash of the update file in the file wrapping package. The IoT device verifies the zk-SNARK proof as well as the vendor's signature, creates itself a digital signature of the hashed symmetric key and the hash of the update file and sends it to the distributing node as a proof-of-delivery. The distributing node then submits the proof-of-delivery along with the symmetric key to the smart contract and receives their reward. The IoT device listens to events from the smart contract to receive the symmetric key, uses it to decrypt the update file and then installs the update.

„A Blockchain-based Medical Data Marketplace with Trustless Fair Exchange and Access Control“ [AN20]

Alsharif and Nabil adopted a similar strategy in their blockchain-based marketplace for medical data. When a seller intends to sell a medical record to a buyer, they initiate the process by uploading the record, encrypted under a randomly generated symmetric key, along with a hash of the record, to a DSN. Next, the seller defines an access policy for the symmetric key and utilizes attribute-based encryption and linear secret sharing techniques to compute a ciphertext. A portion of this ciphertext, together with the record's hash and a zk-SNARK proof for the encryption process, is then submitted to a smart contract. Interested buyers are able to verify the proof and deposit the required funds into the smart contract. Following this, both the seller and the buyer then engage in an off-chain protocol to conduct a fair exchange. In this exchange, the missing ciphertext component is swapped for the buyer's digital signature on the hash. The buyer's signature acts as proof-of-delivery, which the seller can present to the smart contract in order to receive payment. If the buyer's attributes satisfy the record's access policy, the buyer can decrypt the completed ciphertext and access the medical record.

3.2.4 Electricity

This section concentrates on the use of blockchain technology within the domain of decentralized electricity management. Here, a prosumer is an electricity consumer who also possesses the ability to generate power. The suggested blockchain applications serve the purpose of monitoring prosumer data and influence prosumer actions through incentives. The choices made by prosumers play a crucial role in advancing decentralized energy production and the transition to renewable energy sources.

„Blockchain and Demand Response“ [Pop+20]

Prosumers have the freedom to decide whether to store any excess energy for later use or contribute it into the grid. Demand response programs aim to encourage prosumers to assist in balancing the supply and demand of electricity on the grid by reducing their consumption and increasing their production during periods of high demand. These programs are established through mutual agreements among grid operators, energy suppliers, and prosumers, and are typically implemented using dynamic energy pricing methods. Whenever a prosumer or an aggregator on behalf of a group of prosumers is willing to modify their energy consumption or generation patterns and wishes to inform other participants in the grid, it is referred to as a flexibility request. The request usually specifies the amount of energy that the prosumer or group of prosumers is willing to adjust and the start and end time of the flexibility period.

Pop et al. propose a decentralized implementation of demand response programs on a blockchain. Their proposal requires each prosumer to preemptively report their flexibility requests for each hour of the day to an aggregator. The aggregator then inserts the prosumer's reported flexibility requests into a ZoKrates [ET18] program. This program takes the hour of the day as public input and the prosumer's monitored energy value over this hour as private input. From these inputs, it calculates the absolute difference between the flexibility request of the prosumer and their monitored energy value for a specified hour of the day. Once the program is compiled, the system deploys a verifier smart contract for each prosumer. At every hour, each prosumer submits a zk-SNARK proof to the aggregator, which then incentivizes or penalizes the prosumer depending on the degree of deviation.

„The next Stage of Green Electricity Labeling“ [SVS22]

Sedlmeir, Völter, and Strüker propose a blockchain-based solution to track and verify the labeling of green energy production and consumption. Their proposed architecture utilizes a Merkle tree to store the energy balances of prosumers' smart meters, encompassing both green and grey energy data. The Merkle tree starts with empty leaf nodes and its root is stored on the blockchain. During each update transaction, smart meters only include the previous and updated Merkle root, along with three ZKPs to ensure the integrity of the updated Merkle tree. The first ZKP validates the correctness of registration and deregistration processes. The second ZKP confirms that energy balances have been updated according to the signed sensor data from the smart meters. Lastly, the third ZKP guarantees that the total amount of generated energy is equal to or greater than the total amount of consumed energy for both green and grey labels. However, the solution is missing some system details and a proof-of-concept implementation.

3.2.5 Locations

In this section, we discuss the application of ZKPs for spatial reasoning within blockchain applications. Spatial data finds frequent application in mobility-related technologies, such as ride-sharing applications. These applications aim to validate that a user's location is within a defined area where specific services are available.

„Privacy-Preserving Traffic Management“ [Li+20]

The proposal of Li et al. considers a scenario of multiple blockchain-based traffic management systems. Vehicles and edge nodes (roadside units and toll stations) are connected via geographically-separated blockchain networks and there is a need to switch vehicles between these networks based on their location. The authors introduce a gateway between two adjacent blockchain-based traffic management systems, which is responsible for verifying the location of incoming vehicles and transferring moving vehicles from one blockchain network to another. The system uses a ZK range proof scheme to convince the gateway that the ZIP code of a location belongs into a specified range of ZIP codes. However, there is no mechanism that enforces that the location used in the proof is the current location of the moving vehicle.

„Blockchain Based Zero-Knowledge Proof of Location in IoT“ [Wu+20]

Wu et al. propose a system that enables IoT devices to prove their location to a service without revealing it. Their system model considers a blockchain network of smart access points, which issue location certificates to users after receiving a corresponding request via a short range communication technology. The user includes their longitude, latitude, and public key in a location certificate request. To authenticate the request, the user signs it and then encrypts it using the public key of a nearby access point. After decrypting the request and verifying the signature, the access point checks whether the geographic coordinates contained in the request are in the range of the used communication technology. The access point then creates a response containing a random serial number and the current timestamp, signs it and encrypts it using the user's public key. The contents of both the request and the response form a location certificate and the access point broadcasts the hash of the location certificate to the blockchain network.

To gain access to a location-based service, the user must make a request to the server and provide a zk-SNARK proof of their location. When creating the proof, the user can choose which fields in their location certificate should be public or private inputs, resulting in four levels of privacy. The first level of privacy involves no public inputs and proves only that someone has been in an area covered by an access point at some unspecified time. The second level involves the user disclosing their location as public input, but not their identity or the time they were present there. The third level includes the time as public input and can be used for real-time tracking of unauthenticated users by location-based services. Lastly, the fourth level of privacy proves that the user was at a specific place at a specific time, revealing the contents of the user's location certificate except for its serial number.

„B-Ride“ [Baz+21]

BRide is a blockchain-based ride-sharing service that preserves the privacy of riders and drivers. The service divides supported areas into cells, allowing riders to generate ride requests while concealing precise pickup and destination locations within their respective cells. Additionally, the exact departure time is disguised within a time interval. Drivers divide their planned route into segments, also concealing locations and departure times, and monitor ride requests submitted to the blockchain to find matches with a subset of their segments. If a match is found, drivers encrypt their precise locations and departure times using the rider's public key. The resulting ciphertext, combined with the driver's bid price, is then submitted as ride offer to the blockchain. Riders can then make an informed decision by comparing the submitted offers, taking into account factors such as the planned route, bid price, and the driver's reputation.

After a successful match between a rider and a driver, the rider deploys a time-locked deposit smart contract. This contract contains a set of signed locations, including random locations as well as the actual pickup location. Within a specified timeframe, the driver has the opportunity to claim back the deposit by providing a ZK set membership proof of them arriving at one of the locations. Failure to provide such proof will result in the rider receiving the complete deposit amount. To ensure the validity of the committed location, drivers are required to obtain a signature from a nearby roadside unit. In order to prevent impersonation attacks, the rider has to prove their identity to the driver at the pickup location by digitally signing a random challenge presented by the driver.

Once the trip begins, riders must transfer an adequate amount of coins to a fare payment smart contract to cover the fare. The rider's deposit from the time-locked deposit smart contract is utilized in this process as well. At fixed time intervals throughout the ride, the driver measures the distance traveled, signs it, and shares the result with the rider. If the rider agrees with the measured distance, they add their signature and submit it to the fare payment smart contract. As a result, the driver receives payment for the distance traveled since the last submission. This system prevents scenarios where the driver transports the rider for the entire trip without receiving subsequent payment, and it also safeguards against any false claims by the driver regarding a greater distance covered. If the driver fails to complete the trip within a pre-agreed time specified in their offer, the remaining payment can be refunded back to the rider's account.

3.2.6 Supply Chains

Within this section, we delve into the utilization of blockchain technology for supply chain tracking. The potential of blockchain in this context lies in its ability to provide transparency, traceability, and enhanced security when it comes to tracking products within the supply chain. Incorporating ZKPs plays a crucial role in safeguarding confidential supply chain data, preventing its exposure to unauthorized parties.

„Enabling Privacy and Traceability in Supply Chains Using Blockchain and Zero Knowledge Proofs“ [SSD20]

Sahai, Singh, and Dayama present a model for supply chains using blockchain technology, where typical supply chain procedures are represented as documents. These documents capture various stages such as product entry, transfer between sender and recipient, and processing for transforming existing products into new ones. Additionally, the model allows for splitting a product across multiple documents, merging multiple documents referencing the same product type, and includes a specific document type for the exit of a product from the supply chain. A supply chain is modelled as a graph of documents of different process types, where each document references its predecessors and contains process information like product type, quantity, sender, and receiver.

In the implementation by Sahai, Singh, and Dayama, each document is assigned a unique identifier, and the process details within the document are substituted with a Merkle root. The underlying Merkle tree is constructed by organizing product type, quantity, sender, receiver, and additional auxiliary information into a list. Additionally, documents are assigned a cryptographic accumulator for their set of predecessors, enabling efficient proofs that a specific document is not part of a path leading to the document of a given final product. These documents are stored within smart contracts on Hyperledger Fabric[3] and updated through transactions, where each transaction includes the process type, IDs of incoming and outgoing documents, the Merkle roots of all outgoing document, and zk-SNARK proofs. The zk-SNARK proofs are specific to the process type and ensure the validity of the relationship between the process information of incoming and outgoing documents. The implementation achieves transaction anonymity through an anonymous authentication protocol that is built into Hyperledger Fabric.

3.2.7 Voting

This section focuses on proposals for blockchain-based voting protocols. Compared to traditional voting protocols, where a trusted authority computes the tally once all ballots were cast, blockchain-based voting protocols are usually self-tallying and allow anybody to perform the tally computation [KY02].

[3]`https://www.hyperledger.org/use/fabric`

„How to Vote Privately Using Bitcoin“ [ZC16]

In their publication, Zhao and Chan propose a self-tallying voting protocol designed for the Bitcoin blockchain. This protocol allows a group of voters to fund exactly one candidate out of two options and consists of two phases. In the initial vote commitment phase, each of the n voters generates n random numbers, one for each participant, with their sum being equal to zero. Subsequently, they commit to these numbers and submit their commitments on-chain, accompanied by a ZKP that the sum of the committed numbers is zero. Each voter then sends to each other voter the random number associated with the other voter. By summing up all the received numbers and the random number corresponding to themselves, each voter obtains their voting key. The voters proceed to compute their ballot by adding the voting key to their binary vote. They then create commitments to their voting key and the ballot, broadcasting them to the network. Finally, the voters provide ZKPs that demonstrate the correct computation of their voting key and the binary nature of the difference between the committed values. In the second phase, voters reveal their ballot, and, because the sum of all voting keys is zero, the tally is calculated as the sum of all ballots.

„A Smart Contract for Boardroom Voting with Maximum Voter Privacy“ [MSH17]

McCorry, Shahandashti, and Hao present an Ethereum implementation of the self-tallying Open-Vote network protocol [HRZ10] designed for small-scale boardroom voting. In the first round, all eligible voters submit their public key to a smart contract, along with a non-interactive zero-knowledge (NIZK) Schnorr proof to prove their knowledge of the corresponding private key. In the second round, voters submit their ballot and prove that the contained vote is a binary value using a ZKP. The construction of the ballots involves the voter's vote and secret key, as well as the public keys of all other voters. Once all voters have cast their ballots, the tally is computed by taking the product of all ballots, which reveals the number of votes with a binary value of 1. The tally computation can thus be carried out within the smart contract without revealing any individual votes.

„Community Proposal: Semaphore: Zero-Knowledge Signaling on Ethereum“ [GJ20]

The Semaphore proposal put forth by the Ethereum community is a generic protocol that enables users to prove their membership in a group and broadcast signals on various topics like votes and endorsements without exposing their identity. Groups are represented by Merkle trees consisting of commitments to user identity information. A user proves their membership of the group by constructing a zk-SNARK proof that their identity information commitment is a leaf node in the Merkle tree. To prevent multiple broadcasts of signals, the protocol requires that each zk-SNARK proof contains a unique number (nullifier) as public input, which is derived from the user's identity information and the topic The verifier smart contract records the nullifiers that have been submitted and ensures that they cannot be used more than once.

„Chaintegrity“ [ZWX20]

Chaintegrity is a blockchain-based voting system that is specifically designed to accommodate large-scale voting. It utilizes efficient data structures, such as encoding the tally as a base-n number. Here, n is a power-of-two greater than the total number of voters, and each digit within the encoded tally represents the number of votes received by an individual candidate. The voting process involves voters selecting their preferred candidate and encrypting the corresponding power-of-n value using a homomorphic encryption scheme. To ensure the validity of the submitted vote, voters generate a ZKP that the submitted vote is indeed a power-of-n with a positive exponent that is smaller than the total number of candidates. The authenticity of ballots is established through blind signatures obtained from the election holders. Once all ballots are collected, a smart contract publishes the encrypted tally, which is the sum of all ballots. On the opening day, the election holders decrypt the tally and publish the results.

„Efficient, Coercion-free and Universally Verifiable Blockchain-based Voting“ [Dim20]

Coercion resistance is a property of electronic voting systems that prevents voters from creating a proof of their vote to someone trying to influence or manipulate their decision. Dimitriou presents a blockchain-based voting system that achieves coercion resistance using a tamper-resistant device called a token randomizer. The token randomizer serves an intermediary between the voter and the blockchain and hides its internal state from the voter. During the registration phase, the token randomizer generates random values s and r and computes a Pederson commitment to s using randomness r. The voter then signs this commitment and the tripel consisting of the voter's identity, Pederson commitment, and signature is submitted to the blockchain. In the voting period, voters submit their vote to the token randomizer, which encrypts it using a randomly generated symmetric key. The encrypted vote is then posted to the blockchain, along with the nullifier s and a zk-SNARK proof. The zk-SNARK proof ensures that the token randomizer knows the randomness r of a Pedersen commitment to s, which belongs to the set of previously registered Pedersen commitments. Once the election authorities announce the end of the voting period, token randomizers reveal the symmetric keys used to encrypt the votes, and the blockchain computes the tally. Using a token randomizer, it becomes possible to erase the random value r associated with a vote, thereby eliminating any traces of a voter participating in the election.

„A Smart Contract System for Decentralized Borda Count Voting“ [Pan+20]

The Borda count voting system involves voters ranking candidates based on their preferences, with each candidate receiving points corresponding to their ranking position. The candidate with the highest sum of points across all voters is declared the winner. Panja et al. propose a smart contract system for implementing the Borda count voting method, where voters initially submit their voting keys following the Open-Vote network protocol [HRZ10]. Given a list of candidates with each candidate having a fixed position, a vote is represented by a permutation of the points assigned to each rank. In the second round of the protocol, voters submit their ballot as a list of encrypted points and create a NIZK proof for each available rank, proving that the corresponding points are contained in the ballot. Once all ballots have been cast, the tally for a candidate is computed by multiplying all encrypted votes at the candidate's position in the ballots.

3.3 Process Execution on Blockchain

In this section, we focus on scientific literature that covers approaches to the execution of business processes on blockchain technology. The surveyed approaches receive a process model and coordinate participants to execute tasks or trigger events within the process. Through this literature review, we aim to gain insight into the data formats used for representing processes, instances and data, along with the interaction between other software artifacts and the blockchain. To gather related literature, we used the systematic literature review by Stiehle and Weber [SW22] and the following boolean search query.

```
( blockchain OR "smart contract" )
AND ( process OR choreography OR workflow )
```

3.3.1 „Untrusted Business Process Monitoring and Execution Using Blockchain“ [Web+16a; Web+16b]

Weber et al. were the first to propose a blockchain-based system for monitoring and execution of BPMN choreographies. Their approach involves a translator component that operates during the design phase and converts a choreography into a factory smart contract. The main purpose of the factory smart contract is to configure and deploy smart contracts for individual instances of the choreography. The instance-specific smart contracts maintain the process execution status and the assignments of roles to public keys. Additionally, they provide transition functions for the start event and each task within the choreography. The authors refer to these instance-specific smart contracts as choreography monitors. Furthermore, the authors suggest mediator smart contracts as extensions of choreography monitors, which provide additional functionality for message sending and receiving, as well as data transformation. Depending on the specification in the choreography, the translator component generates a factory smart contract for either choreography monitors or mediators.

Within choreography monitors, the process execution status is stored using boolean variables, which represent whether tasks, incoming edges of AND join gateways, and the end event are activated. Whenever a task has been completed, the receiver of the message invokes the corresponding transition function in the choreography monitor. The transition function validates the role of the caller and performs a conformance check by validating the activation status of the respective task. If both checks are successful, the task and any preceding elements are deactivated and subsequent elements in the choreography are activated. The activation of tasks that immediately follow AND join gateways is handled internally, removing the need to call a separate transition function for AND join gateways.

In the proposed architecture by Weber et al., each participant operates a middleware called trigger to handle communication with other participants, smart contracts, and external APIs. The trigger securely stores cryptographic keys on behalf of the participant, which are used for encrypting messages and authenticating transactions. Before the execution of a choreography, the trigger registers the participant's role and public key with the factory smart contract. In return, it receives the roles and public keys of all other participants, and the address of the newly deployed choreography monitor.

When a participant wants to send a message to another participant, their trigger encrypts the message using the recipient's public key and uploads the encrypted result to a DSN. Subsequently, the trigger provides the uniform resource identifier (URI) of the encrypted file, along with its hash to a specific function in the choreography monitor. This function logs both the URI and the hash in a blockchain event. On the recipient's side, their trigger actively listens for these events, retrieves the encrypted file using the provided URI, decrypts it, and verifies its integrity using the hash. Additionally, the recipient's trigger validates that the sender holds the appropriate role before invoking the transition function associated with the corresponding task in the choreography.

3.3.2 „Optimized Execution of Business Processes on Blockchain“ [Gar+17]

García-Bañuelos et al. propose an optimized version of the architecture by Weber et al. [Web+16a]. In the first step of the translation phase, a BPMN process model is converted into a Petri net, followed by the application of well-established reduction techniques to eliminate silent transitions (see Section 2.2.2). However, not all silent transitions can be eliminated, because some are required to check conditions attached to decision gateways in the BPMN model. All remaining transitions in the Petri net are then assigned a guard that represents a conjunction of conditions. These conditions are collected from sequence flows originating from XOR split gateways in the BPMN model by following a path from the start event to the respective task.

The resulting Petri net is converted into a smart contract that stores the process execution status as two bit vectors. The first bit vector maintains the number of tokens of the places within the Petri net, and the second bit vector stores the current truth values of the guards attached to the transitions in the Petri net. The smart contract provides public functions for regular tasks, and private functions for script tasks. Public functions, which are invoked by process participants, include checks to ensure that the corresponding transition is truly enabled. All functions within the smart contract, including the public ones, contain code that evaluates the conditions specified in the guard before invoking an internal step function with the token counts and condition values after the firing of the respective transition. After the transition has fired, the step function iterates through silent transitions and transitions associated with script tasks and fires all enabled transitions until no further transition can be fired.

Because bit vectors can be encoded as integers, the storage requirements on Ethereum are decreased compared to when boolean variables are used. To further reduce the execution cost, García-Bañuelos et al. give the option of combining the factory and instance smart contracts of Weber et al. into a single smart contract. This smart contract stores arrays of bit vectors and thus supports the parallel execution of multiple process instances by the same participants, thereby eliminating the cost of additional smart contract deployments.

3.3.3 „Inter-Organizational Business Processes Managed by Blockchain“ [NMK18]

Nakamura, Miyamoto, and Kudo introduce an approach for converting BPMN process models with a single pool into statecharts [Har87], a visual representation used for finite-state machines. The global state of process execution on the blockchain and the local process execution state of each process participant are represented using a set of statecharts. State transitions within a statechart are triggered by events and can also dispatch additional events upon completion, which is used to model the communication between different statecharts. Similar to the approach taken by Weber et al. [Web+16a], the blockchain serves as an intermediary for all communication among process participants to guarantee its consistency.

The authors define algorithms that traverse the process model and generate statecharts based on the object types. Activities and XOR join gateways are transformed into two state transitions: one establishing dependencies on predecessor states and another dispatching events for successor states. XOR split gateways are converted into a single state transition for predecessor states and one state transition for each decision option, dispatching events for corresponding successor states. AND gateways are not supported. Following the initial generation, statecharts are reduced by removing events, states, and state transitions associated with activities and gateways that do not involve the specific process participant. Through this technique, the authors successfully reduce communication overhead, resulting in an average elimination of 68% of dispatched and received events.

Additionally, Nakamura, Miyamoto, and Kudo present techniques for implementing the resulting statecharts as software artifacts. The execution of the blockchain statechart takes place on a statechart engine deployed on the blockchain, while the participant statecharts are implemented as web applications. Each participant is provided with a state transition table generated from their respective statechart and deploys a web application that incorporates this table. The frontend of the web application displays the current process state and includes interactive buttons for dispatching events according to the state transition table. The Node.js backend of the web application offers APIs that enable interaction between the frontend and the blockchain.

3.3.4 „Confidential Business Process Execution on Blockchain“ [CRF18]

Carminati, Rondanini, and Ferrari investigate a technique for preserving the confidentiality of process data while executing interorganizational business processes on blockchain. In their proposed system, participating organizations provide services that perform process activities, and the blockchain leverages oracles to invoke these services according to a business process execution language (BPEL) specification. The system transforms a BPEL process into a confidential smart contract, where the contents of all variables are encrypted using the public keys of the services that need to consume the variables.

In every service invocation, the confidential smart contract passes the service's inputs, which are encrypted using the service's public key, and an additional header parameter. The header parameter contains information on which other services depend on the output of the invoked service and whether the output is used for computations within the smart contract. The former determines which public keys the invoked service needs to use, while the latter determines whether a homomorphic encryption algorithm is required or if normal encryption suffices.

Upon receiving the encrypted inputs, the invoked service decrypts them, performs the necessary computation, and uses the header parameter to correctly encrypt its result. The oracle then transmits the encrypted result to the corresponding callback within the confidential smart contract, hereby triggering the next process steps. However, it should be noted that variables in confidential smart contracts, which are used beyond merely exchanging values between services or performing simple additions or multiplications, cannot be encrypted.

3.3.5 „Modeling and Enforcing Blockchain-Based Choreographies“ [LWW19]

In their paper, Ladleif, Weske, and Weber propose an extension of BPMN choreography diagrams that maintains backward compatibility. This extension incorporates two modeling elements from other BPMN diagrams, namely data objects and script tasks, and adds ownership semantics to all modeling elements. According to these ownership semantics, all choreography elements except for choreography tasks are owned by the smart contract orchestrating the BPMN choreography and choreography tasks are owned by the process participant who sends the corresponding message.

The authors also introduce a shared data model that is stored on the blockchain and is accessible to all process participants. This shared data model encompasses messages, which are included in transactions triggered by participants, and data objects, which are managed and owned by a smart contract. To limit the visibility of data objects and messages to specific participants, the choreography can be divided into sub-choreographies, each referencing the participants included within its scope. Conditions of data-based exclusive gateways and script tasks may be specified in code, which is then inserted into the smart contract and evaluated using shared data in the corresponding scope as input.

The architecture suggested by Weber et al. [Web+16a] serves as the foundation for their prototype implementation. In this approach, the BPMN choreography is translated into a smart contract that stores the process execution state and another smart contract for the participant registry. Before executing the choreography, each participant submits their address along with their roles in the choreography to this registry. During the choreography, the choreography smart contract uses the registry smart contract to authorize transactions from participants. Participants submit transactions at the beginning of the choreography and for each sent message, which triggers the execution of subsequent choreography elements in the smart contract. Similar to the approach by García-Bañuelos et al. [Gar+17], the control flow is represented by propagating tokens through the model.

3.3.6 „Interpreted Execution of Business Process Models on Blockchain“ [Lóp+19b]

Building upon their earlier work known as „Caterpillar“ [Lóp+19a], which is founded on the research of Weber et al. [Web+16a], López-Pintado et al. extend the proposal by introducing an interpreter for BPMN process models implemented as smart contracts. The interpreter can be attached to multiple process models and supports multiple instantiations per model. Once a process model is instantiated, participants have the ability to monitor the process instance and execute associated tasks through the functions provided by the smart contracts. The interpreted execution of processes enables participants to update their process models, even while they are being executed.

The system consists of smart contracts that implement standard operations for any process model, and there are smart contracts called „IFLOW“ and „IDATA“ that contain process-specific information for each individual process model. For every process and sub-process within a process model, an IFLOW smart contract is generated, while for each process instance, an additional IDATA smart contract is created. Each IDATA smart contract is linked to the IFLOW smart contract of the corresponding process or sub-process. Both the IFLOW and IDATA smart contracts manage references to the IFLOW and IDATA smart contracts of sub-processes, respectively.

The IFLOW smart contract provides functions to manage process elements of the corresponding process, including functions for modifying the incoming and outgoing arcs and the types of process elements. In contrast, the IDATA smart contract stores the process state by keeping track of the token counts for each process element. Additionally, the IDATA smart contract provides functions for executing script tasks and functions for external actors to perform the check-in/check-out operations for tasks. External actors use the check-in function to send data to the process instance and signal the completion of a task, whereas the check-out function is used to request data from the process instance.

When a participant triggers the check-in function of the IDATA smart contract, the IDATA smart contract, in turn, invokes the interpreter smart contract using the unique index of the completed task within the process. Starting with the elements following this task, the interpreter proceeds to execute all enabled elements by creating and destroying tokens until it encounters elements that require external interaction. Instances of sub-processes are created by deploying new IDATA smart contracts and are eventually terminated after propagating the corresponding end event to the parent process.

3.3.7 „Data-Driven Process Choreography Execution on the Blockchain“ [Lic+20]

Lichtenstein et al. model instances of BPMN choreographies as data objects, with the primary objective of minimizing storage expenses through the reuse of data objects from previously executed choreographies. In this model, each message exchanged between participants advances the state of exactly one data object. These data object are stored as separate smart contract on the blockchain, with their state represented by variables of the respective data type. To ensure the correctness of the control flow, the authors introduce pre- and postconditions for choreography tasks that have to be fulfilled during choreography execution.

Rather than directly interacting with the data object smart contracts, each participant in the choreography receives access to an interface smart contract that contains functions representing all the possible messages the participant can send. The interface smart contracts also contain a reference to another smart contract that serves as a data object store. When a participant wishes to send a message, they invoke the corresponding function in their interface smart contract. The function call prompts the interface smart contract to request the data object from the data object store and subsequently call the respective state-changing function within the data object.

The data object store employs a mapping of string identifiers to smart contract addresses for storing data objects. Identifiers for data objects of the same choreography follow a specific schema, which enables the reusability of data objects for new choreography instances. Furthermore, the mapping allows the data object store to accommodate external data objects that may be specified by a choreography. However, the approach by Lichtenstein et al. does not consider role-based access control and is only efficient when choreographies are executed multiple times.

3.3.8 „Runtime Verification for Business Processes Utilizing the Bitcoin Blockchain“ [Pry+20]

Prybila et al. suggest using tokens for monitoring the execution of BPMN choreographies on the Bitcoin blockchain. To initiate the choreography, the process owner generates a process token, which is passed between participants as the choreography progresses. Possessing a token grants the owner the authority to perform choreography tasks, thus enabling the addition of choreography participants during an ongoing execution.

Token handovers are recorded within Bitcoin transactions, which consume a process token from the token holder and produce a process token for the token receiver. At the beginning of the token handover process, the token holder and the token receiver verify their identities with the aid of a public key infrastructure (PKI) before negotiating the terms of the handover, which involves sharing process data and the choreography model.

Following this, both the token holder and token use the PKI to sign their respective Bitcoin addresses and exchange them, enabling them to use the corresponding private keys for future signing purposes. The token holder creates a handover transaction template, signs it, and sends it to the token receiver to obtain their signature. After validating the token receiver's signature for their Bitcoin address, the token holder incorporates it into the transaction template and subsequently publishes the transaction.

Should the token holder fail to publish the transaction in time, the token receiver has the option to report the signed transaction template to the process owner for resolution. The process owner actively monitors blockchain handover transactions and may request the identity of the current token holder from the previous holder after each transaction. At the initiation of the choreography, the process owner supplies sufficient Bitcoins with the process token to cover all transaction fees for the entire process execution. This methodology preserves the anonymity of process participants, as they are not obligated to spend their own potentially traceable Bitcoin.

Alongside handover transactions, the authors suggest start, end, split and join transactions to represent all control flow patterns in BPMN choreographies. Start and end transactions are published by the process owner, where start transactions create a process token from Bitcoin inputs, and end transactions destroy a process token to receive unspent Bitcoin. Split transactions consume a single process token from the token holder and produce multiple process tokens for them, while join transactions reverse this process by exchanging multiple process tokens for a single one. All recorded transactions include a process identifier and a timestamp as data output. Additionally, start, end, split and join transactions include a special marker as data output to identify the transaction type. Handover transactions include a hash of the current process data as data input, and the data output contains the identifier for the next task and the token receiver's signature.

3.4 Zero-Knowledge Proofs for Conformance Checking

The final literature review explores the application of ZKPs in the context of conformance checking. We include approaches that assess the alignment of an entire process trace with a process model, as well as methods that focus on validating the correctness of individual process steps. Since ZKPs typically prove statements that are given as arithmetic circuit, our goal in this review is to understand the arithmetic circuits used for verifying the correctness of a process execution. To compile literature related to the use of ZKPs for conformance checking, we employed a specific search query.

```
  ( zero-knowledge )
AND ( process OR choreography OR workflow OR bpmn )
```

3.4.1 „Zero-Knowledge Proofs for Business Processes“ [Too20]

In their master thesis, Toots proposes to use ZKPs to prove knowledge of a trace within a business process that also conforms to a regular expression. They argue that this methodology could be valuable for white hat hackers aiming to demonstrate the existence of flaws in a business process, considering that the unwanted behavior can be represented by a regular expression. The proposal is implemented as a web application that allows users to generate proving parameters, run the prover, or verify previously generated proofs.

Before constructing a conformance proof for a trace, the BPMN process model is initially transformed into a Petri net. In Petri net transitions, token changes can be represented by elements of the set

$$\mathcal{N} = (\{-1, +1\} \times T) \cup \{(0, -1)\}, \tag{3.1}$$

where T is the set of all places in the Petri net and the element $(0, -1)$ signals no change in tokens. By restricting the number of incoming and outgoing sequence flows of gateways in the process model to a maximum of two, each transition of the corresponding Petri net can be represented by exactly three elements of $\mathcal{N}$. A list of elements from $\mathcal{N}^3$ is given to the prover as public input and they also receive the trace as private input. For each transition in the trace, the prover fist creates a vector with as many elements as places in the Petri net, where each element represents the number of changed tokens at that place. Next, the prover inserts these vectors in order of the transitions in the trace into an array. By calculating the prefix sums over this array, the prover obtains an array containing the token count at each place after every transition. Finally, the prover checks that the resulting array does not contain negative values, thus proving the trace's conformance to the process model.

To prove that a trace matches a regular expression, a deterministic finite automaton (DFA) is derived from the regular expression. Subsequently, the DFA is executed on the trace to generate a sequence of transitions. The prover receives the trace and the indices of these transitions in the transitions array of the DFA as private input, while the DFA itself is given as public input. First, the prover reads the transitions array using the provided indices to retrieve the executed transitions. They then validate that the inputs of these transitions match the elements in the trace, that the initial transition originates from the initial state of the DFA and that the last transition leads to a final state. Lastly, the prover checks that the executed transitions represent a valid path through the DFA.

3.4.2 „Blockchain-Based, Confidentiality-Preserving Orchestration of Collaborative Workflows“ [TK23]

Toldi and Kocsis propose to store the execution state of BPMN processes as commitment inside a smart contract. Participants are then allowed to submit state updates if they include a ZKP ensuring the validity of the update. The validity of these updates is checked by a ZoKrates program, called zkWF program, which is derived from the process model in the modelling phase.

The authors introduce two different annotations for BPMN process models. The first annotation `zkp:publicKey` assigns public keys to pools or lanes to distinguish between tasks of different participants. The second annotation `zkp:variables` is applied to activities and signals that a specific global variable may be written by the annotated activity. While only specific activities may write global variables, all activities can read them. Global variables can also be used in boolean expressions for exclusive gateways.

The execution state of a process consists of the current values of global variables, hashes of messages that have been sent during the process, and a vector that contains the execution status for each executable element in the process model. This status is denoted as 0 for Inactive, 1 for Active, or 2 for Completed. Smart contracts store a commitment to this execution state, and also the state itself encrypted under a key that is shared between all participants in the process.

Each participant aiming to advance the process must provide a ZKP of correctly executing the zkWF program specific to the process. The private inputs to the zkWF program include the current and updated execution states, random values used for the corresponding state commitments, and the participant's public and private keys. The public inputs consist of the current state commitment and the participant's signature on the current and updated state commitments. The zkWF program validates the current state commitment, the BPMN state change and the participant's authorization. The participant's authorization requires verification of the given signature and checking that model elements involved in the state update are annotated with the same public key. If all validations are successful, the zkWF program generates the updated state commitment, intended to replace the current state commitment within the smart contract.

In order to ensure practical proof computation times, zkWF programs do not validate the correctness of encrypted data that is stored within the smart contract. In turn, the signature provided to zkWF programs as public input may be used to identify participants submitting encrypted data that is incorrect. To further reduce proof computation times, in addition to reducing storage requirements, Toldi and Kocsis impose an upper limit of two incoming or outgoing edges on gateways. By restricting the maximum edge count on gateways to three, it becomes possible to represent state transitions as elements of $\mathcal{N}^3$ (refer to equation 3.1), a concept initially introduced by Toots [Too20].

The zkWF program validates the BPMN state change between two execution states by first comparing the vectors containing the execution status of all executable model elements. If both are the same, the state change is immediately accepted, which allows participants to submit fake updates to the execution state. If the vectors differ by more than three elements, the state change is immediately rejected. Otherwise, the zkWF program computes the state transition $\mathcal{N}^3$, which has to be applied to the current vector to obtain the updated vector. If $\mathcal{N}^3$ is contained in the set of possible state transitions, the vector change is accepted.

Because activities that are executed in parallel can be finished in any order, parallel join gateways require an extra verification step to confirm the completion of both preceding activities. For exclusive split gateways, the zkWF program evaluates boolean expressions placed on path options and rejects state changes with evaluations to false. State change validation also includes checks for write permissions to global variables and message-handling is validated by checking whether message senders incorporate message hashes in the state. Messages are sent through off-chain channels, whereas message receivers have to compare received messages against hashes in the execution state.

3.5 Summary

In this chapter, we conducted a comprehensive review of scientific literature in the areas we identified as relevant to this thesis. The section on the use of ZKPs within blockchain technology provides insights into enhancing transaction privacy. Furthermore, it delves into approaches that achieve general private computations on blockchains. Importantly, some of these approaches can potentially be used to execute business process on the blockchain, as long as the business processes are appropriately represented within the framework of these methodologies.

The section covering the application of ZKPs within blockchain applications explores a broad spectrum of use cases. It highlights multiple strategies for integrating ZKPs into larger protocols, with the most prevalent approach being the commitment of data on the blockchain, followed by the use of ZKPs to prove statements about this data. Commonly proven statements involve a user's knowledge of a hash preimage, the validity of digital signatures, and the inclusion of elements within sets via Merkle proofs. Due to the costs associated with on-chain data storage, some approaches store their data on a DSN.

Furthermore, we performed a literature review on the execution of business processes on the blockchain. In most cases, the surveyed approaches compile a process model in BPMN into one or more smart contracts. Upon completing a task or triggering an event, participants subsequently call the corresponding method within the smart contract to advance the process according to the process model. Process data is categorized as either publicly visible or intended for a specific participant. In cases where data is meant for a particular participant, it is encrypted using that participant's public key.

Lastly, we explored methods related to the use of ZKPs for conformance checking, which verify the alignment of entire process traces or individual process steps with process models. In all covered approaches, the process model is transformed into a Petri net, which is represented as a set of transitions. Each transition contains exactly three token changes, whereas transitions with one incoming and one outgoing arc include an element indicating no token change. Toots [Too20] represent execution states as token quantities at each place, whereas Toldi and Kocsis [TK23] categorize transitions during execution into Inactive, Active, and Completed. Furthermore, the latter approach incorporates the current values of global variables and the hashes of previously sent messages into the execution state. It's also worth noting that while the former approach is primarily concerned with conformance checking, the latter approach enables the execution of processes on the blockchain.

CHAPTER 4

Requirements

This chapter aims to answer the research question:

RQ1 Which are the requirements of a system that enables the confidential execution of business processes on blockchain and uses ZKPs to enable public verification of process executions?

The taxonomy of blockchain-based process enactment by Stiehle and Weber [SW22] serves as a starting point for identifying the requirements of our system. We further expand on these by investigating systems from the scientific literature discussed in the previous chapter.

4.1 Functional Requirements

These section explores functional requirements of our system across various capabilities that are useful for process enactment.

4.1.1 Collaborative Execution

The main requirement of the proposed system is to facilitate the execution of collaborative business processes and enforce their correctness in real-time. In the BPM lifecycle (depicted in Figure 2.6), its primary utility lies is in the implementation and monitoring phases.

- Participants have to be able to instantiate a process model, notify the system when they complete a task within the process, send messages to other participants and terminate process instances.

- Following such an operation, the system is responsible for transmitting the updated process state to participants in charge of the next task.
- The communication between participants falls outside the scope of our approach and is entrusted to a separate messaging protocol, such as libp2p[1] or Swarm[2].

4.1.2 Diagram Support

Applications that facilitate the execution of business processes by individuals or groups accept business process models as their input. Most of the approaches in literature focuses on the BPMN, but there is no clear favorite between the different types of BPMN diagrams. The process diagram is the most supported diagram type followed by the choreography diagram [SW22]. In collaborative processes, process diagrams depict the participant's internal processes within pools and swim lanes, while choreography diagrams exclusively show interactions among participants [Cor+18].

- Both the BPMN process and choreography diagrams are widely supported, making them both suitable formats for providing a business process description as input to our proposed system.

4.1.3 Control-Flow Perspective

Conformance checking categorizes checking the rules imposed by process models into the control-flow, data, resource and time perspectives. When a process is executed on the blockchain, the extent to which these perspectives are taken into account may vary. Checking the control-flow perspective in business processes involves verifying the semantics of flow object, such as exclusive and parallel gateways. In the literature on blockchain-based process enactment, nearly all approaches enforce a correct control-flow [SW22].

- Our approach is required to enforce control-flow semantics of start and end events, exclusive and parallel gateways and tasks of the chosen BPMN diagram type.

4.1.4 Resource Perspective

In the realm of blockchain-based process execution, resource allocation refers to the assignment of blockchain addresses to specific tasks. As a result, the task shall only be executable by participants owning the private key associated with that address. Other than directly assigning addresses to tasks, some indirection may be introduced by assigning addresses to roles and roles to tasks. This indirection allows for the reuse of models by different sets of participants, making it the preferred method for implementing resource allocation in business processes. Enforcing the resource perspective ensures that

[1] https://libp2p.io/
[2] https://docs.ethswarm.org/

process executions adhere to the resource allocation. Surprisingly, just slightly over half of the surveyed approaches take the resource perspective into account [SW22].

- Given that neglecting the resource perspective may lead to unauthorized entities altering process executions, our proposed solution has to incorporate resource allocation.
- The system must implement role-based resource allocation to enable the reuse of processes across different groups of participants.

4.1.5 Data perspective

The data perspective in conformance checking compares process instances data to constraints defined within process models. These constraints can take different forms such as data-based gateway conditions, required data types for variables, and restrictions on which tasks are allowed to modify specific variables. In blockchain-based process enactment, data types or data-based gateway conditions are usually defined as blockchain-specific code snippets. While most approaches let participants verify the integrity of instance data, less than half of those enforcing control-flow support data-based gateway conditions [SW22].

- Enabling the verification of instance data is a requirement for our approach, while checking data-based gateway conditions is optional.

4.1.6 Time perspective

Temporal constraints in BPMN are represented through timer events, which trigger at specified times, after certain durations, or in periodic cycles. However, when implementing these constraints on the blockchain, they must rely on the block timestamp. Enforcing such time-based constraints on the blockchain presents challenges due to the uncertainty that a transaction is included in a block within a certain timeframe. In the literature, only two out of the 36 surveyed methods for blockchain-based process execution support temporal constraints [SW22].

- Given the inherent difficulties and infrequent support of the time perspective, enforcing temporal constraints is not a requirement for our system.

4.1.7 Blockchain

The synchronization of process executions shall be achieved through a blockchain. Utilizing a blockchain offers the benefit of maintaining a consistent view of the process state and ensuring that approved state updates are irreversible, once the blockchain transaction is finalized. In the context of business process execution, this property is particularly valuable for ensuring that participants cannot deny performing a task within a business process.

- The process state shall be synchronized via a blockchain.
- The blockchain shall only accept transactions with state updates that align with the process model across all mentioned perspectives.
- Participants shall be able to verify process states against those stored on the blockchain.
- Moreover, participants shall have the ability to track state updates by monitoring transactions on the blockchain.

4.1.8 Interpreted Execution

Except for two instances [Lóp+19b; Pry+20], all the methods surveyed in Section 3.3 of the previous chapter involve the compilation of process models into software artifacts. The compiled approach implies that every new process model or modification to an existing process model necessitates a separate deployment of all required artifacts.

- Considering the cost associated with smart contract deployments, the developed artifacts shall support multiple process models.

4.2 Non-functional Requirements

This section lists non-functional requirements of the proposed system.

4.2.1 Privacy

The demand for transparency in blockchain suits certain use cases, but it raises concerns among organizations reluctant to compromise the confidentiality of their business processes and associated data [LWW19].

- The process state, along with participant identities and instance data, shall only be visible to the participants involved in the process.
- Messages sent throughout the process shall only be visible to the intended recipient.
- For external observers monitoring the blockchain, the process state must be concealed through cryptographic means, and the smart contract must retain the ability to reject incorrect state updates by verifying ZKPs.

4.2.2 Prover Time

Because arithmetic circuits can represent all NP computations, general zk-SNARK protocols are used for a variety of applications. However, one of the primary obstacles preventing current zk-SNARK schemes from proving large statements is the time it takes for the prover to compute a proof [XZS22]. Most of the methods surveyed in Section 3.2 of the preceding chapter require a proving time under 15 seconds [Wan+19; GAC20; Gud+20; SSD20; AN20; YL20]. In contrast, the methods used by Toots [Too20] and Toldi and Kocsis [TK23], which utilize zk-SNARKs to prove business process conformance, necessitate approximately 40 and 120 seconds for proving time, respectively.

- Our approach aspires to significantly reduce the time required for these conformance proofs, aligning it more closely with the timeframes seen in the methods detailed in Section 3.2.

4.2.3 Execution Cost

According to the literature review by Stiehle and Weber [SW22], the assessment of costs associated with deploying blockchain artifacts on the blockchain is the most commonly employed metric in evaluating approaches for blockchain-based process execution. Specifically, in the context of Ethereum, this evaluation metric is quantified as the gas amount, for which the price has risen dramatically over the last years, and thus the execution costs of numerous previously published methods have increased significantly over time. With a projected mean cost of $1010 in 2021 for a single instance run, most recent works argue for using private blockchains for executing collaborative business processes [SW22]. To maintain financial feasibility while utilizing a public blockchain for its enhanced security, it becomes essential to minimize both the amount of stored data and the computational tasks performed on the blockchain [Web+16a].

- By verifying a zk-SNARK proof inside a smart contract, it is possible to verify arbitrary NP computations by performing just a few group operations. While zk-SNARKs contribute to cost reduction, keeping the execution cost as small as possible still represents a core focus in the design of our approach.

4.2.4 Extensibility

Considering that no single diagram type is universally favored for modeling collaborative business processes, electing a particular diagram format as an input format involves making certain trade-offs. As a result, extending our solution to accommodate additional diagram types, such as BPMN process or choreography diagrams, should be straightforward.

- The proposed system employs an intermediate representation for processes, translating processes modeled in the selected diagram type into this intermediary form.
- Integrating an additional diagram type shall not necessitate steps that are significantly more complex than providing a mapping to this internal representation.

CHAPTER 5

System Design

This chapter aims to answer the research question:

RQ2 Which aspects of the state of the art must be adopted to create a system that satisfies these requirements?

Our objective is to employ design science to address this research question, as its iterative approach proves effective in tackling the complexity of the task.

5.1 System Architecture

The proposed system architecture depicted in Figure 5.1 is designed to meet the identified requirements for executing collaborative business processes on the blockchain and consists of the following modules.

1. The Bpmn module transforms BPMN diagrams into a representation that is suitable for execution.
2. The Execution module allows participants to instantiate choreography models, announce task completion, and terminate choreography instances. After these operations, the module also returns a zk-SNARK proof of correctness.
3. The Blockchain module consists of Ethereum smart contracts that verify zk-SNARK proofs and update execution states upon successful verification.
4. The Messaging module provides messaging channels pairs of participants. However, its implementation was deemed beyond the scope of this thesis, which focuses on the use of ZKPs for confidential execution of business processes.

5. The UI module receives commands from the user and coordinates the other modules. It was not implemented in this thesis due to the substantial time resources required, which were prioritized for the development of other modules.

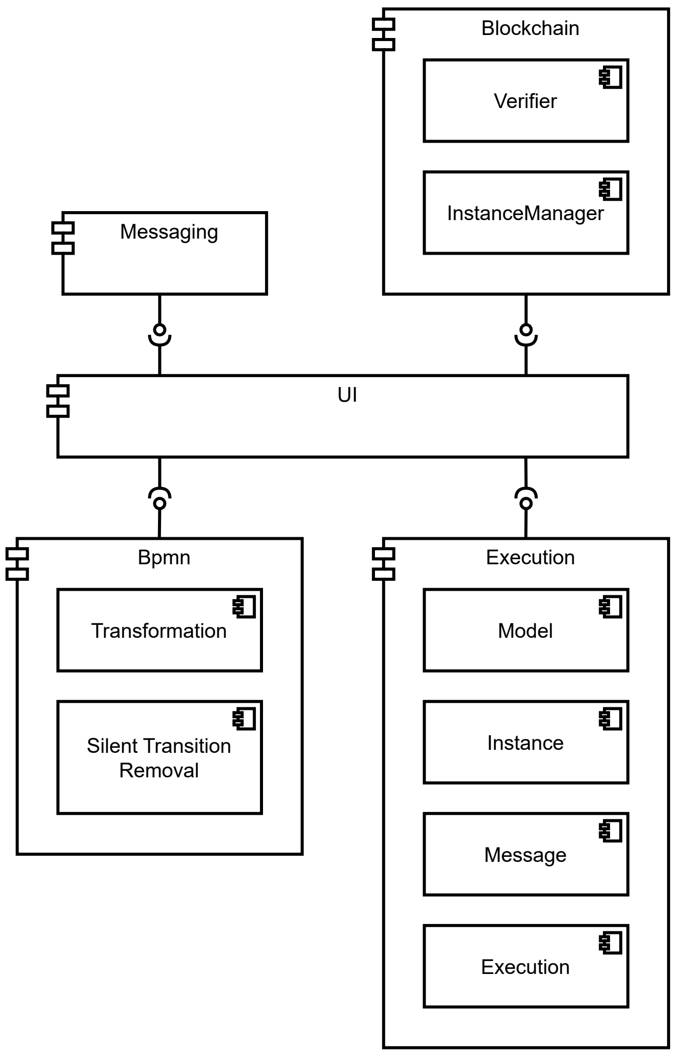

Figure 5.1: Proposed system architecture for the confidential execution of collaborative business processes on blockchain

5.1.1 Module Communication

The UI module allows users to select BPMN diagrams, which are then transformed into an executable representation by the Bpmn module and passed to the Execution module for storage. It also displays models and instances of these models stored within the Execution module. Participants interact with the UI module for instantiating models, executing transitions, or terminating instances. The UI module forwards these requests to the

Execution module, which performs the necessary operations and generates proofs. After receiving the responses, the UI module synchronizes instance data with the Blockchain module and determines whether instance data needs to be sent to the Messaging module in order to enable other participants to continue the choreography.

5.1.2 System Functions

We've categorized the system's functions into four distinct categories. The first category encompasses all tasks necessary for transforming a BPMN diagram into an equivalent model that can be executed by our system. Following the transformation, the second category includes instantiating choreography models, advancing the state of process executions and creating zk-SNARK proofs of correctness. The third category involves all tasks related to managing process execution states on the blockchain, and the last category encompasses data exchange among participants. In the following sections, we aim to offer a more detailed description of each of these categories.

5.2 BPMN Transformation

Both the BPMN process and choreography diagrams are well suited for representing collaborative business processes intended for execution on the blockchain [SW22]. In this thesis, we focus on BPMN choreography diagrams because they excel at portraying participant interactions through message exchanges, all while preserving the confidentiality of their internal processes.

To explain how we transform a BPMN choreography diagram into a form that is executable by our system, we first delve into our internal representation of choreographies. Then, we explore the mapping between BPMN choreography diagrams and this representation. Finally, we outline a technique for reducing the size of models in this representation.

5.2.1 Choreography Representation

Our internal representation for business process choreographies is based on Petri nets, since they are the most commonly used process modelling language in conformance checking literature [Dun+19]. Petri nets consist of places, transitions, arcs connecting places and transitions, and a set of initially marked places. To facilitate the execution of choreography diagrams, our proposed representation incorporates additional information.

Listing 5.1 shows our executable representation of choreographies in Golang [1]. A model is identified by a salted hash of its contents and each model defines the number of places in the Petri net, the number of participants, and the number of defined messages in the choreography. The start places correspond to the initial markings of the Petri net, while the set of end places is used to detect process completion.

[1] `https://go.dev/`

```go
type TransitionId = string
type PlaceId = uint16
type ParticipantId = uint16
type ModelMessageId = uint16

type IntegerType = int32
type ComparisonOperator = uint8

type Constraint struct {
  Coefficients       []IntegerType
  MessageIds         []ModelMessageId
  Offset             IntegerType
  ComparisonOperator ComparisonOperator
}

type Transition struct {
  Id             TransitionId
  IncomingPlaces []PlaceId
  OutgoingPlaces []PlaceId
  Sender         ParticipantId
  Recipient      ParticipantId
  Message        ModelMessageId
  Constraint     Constraint
}

type Hash struct {
  Value [HashSize]byte
}

type SaltedHash struct {
  Hash Hash
  Salt [SaltSize]byte
}

type Model struct {
  Hash             SaltedHash
  PlaceCount       uint16
  ParticipantCount uint16
  MessageCount     uint16
  StartPlaces      []PlaceId
  EndPlaces        []PlaceId
  Transitions      []Transition
}
```

Listing 5.1: Choreography representation in Golang

Transitions are identified by strings and contain references to places connected via incoming and outgoing arcs. Additionally, transitions may optionally refer to a sender, receiver, message, and include constraints. Messages can be either a sequence of bytes or a single integer, whereas only the latter may be referenced within constraints.

Constraints always compare a linear combination of integer messages to zero. Given the integer messages $m_1, \ldots, m_k$ and integer constants $a_1, \ldots, a_k, b \in \mathbb{Z}$, each of these constraints has the form

$$a_1 m_1 + \cdots + a_k m_k + b \odot 0, \tag{5.1}$$

where $\odot \in \{=, >, <, \geq, \leq\}$.

5.2.2 Transformation

Our proposed system supports start and end events, exclusive and parallel gateways, and choreography tasks among the various elements available in BPMN choreography diagrams. The translation of BPMN process diagram elements into Petri net transitions is described in Dijkman, Dumas, and Ouyang [DDO08]. Unlike regular tasks, the transformation of choreography tasks requires the ability to assign a sender, a recipient, and a message to transitions.

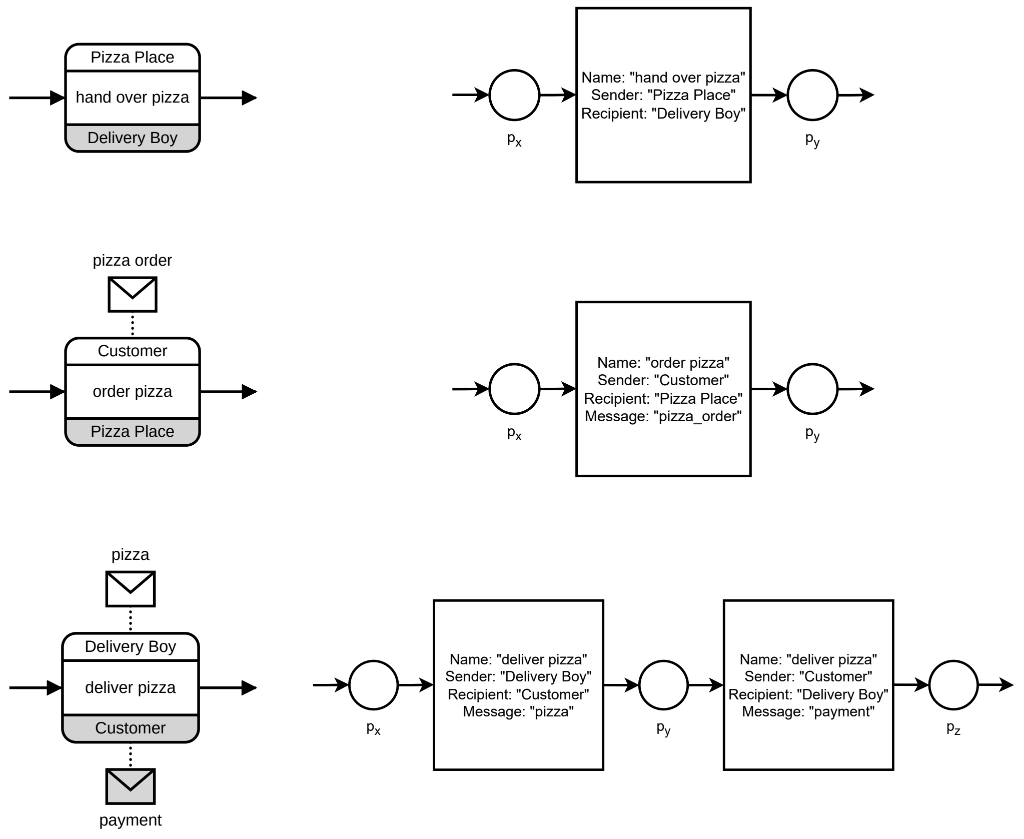

Table 5.1: Mapping from choreography tasks to model transitions

Table 5.1 illustrates how choreography tasks are translated into model transitions. In cases where choreography tasks do not incorporate any messages, a single model transition is created, with the initiating participant acting as the sender and the receiving participant as the receiver. When choreography tasks require sending a single message, this message is referenced within the resulting model transition. For choreography tasks involving both an initial message and a response message, two model transitions are created, each referencing one of the messages, and both participants take on the roles of sender and receiver. It is important to note that participant IDs and message names in the choreography are mapped to integer IDs in the resulting model.

In our methodology, a recipient's approval of a received message is mandatory for any transition involving a sent message. An alternative method involves transforming choreography tasks into two separate transitions for each message sent: one to indicate the sender's approval and another for the recipient's. However, it should be noted that this approach tends to produce larger models.

Within a choreography diagram, constraints can be added to the name field of sequence flows. If a sequence flow has a constraint, the transformation first has to parse it. In order for the parsing to succeed, each constraint must be a valid JavaScript expression that evaluates to a boolean result. This expression is required to consist of a single comparison operator and may contain additions and multiplications of message names and integers. If the parsing is successful, the constraint is included in the transition associated with the destination of the sequence flow.

Figure 5.2 shows a simple choreography in BPMN with a constraint applied to the sequence flow just before the end event. The transformation of this choreography yields the transitions seen in Figure 5.3, where the constraint is integrated into the transition associated with the end event.

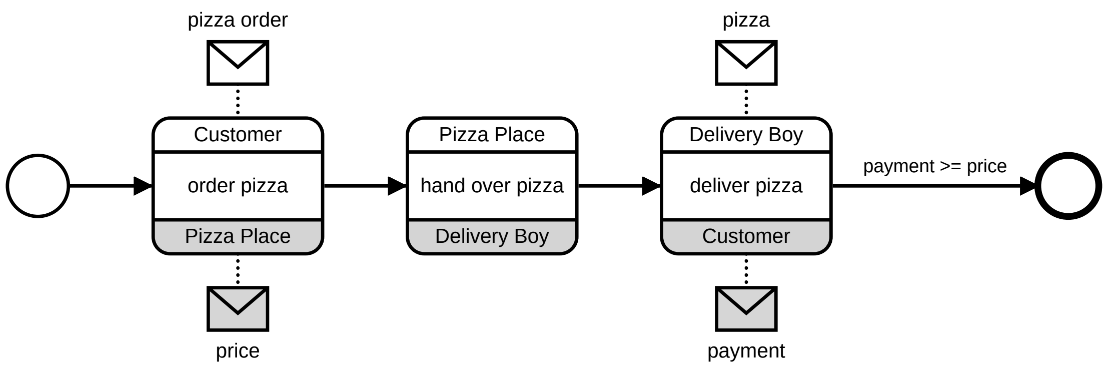

Figure 5.2: Simple choreography example

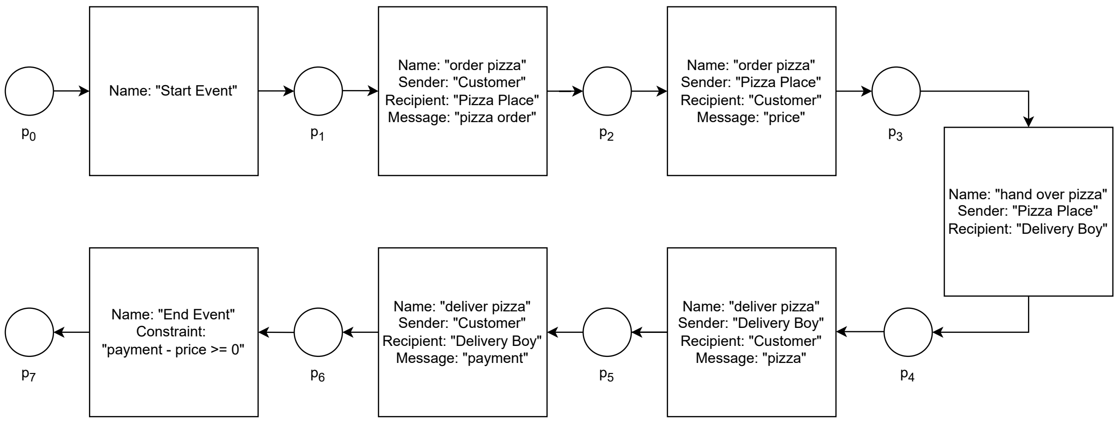

Figure 5.3: Model transitions of simple choreography example

After all choreography elements have been converted into transitions, they are then paired with place, participant, and message quantities of the choreography and assigned to the model. Finally, the model is given its ID by hashing the entire contents of the model along with a salt.

5.2.3 Silent Transition Removal

As outlined in Section 2.2.2, silent transitions are transitions that do not correspond to tasks or events within a choreography. To reduce model size, Algorithm 5.1 removes silent transitions in a subsequent step after the transformation. Once a transition is removed, the adjacent transitions are connected by replacing one set of places with another set of places in all remaining transitions. Additionally, the constraints from silent transitions are transferred to the subsequent transitions.

Algorithm 5.1 is applied to all silent transitions with different sets of places as parameters. For each transition that originates from exclusive split gateways and parallel join gateways, the algorithm replaces the outgoing places with the incoming places. Conversely, for transitions associated with parallel split gateways and exclusive join gateways, the incoming places are replaced with the outgoing places.

Algorithm 5.1: Algorithm for removing a transition from a model

```
Input: model, transitionToRemove, placesToRemove, placesToKeep
1  model.transitions ← model.transitions \ {transitionToRemove};
2  forall $t \in$ model.transitions do
3      if $|t.incomingPlaces \cap placesToRemove| > 0$ then
4          t.incomingPlaces ← t.incomingPlaces \ placesToRemove;
5          t.incomingPlaces ← t.incomingPlaces ∪ placesToKeep;
6          if transition.constraint = null then
7              transition.constraint ← transitionToRemove.constraint;
8          else if transitionToRemove.constraint ≠ null then
9              throw Exception
10         end
11     end
12     if $|t.outgoingPlaces \cap placesToRemove| > 0$ then
13         t.outgoingPlaces ← t.outgoingPlaces \ placesToRemove;
14         t.outgoingPlaces ← t.outgoingPlaces ∪ placesToKeep;
15     end
16 end
```

5.3 Model Execution and Proofs

This section focuses on the verifiable execution of choreographies after they have been converted into our executable format. We begin by outlining our internal representation of choreography instances. Subsequently, we explore the instantiation of choreography models, the effects of executing transitions on instances, and how arithmetic circuits verify the correctness of these processes. Additionally, we introduce an arithmetic circuit designed to validate the termination of an instance. Finally, we cover the creation of zk-SNARK proofs for these circuits and their verification within smart contracts.

5.3.1 Instance Representation

Our instance representation consists of the data associated with a single execution of a choreography model, and is specified in Golang in Listing 5.2. The IDs of these instances are created by hashing their contents along with a salt and each instance references its corresponding model. Instances encompass mappings of place IDs to the quantities of tokens at those specific places, participant IDs to their public keys, and message IDs to salted hashes of the messages corresponding to those IDs. As place IDs, participant IDs and message IDs are all unsigned integers, these mappings are implemented as slices.

```go
type PublicKey struct {
  Value []byte
}

type Instance struct {
  Hash          SaltedHash
  Model         Hash
  TokenCounts   []int8
  PublicKeys    []PublicKey
  MessageHashes []Hash
}

type Message struct {
  Hash           SaltedHash
  IntegerMessage IntegerType
  BytesMessage   []byte
}
```

Listing 5.2: Choreography instance representation in Golang

Message IDs are unique only within a model, while outside the model context, messages are identified using a salted hash. This salted hash is computed from the message's content, which can either be a sequence of bytes or an integer.

5.3.2 Common Circuit Elements

To ensure the correctness of computations, we must represent them as arithmetic circuits. When a single circuit is employed to check the alignment of various instances with different models, the circuit requires both the model and the instance as inputs. Arithmetic circuits have a fixed size and lack loops, so all inputs need to have a fixed length. Consequently, we establish a maximum length for all lists in models and instances and specify out-of-bounds values for cases where the list is shorter than this maximum length.

Furthermore, before models and instances can be used as inputs for an arithmetic circuit, they must be converted into elements within the scalar field of a chosen elliptic curve. Integer values of models and instances can already be regarded as field elements since they are significantly smaller than the moduli typically used in cryptography. Participant public keys must adhere to the format specified by the EdDSA signature scheme [Ber+12], comprising two field elements that represent a point on the chosen elliptic curve.

To compute hashes of models, instances and integer messages, we have chosen the MiMC hash function [Alb+16] because it produces a field element as its output and can be efficiently computed within an arithmetic circuit. It's important to note that MiMC exclusively accepts field elements as input, which is why we compute message hashes of byte messages using SHA2, followed by hashing to the selected elliptic curve [Faz+23].

Given that the list of public keys in an instance, along with the lists of transitions and end places in a model, are immutable, they are each replaced by the root of a Merkle tree containing the same values. While place IDs of end places can be directly represented as leaf nodes in the Merkle tree, public keys and transitions require MiMC hashing before they are included in the Merkle tree.

All inputs to our arithmetic circuits are private, except for the instance hashes. Each arithmetic circuit calculates the hashes of the provided model and all provided instances, then compares the resulting instance hashes with the publicly provided instance hashes, and the computed model hash with the model hashes within the instances.

Participant authentication is provided in form of a signature, a public key and a Merkle proof. Listing 5.3 shows the implementation of this concept in Golang using the Gnark library [Bot+23]. The index refers to the position of an element within the Merkle tree.

```go
type MerkleProof struct {
  MerkleProof merkle.MerkleProof
  Index       frontend.Variable
}

type Authentication struct {
  Signature   eddsa.Signature
  PublicKey   eddsa.PublicKey
  MerkleProof MerkleProof
}
```

Listing 5.3: Participant authentication modelled in Golang using Gnark

All arithmetic circuits check participant authentication by verifying the signature on an instance hash using the provided public key, verifying the Merkle proof, comparing the hash of this public key against the leaf of the Merkle proof, and comparing the root of the Merkle proof to the one stored in the instance.

5.3.3 Model Instantiation

To instantiate a model, each participant's public key is required. These public keys are provided as a mapping that associates participant IDs with their respective keys and are directly applied to a newly created instance. The mapping that connects place IDs to token quantities is initialized with a value of 1 for the start places in the model and 0 for all other places. Message hashes are initially set to a specific value for all messages defined in the model. After that the model hash is assigned to the instance and the instance hash is computed.

Figure 5.4 shows the inputs and validation steps of the arithmetic circuit responsible for verifying model instantiations. For token count validation, the instantiation circuit checks whether all start places have one token and all other valid places have zero tokens. Additionally, the circuit checks whether the token count mapping assigns the correct out-of-bounds value for place IDs equal to or greater than the total number of places in the model. A similar validation process is applied to the message hash mapping, ensuring that valid message IDs are mapped to an initial message hash and all other message IDs are mapped to an out-of-bounds hash.

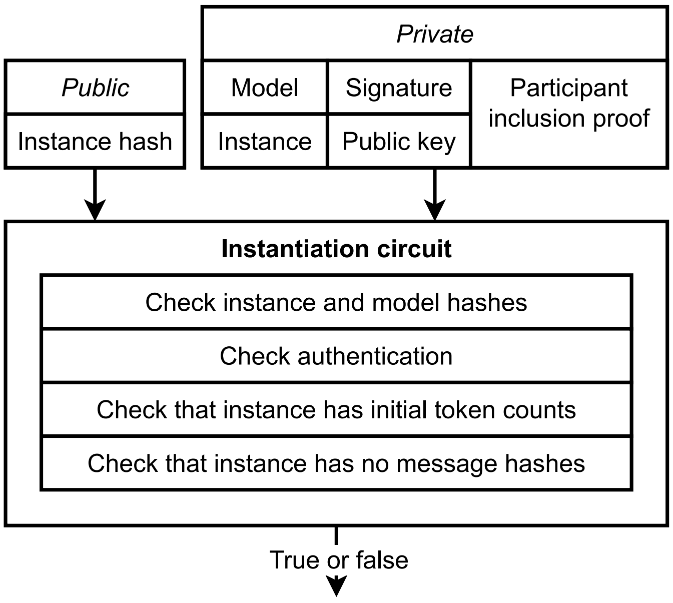
Private
Public
Model
Signature
Participant inclusion proof
Instance hash
Instance
Public key
Instantiation circuit
Check instance and model hashes
Check authentication
Check that instance has initial token counts
Check that instance has no message hashes
True or false

Figure 5.4: Inputs and validation steps of the instantiation circuit

5.3.4 Transition Execution

The execution of a transition on an instance results in the creation of a new instance with a new hash. The number of tokens at incoming places of the transition decreases by one, while the number of tokens at outgoing places increases by one. If the transition references a message, a hash is added to the message hash mapping of the new instance. Transitions that do not specify a sender or recipient can be executed by any participant, while transitions with specified sender and recipient parameters require authentication from both participants.

Figure 5.5 shows the inputs and validation steps of the transition circuit. It should be noted that the inputs for sender and recipient may be of any participant, if the transition does not specify either of them. Using the methods described in Section 5.3.2, the transition circuit begins by validating both instance hashes, the model hash, and both authentications. Furthermore, it ensures that the Merkle root for the public keys of the instance before the transition was not modified by the transition. In the following steps the circuit primarily checks the instance after the transition, as the instance before the transition has already been verified in a previous proof.

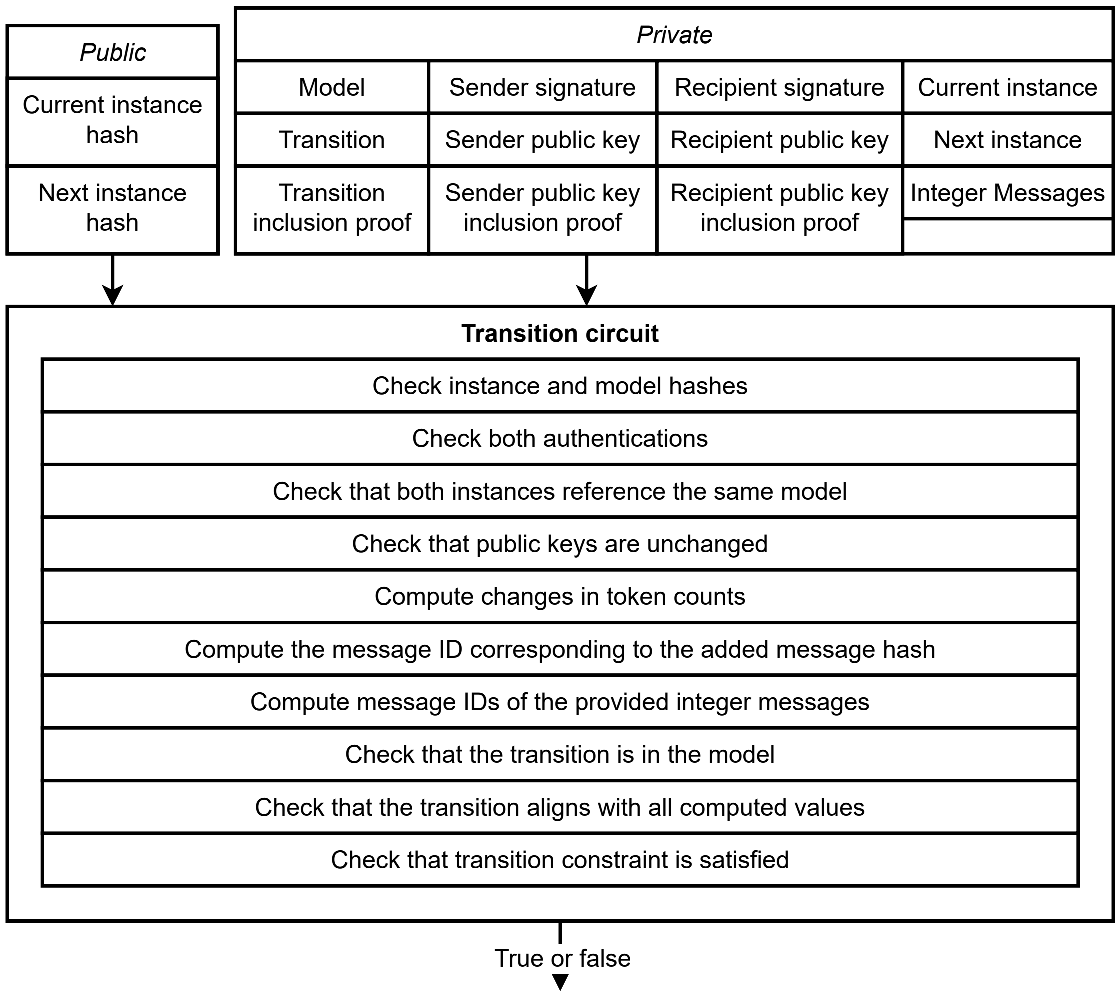
Public
Current instance hash
Next instance hash
Private
Model
Sender signature
Recipient signature
Current instance
Transition
Sender public key
Recipient public key
Next instance
Transition inclusion proof
Sender public key inclusion proof
Recipient public key inclusion proof
Integer Messages
Transition circuit
Check instance and model hashes
Check both authentications
Check that both instances reference the same model
Check that public keys are unchanged
Compute changes in token counts
Compute the message ID corresponding to the added message hash
Compute message IDs of the provided integer messages
Check that the transition is in the model
Check that the transition aligns with all computed values
Check that transition constraint is satisfied
True or false

Figure 5.5: Inputs and validation steps of the transition circuit

Next, the circuit compares the token quantities of both instances. In cases where the token count after the transition is binary, the difference in tokens between the two instances must be either zero, an increase of one, or a decrease of one. For any other number of tokens after the transition, it must match the number of tokens before the transition, which also covers out-of-bounds values. The circuit records the place IDs where token quantities were decreased by one and those where token quantities were increased by one. Subsequently, a validation checks that the counts of both types of place IDs do not exceed a previously defined maximum branching factor for models.

The circuit compares all message hashes in both instances, allowing for the modification of a maximum of one message hash. If a message hash is modified, the circuit registers the associated message ID. Additionally, the circuit computes the hash of the integer messages that are provided as the input for the transition constraint, compares the resulting hashes to those found in the pre-transition instance, and generates a list of message IDs for the provided messages.

The transition verification process begins by confirming the transition's inclusion in the model using a Merkle proof. If all previously computed changes indicate that the instance remains unaltered, except for the salt and hash, the transition verification immediately succeeds. This approach, which permits participants to submit „fake“ transitions, was inspired by the method proposed by Toldi and Kocsis [TK23].

However, if other changes are detected within the instance, the incoming and outgoing places of the transition must align with the places where the token count increases or decreases by one. Additionally, the transition's message must match the added message ID, with a specific value representing an empty message ID. If the transition specifies a sender and a recipient, both must correspond to the participant IDs within the authentications.

Besides ensuring that the transition fits the changes in the instance, the circuit also checks message IDs of the transition constraint against those computed for provided messages. It then evaluates the transition constraint on the provided messages to determine if the transition is eligible for execution. The transition verification succeeds if the transition aligns with the changes in the instance and the transition is eligible for execution.

5.3.5 Instance Termination

An instance is in a terminating state, if the token count at any of the end places of the associated model is one. It's worth highlighting that a proof of termination doesn't directly affect the execution of the choreography but can instead serve as a trigger for further actions or events.

The inputs and validation steps of the arithmetic circuit responsible for validating instance termination is shown in Figure 5.6. The end place inclusion proof is used to verify that a provided place ID is one of the end places of the model. After performing all common verification steps, the circuit checks this Merkle proof and ensures that the token count for the specified place ID within the proof is equal to one.

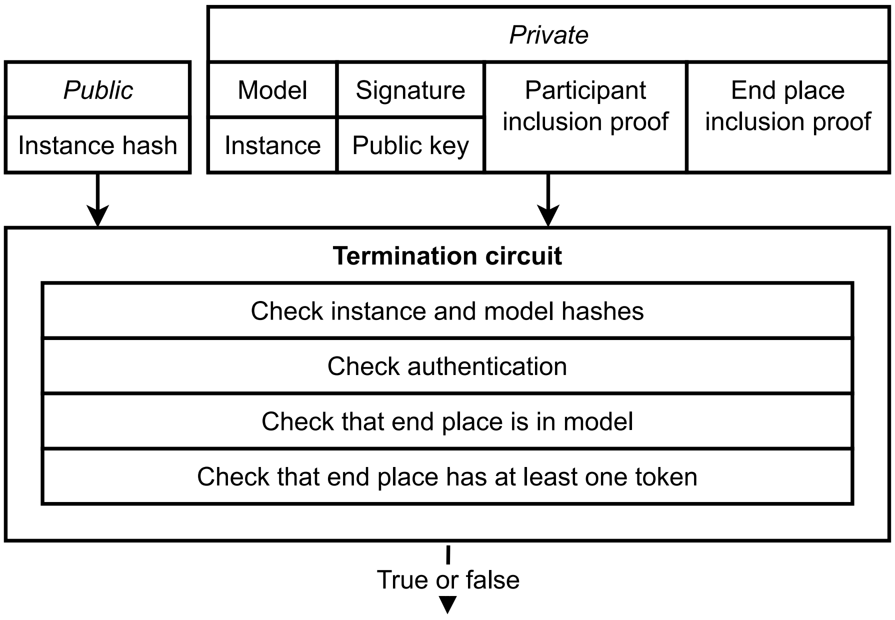
Private
Public
Instance hash
Model
Instance
Signature
Public key
Participant inclusion proof
End place inclusion proof
Termination circuit
Check instance and model hashes
Check authentication
Check that end place is in model
Check that end place has at least one token
True or false

Figure 5.6: Inputs and validation steps of the termination circuit

5.3.6 Proof Creation and Verification

We use the Gnark library [Bot+23], because of its superior performance when compared to other ZKP development frameworks[2]. It supports the Groth16 [Gro16] and the PLONK [GWC19] proving schemes, which can be instantiated with various elliptic curves. We opt for the Groth16 proving scheme, because it produces smaller proofs and has a faster verifier time. Furthermore, we select the BN254 curve, because there exist precompiled contracts on Ethereum for curve addition, scalar multiplication, and pairings. These precompiled contracts help with reducing the gas cost of on-chain zk-SNARK verification of Groth16 proofs. Choosing the BN254 curve implies that all circuit values belong to its scalar field, while public keys and signatures are points on the twisted Edwards curve over the BN254 scalar field [WBB20].

In an initial step, Gnark compiles each of our three circuits into an intermediate representation and performs a trusted setup to generate a set of proving and verifying keys for each circuit. Additionally, Gnark allows to export the verifying key directly as a Solidity smart contract. This smart contract includes a function for proof verification that accepts a proof parameter and the required public inputs. During the verification process, this function utilizes Ethereum precompiles for the point addition, scalar multiplication, and pairing operations, resulting in reduced gas costs for proof verification.

[2]https://blog.celer.network/2023/08/04/the-pantheon-of-zero-knowledge-proof-development-frameworks/

5.4 Smart Contracts

In this section, we delve into the smart contracts necessary for the management of instance data on the Ethereum blockchain. The main point of entry for interacting with the blockchain is the InstanceManager smart contract in Listing 5.4. Within this smart contract, there is a mapping from unsigned integers to boolean values, which represents a set of instance hashes. Whenever updates are made to the instances stored within the InstanceManager, a corresponding event is emitted. The verification of proofs is delegated to smart contracts that were generated by the Gnark library.

```
contract InstanceManager {
    mapping(uint => bool) public instances;

    InstantiationVerifier instantiationVerifier;
    TransitionVerifier transitionVerifier;
    TerminationVerifier terminationVerifier;

    event Instantiation(uint instance);
    event Transition(uint currentInstance, uint nextInstance);
    event Termination(uint instance);
    ...
}
```

Listing 5.4: Smart contract for managing instances

Listing 5.5 shows a method of the InstanceManager that is used after instantiating models. This method requires that the provided instance does not already exist within the contract and that the provided instantiation proof successfully verifies when the instance is used as public input. Subsequently, the instance is stored, and an instantiation event is emitted.

```
function instantiate(uint[8] memory proof, uint next) public {
    require(instances[next] == false);
    instantiationVerifier.verifyProof(proof, [next]);
    instances[next] = true;
    emit Instantiation(next);
}
```

Listing 5.5: InstanceManager method for creating instances

Listing 5.6 depicts an InstanceManager method that is used after a model transition has been executed on an instance. The method validates the existence of the current instance and verifies the transition proof using both instances as public input. If both checks are successful, the current instance is removed, the next instance is stored, and a Transition event is emitted.

```
function transition(uint[8] memory proof, uint current, uint next) public {
    require(instances[current] == true);
    transitionVerifier.verifyProof(proof, [current, next]);
    delete instances[current];
    instances[next] = true;
    emit Transition(current, next);
}
```

Listing 5.6: InstanceManager method for replacing instances with other instances

Lastly, the InstanceManager features a method for deleting instances that have reached a terminating state, as shown in Listing 5.7. This method verifies the existence of the current instance and checks that the termination proof is valid when the instance is used as public input. If both checks are successful, the instance is deleted, and a Termination event is emitted.

```
function terminate(uint[8] memory proof, uint current) public {
    require(instances[current] == true);
    terminationVerifier.verifyProof(proof, [current]);
    delete instances[current];
    emit Termination(current);
}
```

Listing 5.7: InstanceManager method for deleting instances

After a participant instantiated a model or executed a transition, they submit the hash of the new instance to the InstanceManager. Participants exclusively consider instances stored within the InstanceManager as valid, and valid models are those referenced by a valid instance.

5.5 Data Exchange

This section provides a high-level overview of how the proposed system exchanges models, instances, and messages among participants. The process data a participant has to possess before being able to execute a transition on an instance depends on whether the transition has a constraint and involves a sender and recipient. At the very minimum, the participant must have the model and the current instance. For transitions with a constraint, the participant has to provide all referenced messages and for those with a sender and recipient, both participant's signatures are required.

Figure 5.7 shows the data exchange process between participants and the InstanceManager smart contract during the operations: model instantiation, execution of transitions without sender or recipient, and instance termination. For each operation, the initiating participant submits instance hashes and a proof to the InstanceManager smart contract. Upon successful verification, the InstanceManager emits an event specific to the operation, which incorporates the submitted instance hashes. These hashes enable participants to link instances to a choreography execution and to authenticate received instances and models against these hashes. Notably, the distribution of instances and models serves as a notification to participants about their involvement in the choreography.

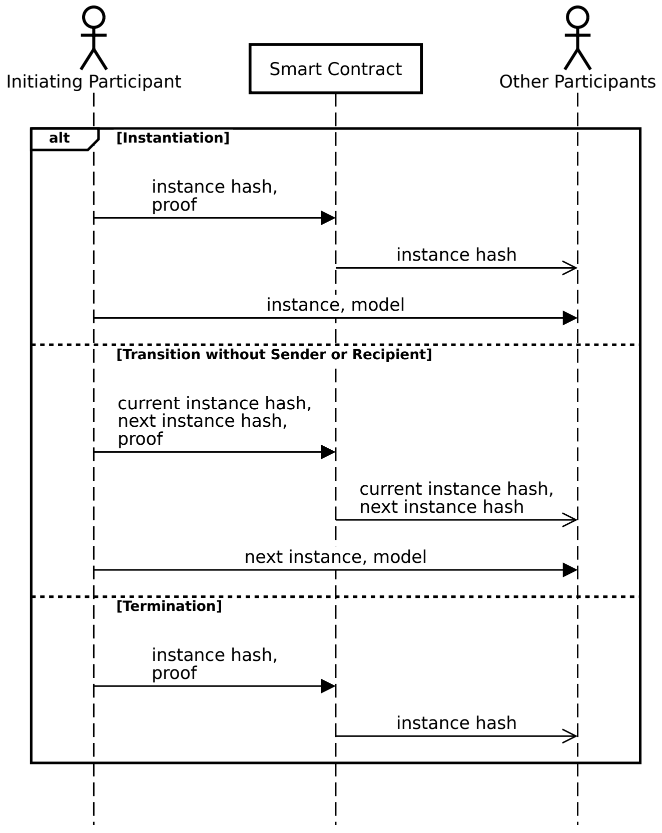

Figure 5.7: Data exchange during model instantiation, execution of transititons without sender or recipient, and instance termination

Figure 5.8 illustrates the data exchange process during a transition that specifies a sender and recipient. This involves the sender, recipient, other participants, and the InstanceManager smart contract. When the sender executes a transition on the current instance to progress to the next instance, they send the data required for generating the transition proof to the recipient. The recipient then verifies the received data using instance hashes stored in the InstanceManager. Upon successful verification, the recipient also signs the hash of the next instance and generates the transition proof. To streamline the data exchange process, the recipient directly submits this proof to the InstanceManager, eliminating the need to return the proof to the sender. While Figure 5.8 shows the recipient distributing the next instance and model to other participants, this step can alternatively be performed by the sender.

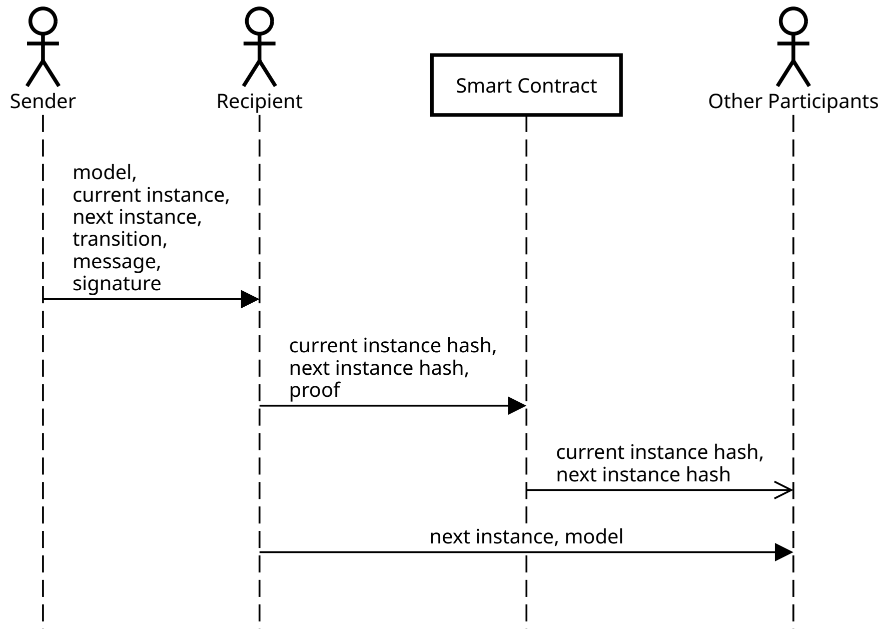

Figure 5.8: Data exchange during a transition that specifies a sender and recipient

The group of other participants in Figures 5.7 and 5.8 is identified by the initiating participant. This group must at least include all senders of transitions that are executable on the latest instance, though it may also encompass additional participants. If there are no executable transitions with a specified sender, the initiating participant can continue with these transitions without having to share the instance and model with others.

The exchange of data between participants is facilitated through the Messaging module, which was not implemented in our prototype. It is required to provide individual messaging channels for each pair of participants, with the option of group channels, and must use encryption to protect process data from unauthorized access. A crucial point is that the EdDSA public keys of participants cannot be utilized for elliptic curve Diffie-Hellman, as the pairing operation of the BN254 curve can be used to derive the shared secret. Therefore, each participant must be assigned a separate key pair dedicated to encryption, and these keys are incorporated into the instance This setup enables the creation of private channels between pairs of participants using a combination of public key and symmetric encryption.

CHAPTER 6

Evaluation & Discussion

This chapter provides an evaluation of the system's performance, conducts a qualitative comparison with a similar proposal, delineates its limitations, and assesses how well it meets the predefined requirements. Section 6.1 evaluates the prover's performance within the Execution module, detailing both the time required for proof generation and providing an asymptotic analysis of the circuit size. As it is one of the key aspects of practical feasibility, Section 6.2 provides an evaluation of execution cost. This is done by measuring the gas usage on the Ethereum blockchain and converting it to an approximate USD value. Due to the similarity to a similar proposal by Toldi and Kocsis [TK23], Section 6.3 conducts a qualitative comparison between our approach and theirs. Section 6.4 explores limitations of our system, specifically addressing the types of processes suitable as input and the associated execution costs. Following this, Section 6.5 provides a thorough evaluation of each requirement previously established.

6.1 Prover Performance

In this section, we provide the number of constraints and the prover time in seconds for the instantiation, transition, and termination circuits. These metrics are primarily determined by eight parameters that define maximum quantities for various model elements, and are listed in Table 6.1.

	×1	×2	×4	×8	×16	×32
MaxPlaceCount	16	32	64	128	256	512
MaxParticipantCount	16	32	64	128	256	512
MaxMessageCount	16	32	64	128	256	512
MaxStartPlaceCount	1	2	4	8	16	32
MaxEndPlaceCount	2	4	8	16	32	64
MaxTransitionCount	16	32	64	128	256	512
MaxBranchingFactor	4	4	4	4	4	4
MaxConstraintMessageCount	4	4	4	4	4	4

Table 6.1: Maximum quantities for various model elements

We calculated the ×1 values for these parameters as the nearest power of two that is greater than or equal to the average occurrence of corresponding BPMN elements within choreography models. These averages were derived from a recent study by Compagnucci et al. [Com+23] and involved dividing the occurrences of a given BPMN element within a representative set of choreography models by the number of these choreography models containing the element.

However, given that the base values of these parameters are derived from the average occurrences, there exist choreographies that require more elements than the parameters permit. To address this, we have defined parameter sets that are multiples of two larger than the base values. Table 6.2 presents the number of constraints within the instantiation, transition, and termination circuits for each of these parameter sets.

Circuit	×1	×2	×4	×8	×16	×32
Instantiation	24,834	36,835	60,558	108,988	211,331	439,820
Transition	63,836	89,819	139,796	237,987	432,384	819,007
Termination	25,529	37,763	60,902	105,964	194,763	370,922

Table 6.2: Number of constraints for different parameter sizes

The prover time for these circuits is shown in Figure 6.1. The computations were performed on a system equipped with a 12th Gen Intel® Core™ i7–12800H and 64GB RAM. Toots [Too20] report a proving time of 51 seconds for a BPMN process model consisting of 71 flow elements. On the other hand, Toldi and Kocsis [TK23] record a proving time of 122 seconds for what they describe as a representative process model, containing 68 flow elements. For the ×8 parameter set that supports models consisting of up to 128 transitions, or equivalently BPMN choreographies with up to 64 tasks, our system requires only 1 second.

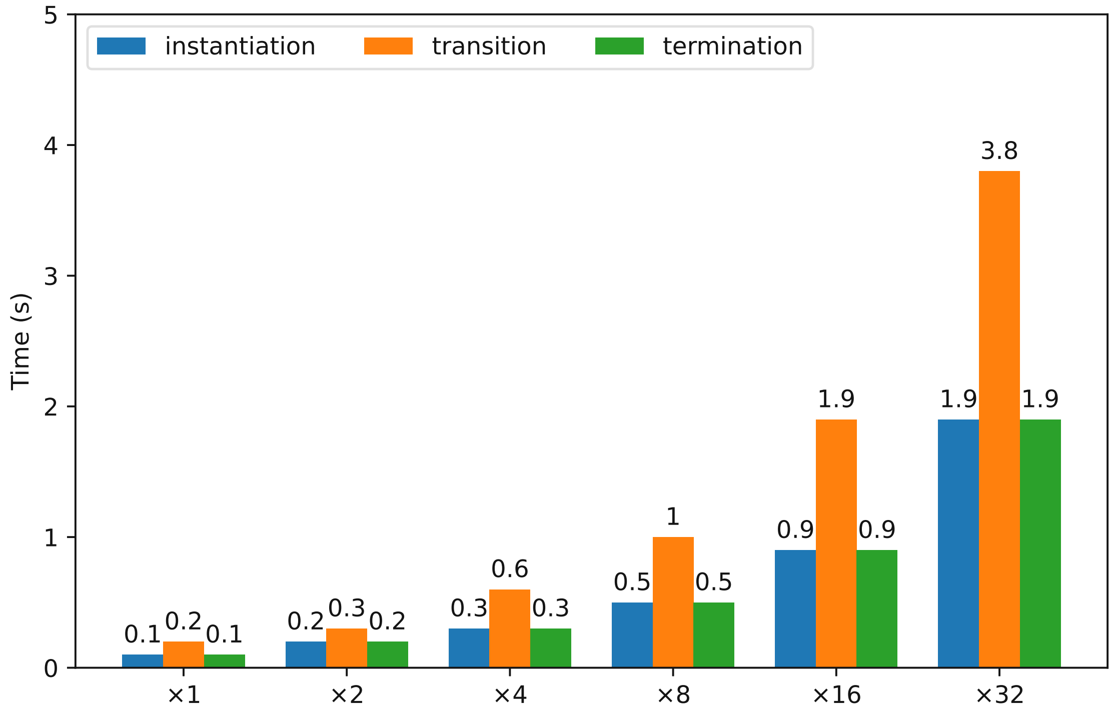

Figure 6.1: Prover time for different parameter sizes

Lastly, we examined asymptotic upper bounds for the number of constraints as a function of parameter size, which are depicted in Table 6.3. This table also lists the validation steps causing the components in each upper bound.

Circuit	$\mathcal{O}(\cdot)$	Validation step
Instantiation	$\mathsf{MaxPlaceCount} \times \mathsf{MaxStartPlaceCount}$	Token quantities
	$+\mathsf{MaxMessageCount}$	Message hashes
	$+\log \mathsf{MaxParticipantCount}$	Authentication
Transition	$\mathsf{MaxPlaceCount} \times \mathsf{MaxBranchingFactor}$	Token changes
	$+\mathsf{MaxMessageCount} \times \mathsf{MaxConstraintMessageCount}$	Message identification
	$+\mathsf{MaxStartPlaceCount}$	Model hash
	$+\log \mathsf{MaxParticipantCount}$	Authentication
	$+\log \mathsf{MaxTransitionCount}$	Transition inclusion
Termination	$\mathsf{MaxPlaceCount}$	Token quantities
	$+\mathsf{MaxMessageCount}$	Instance hash
	$+\mathsf{MaxStartPlaceCount}$	Model hash
	$+\log \mathsf{MaxParticipantCount}$	Authentication
	$+\log \mathsf{MaxEndPlaceCount}$	Token quantities

Table 6.3: Asymptotic upper bounds for the number of constraints

The number of constraints in the instantiation circuit is dominated by the multiplication of MaxPlaceCount and MaxStartPlaceCount, which arises from the validation step ensuring one token at all start places. Similarly, the transition circuit's constraint count is heavily influenced by the multiplication of MaxPlaceCount and MaxBranchingFactor, which results from token quantity comparisons between both instances. Additionally, the transition circuit validates whether the participant provided the correct messages, which introduces a number of constraints equivalent to the multiplication of MaxMessageCount and MaxConstraintMessageCount. Finally, the number of constraints within the termination circuit is primarily determined by MaxPlaceCount and MaxMessageCount. MaxPlaceCount is caused by verifying that provided end place has a token, while MaxMessageCount is introduced by computing the instance hash within the circuit.

6.2 Execution Cost

On Ethereum, execution cost is quantified by gas consumption. Table 6.4 displays the gas usage during the deployment of our smart contracts on Ethereum, while Table 6.5 shows the gas usage for each method of the InstanceManager smart contract. As of the current writing, considering a reasonable gas price of 20 gwei and an Ether price of $1790, the deployment of all our smart contracts amounts to $228.05 and each instantiation, transition, and termination costs $8.90, $9.19, and $8.12, respectively.

Contract	Gas usage
InstantiationVerifier	1,890,136
TransitionVerifier	1,922,797
TerminationVerifier	1,890,196
InstanceManager	666,961
Total	6,370,090

Table 6.4: Deployment gas usage on Ethereum

Method	Our gas usage	[TK23]
instantiate	248,651	2,408,635 (repr.)
transition	256,820	548,898 (repr.)
terminate	226,704	

Table 6.5: Transaction gas usage on Ethereum

Toldi and Kocsis [TK23] report their deployment gas usage and the average gas usage for updates. Since their approach requires a separate deployment for each new process, we list their representative deployment costs in Table 6.5 and compare it against our instantiation costs. Furthermore, each transaction updating the execution state consumes only about half the gas required for each of their updates.

6.3 Qualitative Comparison to Toldi and Kocsis [TK23]

While our system shares similarities with the approach presented by Toldi and Kocsis, it is essential to emphasize the key distinctions between the two approaches. Their approach is designed for BPMN process models with restrictions such as the requirement for gateways to be binary and the exclusion of loop structures. In constast, our approach supports BPMN choreographies containing loops and gateways with up to four incoming and outgoing sequence flows.

While Toldi and Kocsis compile each process model into an individual arithmetic circuit with hardcoded model information, we employ three arithmetic circuits to accommodate all choreography models. In their approach, participants encrypt the process execution state under a shared key and share it through a smart contract, whereas we rely on private messaging channels to distribute the model and instances among participants. Both approaches advocate pairwise messaging channels for message exchange between pairs of participants and for recipients to compare received messages against hashes in the execution state.

In their work, Toldi and Kocsis implement resource constraints by allowing users to attach public keys to various process elements, whereas we support assigning public keys to roles at the time of model instantiation. In contrast to our approach, their approach supports global variables and allows the attachment of variable names to activities to indicate their authorization to modify these variables. Additionally, their approach allows for the placement of boolean expressions on outgoing sequence flows of exclusive split gateways, which are directly compiled into the arithmetic circuit. While the constraints on previously sent integer messages introduced by our approach may be placed on a broader set of sequence flows, they are much less expressive than the general boolean expressions supported by Toldi and Kocsis.

6.4 Limitations

Our use of zk-SNARKs inherently imposes limitations on the supported choreographies, primarily due to the fixed size of our arithmetic circuits, which, in turn, affect the size of choreography models. Nevertheless, our system supports choreography models that are up to $\times 32$ larger than the average size found in literature [Com+23].

Apart from these obvious limitations, the removal of silent transitions introduces further constraints on the supported choreographies. Following the elimination of a silent transition, algorithm 5.1 ensures that choreographies remain executable by connecting transitions occurring before and after the silent transition. However, situations may arise where exclusive split and join gateways are connected through a single sequence flow in one path while another path contains at least one choreography task, as illustrated in Figure 6.6. In such scenarios, the resulting model contains transitions that can be executed multiple times, rather than just once.

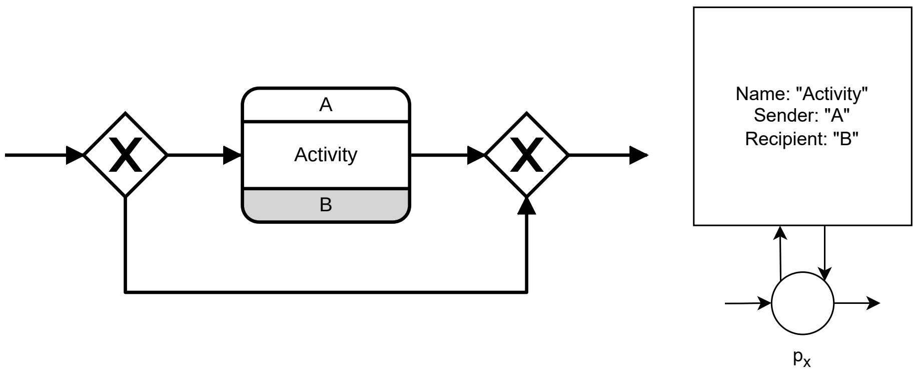

Table 6.6: Forbidden arrangement of BPMN elements and the resulting transition

There are also restrictions on where constraints can be applied. This limitation is due to the selected constraint format, which doesn't support the aggregation of multiple constraints into a single one. Consequently, constraints cannot be placed on multiple sequence flows leading to a parallel join gateway, as the parallel join gateway is converted into a single transition. Additionally, it's not possible to put constraints on both an incoming and outgoing sequence flow of any gateway since these constraints cannot be merged after removing the transition corresponding to the gateway. An exception is thrown at line 9 of algorithm 5.1 in such scenarios.

Given that the transition circuit accommodates multiple models, constraints cannot be compiled directly into the circuit. Consequently, the circuit has to interpret constraints, which imposes practical limitations on the types of constraints and data that can be effectively validated. We chose our constraint format, mainly because linear combinations of integer messages can be efficiently computed within arithmetic circuits. Attempting to evaluate arbitrary boolean expressions on byte data within an arithmetic circuit would result in enormous prover times.

A further limitation arises from the lack of global process data in choreographies, which restricts constraints to be defined over messages. The use of zk-SNARKs implies that participants intending to execute a transition must possess all messages referenced by its constraint. Due to the nature of choreographies, where no single participant has access to all messages, constraints can be formulated such that transitions cannot be executed.

We chose to implement separate circuits for instantiation, transition and termination in order to reduce circuit complexity. However, this approach resulted in increased deployment costs, requiring the deployment of three distinct verifier smart contracts instead of just one. Furthermore, the transaction costs associated with executing choreographies using our system are too high when it comes to frequently executed choreographies. For instance, the execution of a choreography with just 16 transitions costs $164.

6.5 Requirements

An assessment of whether our system meets the requirements outlined in Chapter 4 is presented in Table 6.7. A requirement category is considered fulfilled only if all its listed criteria are met.

Category	Fulfilled
Diagram support	Yes
Control-flow perspective	Yes
Resource perspective	Yes
Data perspective	Yes
Blockchain	Yes
Interpreted Execution	Yes
Privacy	Yes
Prover Time	Yes
Execution Cost	No
Extensibility	Yes

Table 6.7: Fulfillment of requirements

With the limitations discussed in Section 6.4, the system supports the control-flow semantics of start and end events, exclusive and parallel gateways and choreography tasks in BPMN choreography diagrams. We support role-based authentication and enable participants to verify hashes of received messages against those stored within instances. Additionally, constraints placed on sequence flows enable the formulation of data-based gateways that consider the contents of previously sent integer messages.

The InstanceManager smart contract effectively fulfills the blockchain requirements by synchronizing updates to the execution state, validating state changes according to the choreography, and emitting corresponding events. Our interpreted execution criteria are satisfied as the transformed model is used as a parameter in our system. This enables the Execution module to generate proofs, and allows the Blockchain module to verify them for arbitrary models, as long as they meet the predefined size constraints.

The security of the MiMC hash function [Alb+16] ensures that adversaries cannot discover the instance and salt linked to a blockchain hash, and participants cannot extract message contents from the hashes stored within an instance. The combination of the ZK property, and secure channels in the Messaging module guarantees that choreography data remains confidential to anyone not participating in the choreography. In cases where choreography compliance with regulations must be demonstrated to an auditor, any participant can simply supply the model and all instances.

While Section 6.1 shows that the prover time of our approach easily satisfies our requirements, Section 6.4 delves into the reasons behind not meeting the execution cost requirement. The extensibility requirement is fulfilled because integrating new diagram types into the system is relatively simple. This process merely involves translating these new types into our intermediate model representation. This translation can easily be implemented within the Bpmn module, where the existing logic for the removal of silent transitions can be reused for these newly incorporated diagram types.

CHAPTER 7

Conclusion and Future Work

In this thesis, we introduce an approach to executing process choreographies on the Ethereum blockchain. Our approach ensures the confidentiality of the execution state, while alignment with BPMN choreography diagrams is enforced using zk-SNARKs. Initially, we devise a representation of choreographies based on Petri nets and a mapping of the common elements of BPMN choreography diagrams to these choreography models. Based on these models, we define the concepts of choreography instantiation, the transition between choreography instances, and the termination of an instance. Utilizing arithmetic circuits, we then create algorithms to verify these operations, enabling their validation through zk-SNARKs. Subsequently, we present a smart contract designed to store and update hashes of process data for various choreographies, where the correctness of the submitted hashes is established using zk-SNARK proofs. Our architecture distributes these functions among distinct modules and includes a dedicated Messaging module that facilitates private message exchanges between participants.

While the approach by Toldi and Kocsis [TK23] is designed for acyclic BPMN process diagrams, we chose to support BPMN choreography diagrams, including those with loops. Furthermore, by optimizing the validation of choreography instances within arithmetic circuits, our method reduces proof generation times by 99%. In contrast to their approach, which requires a separate smart contract deployment for each new process, our system requires only a single deployment of three smart contracts for all choreographies up to a predefined maximum size. This strategy leads to a significant 90% reduction in gas consumption for instantiating new processes. In addition, compared to the approach of Toldi and Kocsis, our system achieves a 53% reduction in gas usage for each state update.

While our approach presents notable benefits, it has limited support for BPMN choreography diagrams. This limitation arises because the mapping to our internal representation is implemented only for the most commonly used BPMN elements, and certain arrangement of elements are not mapped to an equivalent arrangement in our representation. Furthermore, the expressiveness of our chosen constraint format might be insufficient for some applications. Another impactful limitation of our approach is the high execution costs, which can amount to hundreds of dollars for a single choreography execution. However, as highlighted by Stiehle and Weber [SW22], such high costs are a common issue in blockchain-based process enactment.

Future research directions could focus on reducing deployment costs by integrating our three arithmetic circuits into a single circuit with a dynamic parameter for switching between instantiation, transition, or termination modes. Such a consolidation could potentially decrease deployment gas usage by about 60%, requiring only one verifier smart contract instead of three. Additionally, to mitigate high transaction costs, forthcoming studies could investigate strategies like adopting rollups, which have been shown to substantially lower transaction fees [TSH22].

Future research could focus on developing constraint formats that offer greater expressiveness without compromising on prover times. Following the approach of Ladleif, Weske, and Weber [LWW19], another area for exploration is the integration of shared data into confidential choreography execution. Additionally, two key areas for development are building a functional prototype for the Messaging module and adding support for BPMN process diagrams.

List of Figures

List of Tables

List of Algorithms

Acronyms

Bibliography

[Alb+16] Martin Albrecht et al. „MiMC: Efficient Encryption and Cryptographic Hashing with Minimal Multiplicative Complexity“. In: *Advances in Cryptology – ASIACRYPT 2016*. Ed. by Jung Hee Cheon and Tsuyoshi Takagi. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2016, pp. 191–219. DOI: 10.1007/978-3-662-53887-6_7.

[AN20] Ahmad Alsharif and Mahmoud Nabil. „A Blockchain-based Medical Data Marketplace with Trustless Fair Exchange and Access Control“. In: *GLOBECOM 2020 - 2020 IEEE Global Communications Conference*. GLOBECOM 2020 - 2020 IEEE Global Communications Conference. Dec. 2020, pp. 1–6. DOI: 10.1109/GLOBECOM42002.2020.9348192.

[BA19] A. Berti and Wil M. P. van der Aalst. „Reviving Token-based Replay: Increasing Speed While Improving Diagnostics“. In: ATAED@Petri Nets/ACSD. 2019. URL: https://www.semanticscholar.org/paper/Reviving-Token-based-Replay%3A-Increasing-Speed-While-Berti-Aalst/c48d4680d5e6019460eb8beb98624dfe32f05a23 (visited on 02/26/2023).

[Baz+21] Mohamed Baza et al. „B-Ride: Ride Sharing With Privacy-Preservation, Trust and Fair Payment Atop Public Blockchain“. In: *IEEE Transactions on Network Science and Engineering* 8.2 (Apr. 2021), pp. 1214–1229. DOI: 10.1109/TNSE.2019.2959230.

[Ben+14] Eli Ben Sasson et al. „Zerocash: Decentralized Anonymous Payments from Bitcoin“. In: *2014 IEEE Symposium on Security and Privacy*. 2014 IEEE Symposium on Security and Privacy. May 2014, pp. 459–474. DOI: 10.1109/SP.2014.36.

[Ben+15] Eli Ben-Sasson et al. „Secure Sampling of Public Parameters for Succinct Zero Knowledge Proofs“. In: *2015 IEEE Symposium on Security and Privacy*. 2015 IEEE Symposium on Security and Privacy. May 2015, pp. 287–304. DOI: 10.1109/SP.2015.25.

[Ber+12] Daniel J. Bernstein et al. „High-Speed High-Security Signatures“. In: *Journal of Cryptographic Engineering* 2.2 (Sept. 1, 2012), pp. 77–89. DOI: 10.1007/s13389-012-0027-1.

[Bot+23] Gautam Botrel et al. *ConsenSys/Gnark: V0.9.1*. Version v0.9.1. Zenodo, Feb. 2023. DOI: 10.5281/zenodo.5819104.

[Bow+20] Sean Bowe et al. „ZEXE: Enabling Decentralized Private Computation“. In: *2020 IEEE Symposium on Security and Privacy (SP)*. 2020 IEEE Symposium on Security and Privacy (SP). May 2020, pp. 947–964. DOI: 10.1109/SP40000.2020.00050.

[Car+18] Josep Carmona et al. *Conformance Checking: Relating Processes and Models*. Cham: Springer International Publishing, 2018. DOI: 10.1007/978-3-319-99414-7.

[Com+23] Ivan Compagnucci et al. „A Study on the Usage of the BPMN Notation for Designing Process Collaboration, Choreography, and Conversation Models“. In: *Business & Information Systems Engineering* (June 20, 2023). DOI: 10.1007/s12599-023-00818-7.

[Cor+18] Flavio Corradini et al. „Collaboration vs. Choreography Conformance in BPMN 2.0: From Theory to Practice“. In: *2018 IEEE 22nd International Enterprise Distributed Object Computing Conference (EDOC)*. 2018 IEEE 22nd International Enterprise Distributed Object Computing Conference (EDOC). Stockholm: IEEE, Oct. 2018, pp. 95–104. DOI: 10.1109/EDOC.2018.00022.

[CRF18] Barbara Carminati, Christian Rondanini, and Elena Ferrari. „Confidential Business Process Execution on Blockchain“. In: *2018 IEEE International Conference on Web Services (ICWS)*. 2018 IEEE International Conference on Web Services (ICWS). July 2018, pp. 58–65. DOI: 10.1109/ICWS.2018.00015.

[DDO08] Remco M. Dijkman, Marlon Dumas, and Chun Ouyang. „Semantics and Analysis of Business Process Models in BPMN“. In: *Information and Software Technology* 50.12 (Nov. 2008), pp. 1281–1294. DOI: 10.1016/j.infsof.2008.02.006.

[Dim20] Tassos Dimitriou. „Efficient, Coercion-free and Universally Verifiable Blockchain-based Voting“. In: *Computer Networks* 174 (June 19, 2020), p. 107234. DOI: 10.1016/j.comnet.2020.107234.

[DMP22] Claudio Di Ciccio, Giovanni Meroni, and Pierluigi Plebani. „On the Adoption of Blockchain for Business Process Monitoring“. In: *Software and Systems Modeling* 21.3 (June 2022), pp. 915–937. DOI: 10.1007/s10270-021-00959-x.

[Dum+18] Marlon Dumas et al. *Fundamentals of Business Process Management*. Berlin, Heidelberg: Springer, 2018. DOI: 10.1007/978-3-662-56509-4.

[Dun+19] Sebastian Dunzer et al. „Conformance Checking: A State-of-the-Art Literature Review“. In: *Proceedings of the 11th International Conference on Subject-Oriented Business Process Management - S-BPM ONE ’19*. 2019, pp. 1–10. DOI: 10.1145/3329007.3329014. arXiv: 2007.10903 [cs].

[ET18] Jacob Eberhardt and Stefan Tai. „ZoKrates - Scalable Privacy-Preserving Off-Chain Computations“. 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)*. 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). July 2018, pp. 1084–1091. DOI: 10.1109/Cybermatics_2018.2018.00199.

[Faz+23] Armando Faz-Hernandez et al. *Hashing to Elliptic Curves*. RFC 9380. RFC Editor, Aug. 2023. DOI: 10.17487/RFC9380.

[Fri+18] Gilbert Fridgen et al. „Cross-Organizational Workflow Management Using Blockchain Technology - Towards Applicability, Auditability, and Automation“. In: Hawaii International Conference on System Sciences. 2018. DOI: 10.24251/HICSS.2018.444.

[GAC20] David Gabay, Kemal Akkaya, and Mumin Cebe. „Privacy-Preserving Authentication Scheme for Connected Electric Vehicles Using Blockchain and Zero Knowledge Proofs“. In: *IEEE Transactions on Vehicular Technology* 69.6 (June 2020), pp. 5760–5772. DOI: 10.1109/TVT.2020.2977361.

[Gar+17] Luciano García-Bañuelos et al. „Optimized Execution of Business Processes on Blockchain“. In: Sept. 10, 2017. DOI: 10.1007/978-3-319-65000-5_8.

[Gar+20] Julian Alberto Garcia-Garcia et al. „Using Blockchain to Improve Collaborative Business Process Management: Systematic Literature Review“. In: *IEEE Access* 8 (2020), pp. 142312–142336. DOI: 10.1109/ACCESS.2020.3013911.

[GJ20] Kobi Gurkan and Koh Wei Jie. „Community Proposal: Semaphore: Zero-Knowledge Signaling on Ethereum“. In: (Mar. 31, 2020). URL: https://docs.zkproof.org/pages/standards/accepted-workshop3/proposal-semaphore.pdf.

[Gro16] Jens Groth. „On the Size of Pairing-Based Non-interactive Arguments“. In: *Advances in Cryptology – EUROCRYPT 2016*. Ed. by Marc Fischlin and Jean-Sébastien Coron. Vol. 9666. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016, pp. 305–326. DOI: 10.1007/978-3-662-49896-5_11.

[Gud+20] Ivan Gudymenko et al. „Privacy-Preserving Blockchain-Based Systems for Car Sharing Leveraging Zero-Knowledge Protocols“. In: *2020 IEEE International Conference on Decentralized Applications and Infrastructures (DAPPS)*. 2020 IEEE International Conference on Decentralized Applications and Infrastructures (DAPPS). Aug. 2020, pp. 114–119. DOI: 10.1109/DAPPS49028.2020.00014.

[GWC19] Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. *PLONK: Permutations over Lagrange-bases for Oecumenical Noninteractive Arguments of Knowledge*. 2019. URL: https://eprint.iacr.org/2019/953 (visited on 10/07/2022). preprint.

[GY19] Hisham S. Galal and Amr M. Youssef. „Verifiable Sealed-Bid Auction on the Ethereum Blockchain". In: *Financial Cryptography and Data Security*. Ed. by Aviv Zohar et al. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2019, pp. 265–278. DOI: 10.1007/978-3-662-58820-8_18.

[Har87] David Harel. „Statecharts: A Visual Formalism for Complex Systems". In: *Science of Computer Programming* 8.3 (June 1, 1987), pp. 231–274. DOI: 10.1016/0167-6423(87)90035-9.

[Hev+04] Alan Hevner et al. „Design Science in Information Systems Research". In: *Management Information Systems Quarterly* 28 (Mar. 1, 2004), p. 75.

[HRZ10] Feng Hao, Peter Ryan, and Piotr Zielinski. „Anonymous Voting by Two-Round Public Discussion". In: *Information Security, IET* 4 (July 1, 2010), pp. 62–67. DOI: 10.1049/iet-ifs.2008.0127.

[Kha+21] Shafaq Naheed Khan et al. „Blockchain Smart Contracts: Applications, Challenges, and Future Trends". In: *Peer-to-Peer Networking and Applications* 14.5 (Sept. 1, 2021), pp. 2901–2925. DOI: 10.1007/s12083-021-01127-0.

[Kos+16] Ahmed Kosba et al. „Hawk: The Blockchain Model of Cryptography and Privacy-Preserving Smart Contracts". In: *2016 IEEE Symposium on Security and Privacy (SP)*. 2016 IEEE Symposium on Security and Privacy (SP). San Jose, CA: IEEE, May 2016, pp. 839–858. DOI: 10.1109/SP.2016.55.

[KY02] Aggelos Kiayias and Moti Yung. „Self-Tallying Elections and Perfect Ballot Secrecy". In: *Public Key Cryptography*. Ed. by David Naccache and Pascal Paillier. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2002, pp. 141–158. DOI: 10.1007/3-540-45664-3_10.

[Lei+18] Oded Leiba et al. „Incentivized Delivery Network of IoT Software Updates Based on Trustless Proof-of-Distribution". In: *2018 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*. 2018 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW). Apr. 2018, pp. 29–39. DOI: 10.1109/EuroSPW.2018.00011.

[Li+20] Wanxin Li et al. „Privacy-Preserving Traffic Management: A Blockchain and Zero-Knowledge Proof Inspired Approach". In: *IEEE Access* 8 (2020), pp. 181733–181743. DOI: 10.1109/ACCESS.2020.3028189.

[Lic+20] Tom Lichtenstein et al. „Data-Driven Process Choreography Execution on the Blockchain: A Focus on Blockchain Data Reusability". In: *Business Information Systems*. Ed. by Witold Abramowicz and Gary Klein. Lecture Notes in Business Information Processing. Cham: Springer International Publishing, 2020, pp. 224–235. DOI: 10.1007/978-3-030-53337-3_17.

[Lóp+19a] Orlenys López-Pintado et al. „Caterpillar: A Business Process Execution Engine on the Ethereum Blockchain“. In: *Software: Practice and Experience* 49.7 (2019), pp. 1162–1193. DOI: 10.1002/spe.2702.

[Lóp+19b] Orlenys López-Pintado et al. „Interpreted Execution of Business Process Models on Blockchain“. In: *2019 IEEE 23rd International Enterprise Distributed Object Computing Conference (EDOC)*. 2019 IEEE 23rd International Enterprise Distributed Object Computing Conference (EDOC). Oct. 2019, pp. 206–215. DOI: 10.1109/EDOC.2019.00033.

[LWW19] Jan Ladleif, Mathias Weske, and Ingo Weber. „Modeling and Enforcing Blockchain-Based Choreographies“. In: *Business Process Management*. Ed. by Thomas Hildebrandt et al. Vol. 11675. Cham: Springer International Publishing, 2019, pp. 69–85. DOI: 10.1007/978-3-030-26619-6_7.

[MDW22] Edoardo Marangone, Claudio Di Ciccio, and Ingo Weber. „Fine-Grained Data Access Control for Collaborative Process Execution on Blockchain“. In: *Business Process Management: Blockchain, Robotic Process Automation, and Central and Eastern Europe Forum*. Ed. by Andrea Marrella et al. Lecture Notes in Business Information Processing. Cham: Springer International Publishing, 2022, pp. 51–67. DOI: 10.1007/978-3-031-16168-1_4.

[Mie+13] Ian Miers et al. „Zerocoin: Anonymous Distributed E-Cash from Bitcoin“. In: *2013 IEEE Symposium on Security and Privacy*. 2013 IEEE Symposium on Security and Privacy. May 2013, pp. 397–411. DOI: 10.1109/SP.2013.34.

[MSH17] Patrick McCorry, Siamak F. Shahandashti, and Feng Hao. „A Smart Contract for Boardroom Voting with Maximum Voter Privacy“. In: *Financial Cryptography and Data Security*. Ed. by Aggelos Kiayias. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2017, pp. 357–375. DOI: 10.1007/978-3-319-70972-7_20.

[Mur89] T. Murata. „Petri Nets: Properties, Analysis and Applications“. In: *Proceedings of the IEEE* 77.4 (Apr. 1989), pp. 541–580. DOI: 10.1109/5.24143.

[NMK18] Hiroaki Nakamura, Kohtaroh Miyamoto, and Michiharu Kudo. „Inter-Organizational Business Processes Managed by Blockchain“. In: *Web Information Systems Engineering – WISE 2018*. Ed. by Hakim Hacid et al. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2018, pp. 3–17. DOI: 10.1007/978-3-030-02922-7_1.

[OMG11] OMG. *Business Process Model and Notation (BPMN), Version 2.0*. Object Management Group, Jan. 2011. URL: http://www.omg.org/spec/BPMN/2.0 (visited on 08/05/2022).

[Pan+20] Somnath Panja et al. „A Smart Contract System for Decentralized Borda Count Voting“. In: *IEEE Transactions on Engineering Management* 67.4 (Nov. 2020), pp. 1323–1339. DOI: 10.1109/TEM.2020.2986371.

[Ped92] Torben Pryds Pedersen. „Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing“. In: *Advances in Cryptology — CRYPTO ’91*. Ed. by Joan Feigenbaum. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 1992, pp. 129–140. DOI: 10.1007/3-540-46766-1_9.

[PNP20] Juha Partala, Tri Hong Nguyen, and Susanna Pirttikangas. „Non-Interactive Zero-Knowledge for Blockchain: A Survey“. In: *IEEE Access* 8 (2020), pp. 227945–227961. DOI: 10.1109/ACCESS.2020.3046025.

[Pop+20] Claudia Daniela Pop et al. „Blockchain and Demand Response: Zero-Knowledge Proofs for Energy Transactions Privacy“. In: *Sensors* 20.19 (19 Jan. 2020), p. 5678. DOI: 10.3390/s20195678.

[Pra+20] Miguel Ángel Prada-Delgado et al. „PUF-derived IoT Identities in a Zero-Knowledge Protocol for Blockchain“. In: *Internet of Things* 9 (Mar. 1, 2020), p. 100057. DOI: 10.1016/j.iot.2019.100057.

[Pry+20] Christoph Prybila et al. „Runtime Verification for Business Processes Utilizing the Bitcoin Blockchain“. In: *Future Generation Computer Systems* 107 (June 1, 2020), pp. 816–831. DOI: 10.1016/j.future.2017.08.024.

[SSD20] Shubham Sahai, Nitin Singh, and Pankaj Dayama. „Enabling Privacy and Traceability in Supply Chains Using Blockchain and Zero Knowledge Proofs“. In: *2020 IEEE International Conference on Blockchain (Blockchain)*. 2020 IEEE International Conference on Blockchain (Blockchain). Nov. 2020, pp. 134–143. DOI: 10.1109/Blockchain50366.2020.00024.

[Ste+19] Samuel Steffen et al. „Zkay: Specifying and Enforcing Data Privacy in Smart Contracts“. In: *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. CCS ’19. New York, NY, USA: Association for Computing Machinery, Nov. 6, 2019, pp. 1759–1776. DOI: 10.1145/3319535.3363222.

[Ste+22] Samuel Steffen et al. „ZeeStar: Private Smart Contracts by Homomorphic Encryption and Zero-knowledge Proofs“. In: *2022 IEEE Symposium on Security and Privacy (SP)*. 2022 IEEE Symposium on Security and Privacy (SP). May 2022, pp. 179–197. DOI: 10.1109/SP46214.2022.9833732.

[SVS22] Johannes Sedlmeir, Fabiane Völter, and Jens Strüker. „The next Stage of Green Electricity Labeling: Using Zero-Knowledge Proofs for Blockchain-Based Certificates of Origin and Use“. In: *ACM SIGEnergy Energy Informatics Review* 1.1 (Jan. 3, 2022), pp. 20–31. DOI: 10.1145/3508467.3508470.

[SW22] Fabian Stiehle and Ingo Weber. „Blockchain for Business Process Enactment: A Taxonomy and Systematic Literature Review“. In: *Business Process Management: Blockchain, Robotic Process Automation, and Central and Eastern Europe Forum*. Ed. by Andrea Marrella et al. Lecture Notes in Business Information Processing. Cham: Springer International Publishing, 2022, pp. 5–20. DOI: 10.1007/978-3-031-16168-1_1.

[Tha22] Justin Thaler. „Proofs, Arguments, and Zero-Knowledge“. In: *Foundations and Trends® in Privacy and Security* 4.2–4 (Dec. 6, 2022), pp. 117–660. DOI: 10.1561/3300000030.

[TK23] Balázs Ádám Toldi and Imre Kocsis. „Blockchain-Based, Confidentiality-Preserving Orchestration of Collaborative Workflows“. In: *Infocommunications journal* 15.3 (2023), pp. 72–81. DOI: 10.36244/ICJ.2023.3.8. arXiv: 2303.10500 [cs].

[Too20] Aivo Toots. „Zero-Knowledge Proofs for Business Processes“. MA thesis. University of Tartu, 2020. URL: https://cyber.ee/uploads/aivo_toots_msc_438a500a90.pdf (visited on 09/15/2022).

[TSH22] Louis Tremblay Thibault, Tom Sarry, and Abdelhakim Senhaji Hafid. „Blockchain Scaling Using Rollups: A Comprehensive Survey“. In: *IEEE Access* 10 (2022), pp. 93039–93054. DOI: 10.1109/ACCESS.2022.3200051.

[vdAal16] Wil van der Aalst. *Process Mining*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016. DOI: 10.1007/978-3-662-49851-4.

[Wan+19] Zhiguo Wan et al. „Zk-AuthFeed: How to Feed Authenticated Data into Smart Contract with Zero Knowledge“. In: *2019 IEEE International Conference on Blockchain (Blockchain)*. 2019 IEEE International Conference on Blockchain (Blockchain). July 2019, pp. 83–90. DOI: 10.1109/Blockchain.2019.00020.

[WBB20] Barry WhiteHat, Jordi Baylina, and Marta Bellés. „Baby Jubjub Elliptic Curve“. In: *Ethereum Improvement Proposal, EIP-2494* 29 (2020).

[Web+16a] Ingo Weber et al. „Untrusted Business Process Monitoring and Execution Using Blockchain“. In: *Business Process Management*. Ed. by Marcello La Rosa, Peter Loos, and Oscar Pastor. Vol. 9850. Cham: Springer International Publishing, 2016, pp. 329–347. DOI: 10.1007/978-3-319-45348-4_19.

[Web+16b] Ingo Weber et al. *Using Blockchain to Enable Untrusted Business Process Monitoring and Execution*. Technical Report UNSW-CSE-TR-201609. University of New South Wales, 2016.

[Wes07] Mathias Weske. *Business Process Management: Concepts, Languages, Architectures*. Berlin ; New York: Springer, 2007. 368 pp. ISBN: 978-3-540-73521-2.

[Wu+20] Wei Wu et al. „Blockchain Based Zero-Knowledge Proof of Location in IoT“. In: *ICC 2020 - 2020 IEEE International Conference on Communications (ICC)*. ICC 2020 - 2020 IEEE International Conference on Communications (ICC). June 2020, pp. 1–7. DOI: 10.1109/ICC40277.2020.9149366.

[XZS22] Tiancheng Xie, Yupeng Zhang, and Dawn Song. „Orion: Zero Knowledge Proof with Linear Prover Time“. In: *Advances in Cryptology – CRYPTO 2022*. Ed. by Yevgeniy Dodis and Thomas Shrimpton. Vol. 13510. Cham: Springer Nature Switzerland, 2022, pp. 299–328. DOI: 10.1007/978-3-031-15985-5_11.

[YL20] Xiaohui Yang and Wenjie Li. „A Zero-Knowledge-Proof-Based Digital Identity Management Scheme in Blockchain“. In: *Computers & Security* 99 (Dec. 1, 2020), p. 102050. DOI: `10.1016/j.cose.2020.102050`.

[ZC16] Zhichao Zhao and T.-H. Hubert Chan. „How to Vote Privately Using Bitcoin“. In: *Information and Communications Security*. Ed. by Sihan Qing et al. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2016, pp. 82–96. DOI: `10.1007/978-3-319-29814-6_8`.

[ZWX20] Shufan Zhang, Lili Wang, and Hu Xiong. „Chaintegrity: Blockchain-Enabled Large-Scale e-Voting System with Robustness and Universal Verifiability“. In: *International Journal of Information Security* 19.3 (June 1, 2020), pp. 323–341. DOI: `10.1007/s10207-019-00465-8`.