SEPAR: A Privacy-Preserving Blockchain-based System for Regulating Multi-Platform Crowdworking Environments

Mohammad Javad Amiri\textsuperscript{1} Joris Duguépéroux\textsuperscript{2} Tristan Allard\textsuperscript{2} Divyakant Agrawal\textsuperscript{1} Amr El Abbadi\textsuperscript{1} \\
\textsuperscript{1}Department of Computer Science, University of California Santa Barbara \hspace{1cm} \textsuperscript{2}Univ Rennes, CNRS, IRISA \\
\{amiri, agrawal, amr\}@cs.ucsb.edu, \{joris.dugueperoux, tristan.allard\}@irisa.fr

Abstract

Despite recent intensive research, existing crowdworking systems do not adequately address all the requirements of a real-world crowdworking environment. First, crowdworking platforms need to integrate within society and in particular to interface with legal and social institutions. Global regulations must be enforced, such as minimal and maximal work hours that participants can spend on crowdworking platforms. Second, crowdworking platforms are naturally distributed and need to collaborate with each other to process complex tasks, resulting in the rise of multi-platform crowdworking systems. Moreover, while collaborating to enforce global regulations or while processing complex tasks that require the transparent sharing of information about the tasks, the system needs to preserve the privacy of all participants. In this paper, we present SEPAR, a multi-platform crowdworking system that enforces global constraints on distributed independent entities. In SEPAR, Privacy is ensured using lightweight and anonymous tokens, while transparency is achieved using a permissioned blockchain shared across multiple platforms. To support fault tolerance and support collaboration among platforms, SEPAR provides a suite of distributed consensus protocols. The privacy guarantees of SEPAR against covert adversaries are formalized and thoroughly demonstrated, and the experiments reveal the efficiency of SEPAR in terms of performance and scalability.

1 Introduction

The rise of the platform economy\textsuperscript{1} \cite{17,20} is reshaping work all around the world. Crowdsourcing platforms dedicated to work (also called crowdworking platforms\textsuperscript{2} \cite{8}) are online intermediaries between requesters and workers, where requesters propose tasks while workers propose skills and time. By providing requesters (resp. workers) 24/7 access to a worldwide workforce (resp. worldwide task market), crowdworking platforms have grown in numbers, diversity, and adoption\textsuperscript{1}. Today, crowworkers come from countries spread all over the world, and work on several, possibly competing, platforms \cite{8}. The use of crowdworking platforms is expected to continue growing \cite{31}, and in fact they are envisioned as key technological components of the future of work \cite{32}.

Crowdworking platforms, however, challenge national boundaries, weaken the formal relationships between workers and requesters, and are often not considered legal as employers. Guaranteeing the compliance of crowdworking platforms with national or regional labour laws is hard\textsuperscript{2} \cite{31} despite the stringent need for

\textsuperscript{1}See for example: Amazon Mechanical Turk (https://www.mturk.com/), Wirk (https://www.wirk.io/), or Appen (https://appen.com/) for micro-tasks, Uber (https://www.uber.com/) or Lyft (https://www.lyft.com/) for rides, TaskRabbit (https://www.taskrabbit.com/) for home maintainance, Kicklox (https://www.kicklox.com/) for collaborative engineering.

\textsuperscript{2}See, e.g., the Otey V Crowdflower class action against a famous microtask platform for "substandard wages and oppressive working hours" (https://casetext.com/case/otey-v-crowdflower-1).
regulating work. For example, the preamble of the 1919 constitution of the International Labour Organization [30], written in the ruins of World War I, states that: “Whereas universal and lasting peace can be established only if it is based upon social justice; (...) an improvement of those conditions is urgently required; as, for example, by the regulation of the hours of work, including the establishment of a maximum working day and week, the regulation of the labour supply (...).” The global regulation of the work hours represents the minimal and maximal number of hours that participants, i.e., worker, requester, and platform, can spend on crowdsourcing platforms. While legal tools are currently being investigated, e.g., a Universal Labour Guarantee [31], there is a stringent need for technical tools allowing official institutions to enforce regulations.

Most current crowdsourcing platforms are independent of each other. However, the emergence of more complex tasks and novel requirements for both workers and requesters, on one hand, and the enforcement of legal regulations, on the other hand, highlights the need for collaboration between crowdsourcing platforms, thus resulting in multi-platform crowdsourcing systems. For example, many drivers work for both Uber and Lyft concurrently\(^3\), while requesters may also request multiple drivers from both Uber and Lyft concurrently. The observation holds also for microtask platforms [8], where a common combination among workers is Amazon Mechanical Turk and Prolific, or for on-demand services\(^4\). Participants in a crowdsourcing task may also behave maliciously or act as adversaries for their benefits, e.g., violate the privacy of participants or the regulations. Therefore, to check the enforcement of legal regulations in a multi-platform crowdsourcing environment, we need to reconcile transparency with privacy. Indeed, while enforcing limits on the hours of work over several crowdsourcing platforms requires the transparent sharing of information about the crowdsourcing tasks performed by each platform, without any privacy protection measures, this may lead to out-of-control disclosures about the participants. Transparent and privacy-preserving collaboration between multiple platforms might also be needed to address complex cross-platform tasks. If a requester submits a task with a specified number of requested solutions to multiple platforms, the involved platforms need to collaborate with each other in order to assign workers and provide the specified number of solutions. As a result, a multi-platform system needs to establish consensus between platforms to enable them either to enforce legal regulations or to process cross-platform tasks.

In this paper, we present SEPAR, a technical solution to the problem of imposing global constraints on distributed independent entities in the context of multi-platform crowdsourcing systems. The problem is non-trivial because of the complexity of the conjunction of the required properties:

1. **Expressibility**: The constraints need to be expressed in a simple and non-ambiguous manner.

2. **Transparent and Privacy-preserving Constraint Enforcement**: Crowdsourcing platforms need to share information about the tasks performed without jeopardizing the privacy of participants in order to allow both the enforcement of the global constraints and the collaborative processing of cross-platform tasks.

3. **Distributed Collaboration**: Crowdsourcing platforms are naturally distributed and need to collaborate through distributed consensus algorithms.

SEPAR proposes a privacy-preserving token-based system where global constraints are modeled using lightweight and anonymous tokens distributed to workers, platforms, and requesters. Our system formally guarantees that global constraints are satisfied by construction and limits the information shared among platforms and participants to the minimum necessary for performing the tasks against adversarial participants acting as covert adversaries. We extend our token-based system to allow participants to prove to external entities (e.g., social security agencies) their involvement in crowdsourcing tasks. The resulting proofs are called certificates. To provide transparency across multiple platforms, SEPAR proposes a blockchain-based distributed ledger shared across platforms. Nonetheless, for the sake of privacy and to improve performance, the blockchain ledger is not maintained by any single platform and each platform maintains only a view of the ledger. We then design a suite of distributed consensus algorithms across platforms for coping with the concurrency issues inherent to a multi-platform context and formally prove their correctness. Salient features of SEPAR include the simplicity of its building blocks (e.g., usual asymmetric encryption scheme) and its compatibility with today’s platforms (e.g., it does not jeopardize their privacy requirements about requesters and workers for enforcing the regulation).

In a nutshell, the contributions of this paper are as follows:

---
\(^3\)For example, ridesharapps.com provides tutorials to help drivers manage apps to optimize their earnings [https: //ridesharapps.com/drive-for-uber-and-lyft-at-the-same-time/].

\(^4\)See, e.g., [https://tinyurl.com/nytgigmult].
1. A privacy model stating formally the privacy requirements of a multi-platform regulated crowdworking system based on the well-known simulatability paradigm,
2. A simple language for expressing global constraints, e.g., limits on the number of work hours, and mapping them to SQL constraints to ensure semantic clarity,
3. SEPAR, a privacy-preserving transparent multi-platform crowdworking system that enforces a given set of constraints. (1) Privacy is ensured using lightweight and anonymous tokens, while (2) transparency is achieved using a blockchain shared across platforms. The token-based system is extended to allow participants to prove to external entities their involvement in crowdworking tasks.
4. A suite of distributed consensus protocols for coping with the concurrency issues inherent to a multi-platform context.
5. A formal security analysis and thorough experimental evaluation.

The paper is organized as follows. The technical background and related work are discussed in Section 2. Section 3 defines the problem that SEPAR addresses. The language for expressing constraints is expressed in Section 4. The token-based system for enforcing constraints and the extended system for certificates are designed in Section 5. The blockchain ledger and consensus protocols are presented in Section 6. Section 7 details our thorough experimental evaluation, and finally, Section 8 concludes the paper.

2 Relevant Background

In this section we provide the relevant background for SEPAR. This is divided into two parts. First, we provide some basic background material on group signatures which provide a basic cryptographic approach for signing collectively while ensuring verifiable anonymity of the individual signers, and blockchains, which will be used as a basic building block for SEPAR. Second, we present relevant related work from both the crowdworking and the blockchain literature.

2.1 Relevant Technical Material

**Group Signature.** A group signature scheme is a signature scheme for groups that respects three main properties, as defined first in [16]: (1) only members of the group can sign messages, (2) the receiver of the signature can verify that it is a valid signature of that group, but cannot discover which member of the group made it, and (3) in case of dispute later on, the signature can be "opened" (with or without the help of the group members) to reveal the identity of the signer. A common way to enforce the third property is to rely on a group manager, that can add new members to the group, or revoke the anonymity of a signature. Instances of such schemes are proposed in [16], but also in [5,11]. In this paper, we use the protocol proposed in [11], and denote \( \text{GroupSign}(\text{key}_{\text{priv}}, p, g, m) \) the group signature of participant \( p \) (with her private key) of group \( g \), for the message \( m \). The notation \( \text{Sign}(\text{key}_{\text{priv}}, p, m) \) may also be used to refer to a simple asymmetric signature of the message \( m \) by user \( p \) (e.g., RSA).

**Blockchain.** A blockchain is a distributed data structure for recording transactions maintained by several nodes without a central authority [10]. In a blockchain, transactions are recorded in an append-only data structure, called Blockchain ledger. Nodes of a blockchain system agree on their shared states across a large network of untrusted participants. While in a permissionless blockchain system, e.g., Bitcoin [29], the network is public, and anyone can participate without a specific identity, a permissioned blockchain, e.g., Hyperledger Fabric [4], consists of a set of known, identified nodes which might not fully trust each other. The unique features of blockchain such as transparency, provenance, fault tolerance, and authenticity are used by many systems to deploy a wide range of distributed applications such as supply chain management [21] and healthcare [7] in a permissioned settings. In particular and for a crowdworking system, the transparency of blockchains can be used to check integrity constraints, provenance enables the system to trace how data is transformed, fault tolerance helps to enhance reliability and availability, and finally, authenticity guarantees that signatures and transactions are valid.

2.2 Related Work

Enhancing privacy in the context of crowdworking has been addressed by several recent studies with various kinds of guarantees, from differential privacy [36,37] to cryptography [25–27], mostly focusing on spatial crowdsourcing and the use of geolocation to perform assignment. In ZebraLancer [28] and ZKCrowd [38], blockchains and consensus protocols are also used to add transparency guarantees on top of privacy. However, all these works consider a single-platform context, with no external constraints, preventing many real-
life legislation to apply. To the best of our knowledge, SEPAR is the first to support a multi-platform crowdworking context, with external constraints, transparency, and privacy expectations at the same time.

In the context of blockchains, Hyperledger Fabric [4] ensures the confidentiality of data using Private Data Collections [1]. Private Data Collections manage confidential data that two or more entities want to keep private from other entities. Quorum [15] supports public and private transactions and ensures the confidentiality of private transactions using the Zero-knowledge proof technique. In Quorum, however, all public as well as private transactions are ordered using a single consensus protocol resulting in low throughput.

Providing anonymity as well as untraceability has been addressed by ZCash [19] which is restricted to the management of crypto-currency issues. Hawk and Raziel [22,33] manage wider issues, and include general smart contracts. However, these solutions do not incorporate infrastructures with multiple platforms, nor implement constraints (let alone anonymized ones). Finally, Solidus [14] proposes to privately manage a multi-platform banking system, with individual banks managing their own clients, while allowing cross-platform transactions. While Solidus may be sufficient for banking systems, it does not consider users that subscribe to multiple platforms, nor envisions global profiles or constraints.

3 Problem Formulation

In this section we provide a motivating example to illustrates the challenges of crowdworking systems and then formulate the problem. We finally, explain the security model.

3.1 Motivating Example

Multi-platform crowdworking systems face two main privacy preserving challenges: enforcing multi-platform regulations and supporting cross-platform tasks. We consider constraining the number of work hours in a ridesharing use-case to illustrate the challenge of enforcing privacy preserving multi-platform regulations. In ridesharing scenarios, a set of workers (i.e., drivers) gives rides to a set of requesters (i.e., travelers) through a set of platforms, e.g., Uber, Lyft, Curb, and Juno, where each driver (resp. traveler) registers to one or more platforms. Regulations on the hours of work often specify minimal and maximal number of work hours that can be performed by the participants. For instance, (1) the total work hours of a driver per week may not exceed 40 hours to follow the Fair Labor Standards Act (FLSA), (2) a driver has to work at least 5 hours per week to be eligible for insurance coverage, and (3) the total work hours of all drivers on a platform should be at least 1000 hours per week to enable the platform to fill for a tax refund. A multi-platform crowdworking system needs to express and enforce such regulations while preserving the privacy of participants. Indeed, the system needs to (1) provide a technical tool to enable official institutions expressing the regulations, (2) support transparent sharing of information about the crowdworking tasks performed by each platform to enable them checking the enforcement of regulations, and (3) preserve the privacy of participants.

Supporting complex cross-platform tasks that may need multiple contributions from possibly different platforms raises the second set of challenges. For instance, a requester who has registered with Amazon Mechanical Turk, Appen and other microtask platforms might need hundreds or thousands of contributions at the same time. The requester would like to accept these contributions from workers regardless of the platforms the microtasks are performed on. Since workers from different platforms might want to perform these contributions, the system needs to establish consensus among the various microtask platforms to assign workers and provide the specified number of solutions without revealing any private information about the workers to competing platforms.

3.2 Crowdworking Environment

3.2.1 Participants

Today’s realistic crowdworking environments consist of a set of workers \( W \) interacting with a set of requesters \( R \) through a set of competing platforms \( P \). We call participants the workers, platforms, and requesters of a crowdworking environment. Each worker \( w \in W \) (1) registers to one or more platforms \( P_w \subset P \) according to her preferences and, through the latter, (2) accesses the set of tasks available on \( P_w \), (3) submits each contribution to the platform \( p \in P_w \) she elects, and (4) obtains a reward for her work. On the other side, each requester \( r \in R \) similarly (1) registers to one or more platforms \( P_r \subset P \), (2) issues a submission which contains her tasks \( T_r \) to one or more platforms \( p \in P_r \), (3) receives the contributions of each worker \( w \).
registered to $P_r \cap P_w$, having elected a task $t \in T_r$, and (4) launches the distribution of rewards. Platforms are thus in charge of facilitating the intermediation between workers and requesters. A crowdworking process $\pi$ connects three parties – a worker $w$, a platform $p$, and a requester $r$ – with each other and aims to solve a task $t \in T_r$ through $p$, and consists in the steps (2) to (4) above. Figure 1 shows a crowdworking infrastructure with four platforms, four workers and four requesters.

In this work, we do not focus on the description of tasks and contributions and consequently model both as arbitrary bitstrings $\{0, 1\}^*$ and make no assumption on the distribution of rewards to workers$^6$.

Finally, workers, requesters, and platforms are all equipped with the cryptographic material required by SEPAR: a pair of usual public/private asymmetric keys (e.g., RSA) and a pair of public/private asymmetric group keys (e.g., [11]) where the union of all workers forms a group (in the sense of group signatures), similar to the union of all requesters, and to the union of all platforms. Participants acquire them when joining SEPAR (see Section 5 for more information).

### 3.2.2 Interactions with Institutions

Crowdworking environments do not exist in a vacuum but rather are integrated within society as a whole. The participants need in particular to interact with legal institutions (in order to enforce the local labor laws) and with social institutions (in order to enable local social rights). We capture these interactions through less-than constraints and greater-than certificates.

**Constraints.** A set of constraints embodies the labor policy that applies to a given crowdworking environment. Essentially, a constraint expresses a limit on the actions that can be performed by the participants of the crowdworking environment, $e.g.$, the total working hours of a worker per week must not exceed 40 hours across all platforms. Constraints must be expressed in an intuitive language that is both expressive enough to adapt to a variety of real-world policies and at the same time restrictive enough to guarantee the efficiency of their enforcement.

**Certificates.** A certificate is a piece of information that participants can provide to third parties to prove that they took part in a given crowdworking process. Contrary to constraints, they are not a priori specification: they are made available during the process to the participants involved and can be provided by participants to other parties on demand after the process. Certificates are well suited to real-world situations such as conforming to legal obligations, suing other parties in court in case of abuse, or legitimizing applications to grants or tax refund depending on local legislation, $e.g.$, a driver has to work at least 5 hours

---

$^6$A task embeds all the information necessary to be performed by a worker ($e.g.$, the precise description of the work that must be performed, a reward policy for distributing the reward among contributors, the expected number of contributions).
per week to be eligible for insurance coverage or the total work hours of all drivers on a platform should be at least 1000 hours per week to enable the platform to fill for a tax refund.

3.2.3 Distribution of Platforms

We do not make any assumptions on the inner working of platforms, especially on their inner implementation of crowdworking processes (e.g., task assignment algorithm, workers contributions delivery). However, we stress that our approach is compatible with distributed infrastructures, supported by one or more data centers, following today’s fault tolerance and performance standards. In particular, we assume each platform consists of a set of nodes in an asynchronous distributed network. Nodes are connected by point-to-point bi-directional communication channels. To guarantee data consistency, a total order among the transactions of each platform is needed. To establish a total order, asynchronous consensus protocols can be used where nodes agree on an ordering of incoming requests using the state machine replication algorithm [23]. Each node has a distributed persistent transparent datastore where transactions are committed to the datastore. In this paper and due to the unique features of blockchains such as transparency, provenance, and fault tolerance, the datastore is implemented using a blockchain.

A crowdworking environment processes internal, i.e., submitted to a single platform, and cross-platform, i.e., submitted to more than one platform, tasks. Processing a task (either internal or cross-platform) requires agreement from the nodes of the involved platforms. To establish agreement among the nodes, we introduce local and cross-platform consensus protocols. In addition, we enable all platforms checking the satisfaction of constraints by establishing consensus among every node of all platforms. To do so, a global consensus protocol is introduced.

3.3 Security Model

We consider that any participant in a crowdworking environment may act as a covert adversary [6] that aims at inferring anything that can be inferred from the execution sequence and that is able to deviate from the protocol if no other participant detects it. Adversarial participants may additionally collude.

The privacy definition that we adopt requires that no participant obtains or infers any information about a crowdworking process beyond what is strictly needed for accomplishing its local crowdworking processes and for the distributed enforcement of constraints. We formalize below this requirement by defining the set of secrets and by using the well-known simulatability model often used by secure multi-party computation algorithms.

Consider a crowdworking process \( \pi \) between worker \( w \), platform \( p \), and requester \( r \) for solving task \( t \). The information generated by the execution of \( \pi \) consists at least of a starting event \( \text{BEGIN} \), an ending event \( \text{END} \), and the relationship between the three participants \((w, p, r)\) with the task \( t \). We denote it by the 6-tuple \((\text{BEGIN}, \text{END}, w, p, r, t)\). The \(\{\text{BEGIN}, \text{END}\}\) events are abstract representations of the information that \( \pi \) is starting or ending. They may be given, e.g., by the exchange of messages between the participants to \( \pi \), and may come with additional concrete information (e.g., timestamps, IP address). Our privacy definition focuses on workers, requesters, and tasks and requires the secrecy of the corresponding parts of this tuple from the participants that are not involved in \( \pi \). However, since a given task \( t \) may be submitted to several platforms and then be accessed by several workers, the platforms and workers not involved in \( \pi \) but that receive \( t \) still need to learn that \( t \) has been completed (e.g., to manage their local copy of the task). We capture this subtlety through a varying set of disclosures, denoted \( \delta^\pi \), plugged into a unified simulation-based privacy definition. Our definition does not leak any information about the worker and the requester involved in \( \pi \) when it is not needed by \( \pi \). It tolerates the disclosure of the \(\{\text{BEGIN}, \text{END}\}\) events and of the platform \( p \) to all participants, whatever their involvement in \( \pi \). This allows platforms to share information for enforcing global regulations (e.g., check that all participants satisfy the related constraints before executing \( \pi \)), and to collaborate for managing correctly the cross-platforms tasks. Note that for simplicity we will use the same notation \( \delta \) for disclosures concerning sets of crowdworking processes as well.

We specify the set of disclosures as follows:

- Secrecy against the participants that are not involved in \( \pi \) and that have not received task \( t \) from requester \( r \): they must not learn anything about the worker, the task, and the requester involved in \( \pi \):
  \[
  \delta^\pi_{R \rightarrow t} = (\text{BEGIN}, \text{END}, p)
  \]

- Secrecy against the platforms and workers that have received the task \( t \) from \( r \) but that are not involved in \( \pi \): they must be aware that \( t \) has been performed (e.g., for not contributing to \( t \)) but
must not know that it has been performed by worker \( w \):

\[
\delta_{R,I}^{\pi} = (\text{BEGIN}, \text{END}, p, r, t)
\]

- Secrecy against participants that are directly involved in \( \pi \) (and have thus received task \( t \)): the information about the execution of \( \pi \) disclosed to \( w, p, \) and \( r \) is complete simply because they run \( \pi \):

\[
\delta_{RI}^{\pi} = (\text{BEGIN}, \text{END}, w, p, r, t)
\]

**Definition 3.1** Let \( \Pi \) be a set of crowdworking processes executed by \( \sigma \) an instance of SEPAR over a set of participants. We say that \( \sigma \) is \( \delta^{\Pi} \)-private if, for all \( \pi \in \Pi \), for all computationally-bounded adversaries \( \chi \), the sets of disclosures \( \delta_{R,I}^{\pi}, \delta_{RI}^{\pi}, \delta_{RI}^{\pi} \), arbitrary background knowledge \( \chi \in \{0, 1\}^{\ast} \), the distribution representing the adversarial knowledge in the real setting is computationally indistinguishable from the distribution representing the adversarial knowledge in an ideal setting in which a trusted third party \( \text{cp} \) executes the crowdworking process \( \Pi \) of \( \sigma \):

\[
\text{REAL}_{\sigma, \chi}(W, P, R, T) \equiv_{c} \text{IDEAL}_{\text{cp}, \chi}(W, P, R, T)
\]

where \( i \in \{-R-I, R-I, RI\} \), and \( \text{REAL} \) denotes the adversarial knowledge in the real setting and \( \text{IDEAL} \) its counterpart in the ideal setting.

### 3.4 Problem

We address the problem of designing the SEPAR system in charge of allowing the execution of crowdworking processes (1) while guaranteeing together the correctness of the constraints defined by external institutions and the privacy of participants against covert adversaries, over (2) distributed crowdworking platforms that communicate through the local, cross-platforms, and global consensuses.

### 4 Expressing Global Regulations

We express global regulations using constraints and certificates. A constraint demonstrates a limit on the actions that can be performed by the participants of the crowdworking environment and a certificate is a piece of information that participants can provide to third parties to prove that they took part in a given crowdworking process. In this Section, we express both constraints and certificates.

#### 4.1 Expressing Constraints

**Syntax.** We define a constraint \( c \) as being essentially (1) a triple \((w, p, r)\) that associates a worker \( w \), a platform \( p \), and a requester \( r \), and (2) a threshold \( \theta \) (an integer) that defines the upper bound of \( c \).

Intuitively, a constraint \(((w, p, r), \theta)\) states that there must not be more than \( \theta \) actions between the worker \( w \), the platform \( p \), and the requester \( r \) (see below for the detailed semantics). We also allow two wildcards to be written in any position of a triple: \( * \) and \( \forall \). First, the \( * \) wildcard allows to ignore one or more elements of a triple. For example, \((*, p, r)\) means that the constraint applies to the couple \((p, r)\). A triple may contain up to three \( * \) wildcards. An element of a triple that is not a \( * \) wildcard is called a specified participant of the constraint. Second, the \( \forall \) wildcard factorizes the writing of triples because it allows to express a constraint that must hold for all participants in the same group of participants. For example, \((\forall, p, r)\) represents the following set of triples: \( \{(w, p, r)\} \), \( \forall w \in W \). We denote \( C \) the complete set of constraints.

**Semantics.** We give now a precise definition of the semantics of our constraints by illustrating how they translate to SQL constraints. Let assume that there exists a table of actions \( A \) that records all actions performed between any triple of worker, platform, requester. The attributes of \( A \) are WORKER, PLATFORM, REQUESTER. For simplicity, we consider a constraint \( c \) without any wildcard, i.e., \( c \leftarrow ((w, p, r), \theta) \). The semantics of \( c \) is the same as the following SQL query:

\[
\begin{aligned}
\text{SELECT } & \text{worker, platform, requester} \\
\text{FROM } & \text{actions} \\
\text{WHERE } & \text{worker} = w \land \text{platform} = p \land \text{requester} = r \\
\text{AND } & \text{number_of_actions} \leq \theta
\end{aligned}
\]
ALTER TABLE A ADD CONSTRAINT c CHECK (  
    NOT EXISTS (  
        SELECT *  
        FROM A  
        WHERE WORKER = w AND PLATFORM = p AND REQUESTER = r  
        GROUP BY WORKER, PLATFORM, REQUESTER  
        HAVING COUNT(*) ≥ θ  
    )  
);  

The presence of a * wildcard in the triple simply leads to removing the corresponding attributes in the WHERE and GROUP BY clauses. The presence of a ∀ wildcard leads to expanding it to the set containing all the elements that it represents (e.g., all workers if the ∀ wildcard is at the first position in the triple) and to generate the cartesian product between the resulting set and the elements at the two other positions of the triple (that may be ∀ wildcards as well). Finally, the semantics of a set of constraints is the conjunction of the constraints contained in the set.  

Example. the weekly FLSA limit on the total work hours per worker can easily be expressed as \( c_{FLSA} \leftarrow (\forall, *, *), 40 \).  

4.2 Expressing Requests for Certificates  

Syntax and Semantics. Certificates allow a participant called prover (e.g., worker) to prove to an external entity called verifier (e.g., social security agency) that a minimal number of hours have been spent on crowdworking platforms (e.g., for applying to insurance coverage). Requests for certificates (e.g., from social security agencies) are expressed using the same syntax as the constraints with the following two differences. First, the \( \theta \) threshold does not represent an upper bound on actions that cannot be exceeded, but a lower bound on actions that have to be proved. And second, there must always be at least one specified participant in a request for certificates, i.e., typically the prover. This syntax allows verifiers to follow minimal disclosure principles by requesting from the prover exactly the information needed about the crowdworking processes performed. There is no need to request the identities of the participants with whom the prover collaborated. Additionally, it is trivial to connect multiple requests for certificates through conjunctions and disjunctions if needed. Examples. A social security institution can request each worker \( w \) applying for insurance coverage to prove that she worked in total more than 5 hours: \( r_1 \leftarrow ((w, *, *), 5) \) is both necessary and sufficient. Similarly, the request \( r_2 \leftarrow ((*, p, *), 1000) \) allows a tax institution to ask for each platform \( p \) applying for a tax refund to prove that the total work hours of all its workers is at least 1000 hours.  

5 Enforcing Global Regulations  

In this section, we develop our conception of constraints and certificates, how they are built and how to use them, and we prove the correctness and the privacy guarantees of our construction.  

5.1 Implementing a Token-Based System  

In this section we show how constraints and certificates, which are expressed in Section 4, can be enforced and produced respectively. Inspired by e-cash systems, we enforce constraints and produce certificates by managing two budgets per participant while preserving both the privacy of participants and the correctness of budgets. Our proposal makes use of a centralized authority, called the registration authority (RA for short). RA registers the participants to the crowdworking environment, sends them the required cryptographic material, receives the set of constraints, and manages the budgets. The required cryptographic material includes a pair of public/private asymmetric keys (e.g., RSA) and a pair of public/private asymmetric group keys (e.g., [11]) for which the registration authority is the group manager, while the set of constraints may be expressed by the regulators through a dedicated interface. We instantiate the budgets based on labeled, single-use, anonymous tokens and use a persistent transparent datastore to guarantee their correct and validated spending by participants. The persistent datastore is implemented using a blockchains. To process crowdworking tasks, our token-based system is defined by five functions: GENERATE for initializing the budgets and refilling them, SPEND for spending portions of the budgets, PROVE for providing certificates to a third party, CHECK for checking whether a given spending is allowed or not, and ALERT for reporting dubious spending.
5.1.1 The GENERATE Function

The registration authority uses the GENERATE function to initialize the budgets, i.e., constraint and certificate tokens of all participants (i.e., workers, platforms, requesters) and refill them periodically\(^{11}\) according to the set of constraints \(\mathcal{C}\) to enforce.

**Constraint tokens.** For each constraint \(c \in \{(w,p,r),\theta\}\), the registration authority generates \(\theta\) tokens and sends a copy of each token to each specified participant of \(c\). A token consists of a public and a private component. The public component is a pair made of a number used only once (referred to as a nonce below) generated by the registration authority and a signature of the nonce produced by the registration authority\(^{12}\). The public component will be used later (upon completion of the corresponding task) by other platforms to check the validity of tokens. The private component is an index allowing the participants involved in a crowdfunding process to select the correct set of tokens given the other specified participants involved in the process. We implement this private component as a list containing the public keys of the specified participants (in the corresponding constraint)\(^{13}\). Let \(tk^\dagger\) be a constraint token, \(tk^\dagger_{pub}\) be its public component, \(tk^\dagger_{priv}\) be its private component, \(N\) be a nonce, and \(pubs\) the list of public keys. The constraint token is thus the couple \((tk^\dagger_{pub},tk^\dagger_{priv})\) where \(tk^\dagger_{pub} = (N,Sign(key_{priv,RA},(N)))\) and \(tk^\dagger_{priv} = pubs\).

**Certificate Tokens.** Certificate tokens are generated initially by the registration authority for all participants. For each crowdfunding process, a single certificate token is linked to a fully specified triplet of participants \((w,p,r)\). The number of certificate tokens produced is decided initially, but their quantity is not as easy to decide as for constraint tokens since it is not capped by a \(\theta\) threshold. For simplicity, we assume that there is at least one constraint in the system, and the smallest threshold for all constraints is \(\theta_{min}\). Then, \(\theta_{min}\) is a sufficient upper bound of the number of crowdfunding processes in which any given triplet of participants is involved. It is therefore enough to produce \(\theta_{min} \times |W| \times |P| \times |R|\) certificate tokens. In practice, the number of tokens produced can be drastically reduced in a straightforward manner by letting participants declare to RA the subset of participants they may work with (e.g., selecting a subset of platforms, or domains of interest).

As stated above, a certificate token always relates to a fully specified triplet \((w,p,r)\) and to its owner \(o\) (i.e., one of the participants in the triplet). Similar to a constraint token, it consists of a public and a private component. The public component consists of a nonce as well as the signature of the nonce produced by the registration authority. The private component, on the other hand, is a triplet in which each element certifies (i.e., signs) the association between the owner \(o\) and another specified participant. More formally, let \(tk^*\) be a certificate token, \(tk^*_{pub}\) be its public part, \(tk^*_{priv}\) be its private part, \(N\) be a nonce, \(o\) be the identity of the participant owner of the token, and \((w,p,r)\) be the related triplet. The certificate token is thus the pair \((tk^*_{pub},tk^*_{priv})\) where \(tk^*_{pub} = (N,Sign(key_{priv,RA},(N)))\) and \(tk^*_{priv} = (Sign(key_{priv,RA},(N,o,w)), Sign(key_{priv,RA},(N,o,p)), Sign(key_{priv,RA},(N,o,r)))\).

5.1.2 The SPEND Function

Requesters create and send their tasks to a platform and the platform submits the tasks in either its own datastore (for local tasks) or all involved platforms (for cross-platform tasks). Once the task is published, the workers can indicate their intent to perform the task by sending a contribution intent to their platforms. If a contribution is still needed for the task, the SPEND function is performed as follow. First, for a given constraint \(c \in \mathcal{C}\), the platform requests the public component of a constraint token corresponding to \(c\) from the initiator (one of the specified participants in constraint \(c\)). For certificates, the platform is the initiator. The platform includes the task, an identifier for the contribution (e.g., a nonce generated by the platform), and a signature of the identifier concatenated to the task in its request message. Therefore, workers and requesters will be able to prove that they were asked for tokens, even if the platform fails. The initiator then chooses a token to spend and sends it to the platform. Once the platform receives the required token, it sends the public component of the constraint token to all the specified participants. For certificates, the

---

\(^{11}\) The refreshment rates of budgets is easily computed from the validity periods of constraints (see Section 4).

\(^{12}\) Extending tokens with labels and/or timestamps for supporting the validity periods of constraints is straightforward.

\(^{13}\) The use of a public key generated by the registration authority is important here because (1) it can be shared among participants without disclosing their identities, i.e., it is a pseudonym, (2) the corresponding private key can be used by participants for mutual authentication in order to guarantee the correctness of the index and consequently of the choice of tokens.
platform sends the public part of the certificate tokens to all participants of the process. The platform also requires them to send back two signatures: (1) the group signature of the token (which will be later verified by all platforms, together with the token, when it is shared with all platforms), and (2) the group signature of the pair consisting of a token and a task. Note that the second signature while it does not reveal the task by itself, can be used by participants to verify that tokens are used on the task they are intended to be used on. Again, this demand is associated with the task and the identifier of the contribution, and is signed by the platform.

Finally, for each task, a transaction consisting of all spent constraint tokens from each specified participant (all spent certificate tokens from every participant in the case of certificates) is committed to the datastore of all platforms. For each token, the transaction includes first, the public component of each token, second, the group signature of the public component of each token (i.e., for a constraint token $tk^†$: $GroupSign(key_{priv,part}, Group, tk^†_{pub})$), and third, the group signature of the public part of each token together with the associated task $t$ (i.e., for a constraint token $tk^†$: $GroupSign(key_{priv,part}, Group, (t, tk^†_{pub}))$).

### 5.1.3 The PROVE function

The PROVE function is used by participants to provide certificates to a third party. The use of certificate tokens is relatively straightforward. During the crowdworking processes, participants store the private components of certificate tokens which will be used later to deliver certificates on demand. A participant indeed initiates the PROVE function by sending the related subpart(s) of the private component of the corresponding tokens to the verifier. As an example, for a $((w, *), 5)$ request for certificates, the worker $w$ sends the subparts containing $w$ from the private parts of all 5 certificate tokens. The verifier, first, checks the signature of the registration authority to verify that the participant was involved in the task, and then, checks the nonce stored in the datastore to ensure that the token has been shared and validated by all platforms.

### 5.1.4 The CHECK and ALERT Functions

The CHECK and ALERT functions are used to detect and report either the malicious behavior of participants resulting in an invalid consumption of tokens or the failure of a platform. The complete set of verifications protects against (1) the forgery of tokens (verification of the signatures), (2) the replay of tokens (verification of the absence of double-spending), (3) the relay of tokens (verification of the absence of usurpation), and (4) the illegitimate invalidation of tokens (timeout against malicious platform failures). The first two verifications are straightforward and performed during the global consensus (when all platforms can access tokens and signatures). We explain the last two verifications.

**Usurpation.** When a token is appended to the datastore of all platforms, anyone (whether involved in the corresponding crowdworking process or not) can CHECK its nonce. If a participant detects a nonce that was received from the registration authority but not spent $^{14}$, she ALERTs the registration authority. The registration authority will de-anonymize the group-signature of the corresponding participant (e.g., the worker’s group signature if the alert comes from a worker), and checks whether it has been signed by the same participant that sent the token. Similarly, if a participant detects that a token has not been spent on the right task, she ALERTs the registration authority. After an alert, the registration authority has to act either against the target participant of the alert (true positive) or against the participant originating the alert (false positive). The possible actions (e.g., ban the participant) depend on the context.

**Platform failure.** If a platform fails after it requested tokens or signatures and does not recover (e.g., tokens are not appended to the datastore), the tokens revealed to the platform are lost: they cannot be used in any other crowdworking process because they are not anonymous anymore (i.e., the platform knows the association between them and the corresponding participants), and they are not spent either. In that case, workers or requesters send an ALERT to the registration authority including (1) the identifier of the platform, (2) the identifier of the task, and (3) all the requests received from the platform. The registration authority then checks whether the number of requests sent by the platform for the given task matches the corresponding number of messages committed in the datastore. If there are more requests, the registration authority sets a timeout (e.g., to let unfinished transactions end or the platform recover from a failure). When the timeout is over, the registration authority can act against the platform.

---

$^{14}$For example, a platform $p$ can collude with a worker $w_1$ to spend a token dedicated to a $(w_2, p, *)$ constraints.
5.2 Task Processing Sequence

In summary, five main phases exist during the processing of a crowdworking task are: (1) initialization, (2) publication, (3) assertion, (4) verification, and (5) execution.

Initialization. The registration authority provides all parties with their keys and tokens.

Publication. Requesters create and send their tasks to platforms. If a requester wants to publish its task on more than one platform (i.e., a cross-platform task), the involved platforms collaborate with each other to create a common instance of the task. The involved platforms then publish the tasks on their datastores through submission transactions and inform their workers in their preferred manner for accessing tasks.

Assertion. After a worker has retrieved a task, the worker sends a contribution intent message to the platform without revealing the actual contribution. The platform then updates the number of required contributions for the task and publishes the contribution intent in its datastore through a claim transaction. For cross-platform tasks, the platform informs other involved platforms about the received contribution intent, so that all involved platforms agree with the number (and order) of the received contribution intents (i.e., claim transactions). If the desired number of contribution for the task has been achieved, the process is aborted. Note that while the requester does not choose the workers, it is possible to enforce a selection with a priori criteria, passed through the platform. Another straight-forward enhancement would be to add a communication step, by forwarding the contribution intent, together with the worker’s identity to the requester, and letting her approve of it or not. This communication, however, requires the disclosure of the worker’s identity to requesters even before a contribution is accepted.

Verification. Once the contribution intent has been accepted by the platform(s), the platform asks the corresponding requester and worker to send the required tokens and signatures, through the SPEND function, developed in Section 5.1.2. Upon receiving all tokens and signatures, the platform shares them with all platforms and the tokens and their signatures are published to the datastores through verification transactions. From this point, anyone can check the validity of requirements with the CHECK function (and ALERT if required), as developed in Section 5.1.4.

Execution. Once all parties have checked the validity of the task, its tokens and group signatures, the actual contribution can be given to the requester and reward to the worker through the platform.

A sequence chart of this protocol is provided in Figure 2.

5.3 Privacy Analysis

We show below in the suite of Theorem 1, Lemma 1, Lemma 2, and Theorem 2 that the global execution of SEPAR satisfies the $\delta^{11}$-private model against covert adversaries.

First, Theorem 1 restricts the adversarial behavior to inferences (i.e., similar to a honest-but-curious adversary) and shows that the execution of SEPAR satisfies $\delta^{11}$-privacy. Second, we extend the possible
behaviors to malicious behaviors aiming at jeopardizing the enforcement of constraints and of requests for certificates, and show that they are systematically detected by SEPAR (Lemma 1 focuses on constraints and Lemma 2 on certificates). This prevents covert adversaries to perform malicious actions, limiting them to inferences. Since Theorem 1 shows that SEPAR is $\delta^{\Pi}$-private against adversaries restricted to inferences, and Lemma 1 and Lemma 2 show that malicious behaviors are prevented, it follows that SEPAR is $\delta^{\Pi}$-private against covert adversaries (Theorem 2).

**Theorem 1** (Privacy (inferences)) For all sets of crowdworking processes $\Pi$ executed over participants $\mathcal{W}$, $\mathcal{P}$, and $\mathcal{R}$ by an instance of SEPAR $\sigma$, then it holds that $\sigma$ is $\delta^{\Pi}$-private against covert adversaries restricted to inferences.

**Proof:** ([sketch]) First, we focus on the content of tokens and show that it is harmless. For each crowdworking process $\pi$, the information contained within the tokens exchanged and stored in the datastore (shared by all platforms) is made of (1) the public parts $tk_{\text{pub}}$ of the tokens involved (i.e., a nonce and a signature) and (2) of the group signatures of the participants to $\pi$. The nonce is generated by the registration authority independently from $\pi$, thus do not leak any information about $\pi$. Since the group signatures are generated by a semantically-secure group signature scheme, they do not leak anything to the real adversary (computationally bounded) beyond the groups of the signers, which is also available to the ideal adversary.

Second, we concentrate on the information disclosed along the execution sequence of crowdworking processes. We consider below each disclosure set $\delta^{\pi}_{\mathcal{R}_{\rightarrow}I}$ in turn and show that the computational indistinguishability requirement between the ideal setting and the real setting (where the instance of SEPAR $\sigma$ executes $\Pi$) is satisfied in all cases.

**Disclosure set $\delta^{\pi}_{\mathcal{R}_{\rightarrow}I}$**. The $\delta^{\pi}_{\mathcal{R}_{\rightarrow}I}$ disclosure set contains the information allowed to be disclosed to the participants that are not involved in a crowdworking process $\pi \in \Pi$ and that have not received the related task (i.e., $(\text{BEGIN}, \text{END}, p)$). First, we focus on the subset of such participants that are requesters or workers. In the ideal settings, these participants learn nothing beyond the information contained in $\delta^{\pi}_{\mathcal{R}_{\rightarrow}I}$. In the real setting, when $\pi$ is executed by $\sigma$, these participants are not involved in any consensus. They are only able to observe the state of the datastore. The latter is updated exactly once for $\pi$, when $\pi$ ends, for storing the tokens spent: only the ending event (END) is disclosed, which is already contained within $\delta^{\pi}_{\mathcal{R}_{\rightarrow}I}$. Second, we focus on the platforms. In the ideal setting, they are given $\delta^{\pi}_{\mathcal{R}_{\rightarrow}I}$. In the real setting, they participate to the global consensus and are able to observe the state of the datastore. Consequently, they learn $p$ from the global consensus (i.e., the platform that initiates the global consensus) and the ending event END. Both are already contained within $\delta^{\pi}_{\mathcal{R}_{\rightarrow}I}$.

**Disclosure set $\delta^{\pi}_{\mathcal{R}_{\rightarrow}I}$**. The $\delta^{\pi}_{\mathcal{R}_{\rightarrow}I}$ disclosure set contains the information allowed to be disclosed to the workers and platforms that have received the task $t$ from $r$ but that are not involved in the crowdworking process $\pi \in \Pi$ (i.e., $(\text{BEGIN}, \text{END}, p, r, t)$). First, we focus on the subset of such participants that are workers. In the ideal settings, they learn nothing beyond the information contained in $\delta^{\pi}_{\mathcal{R}_{\rightarrow}I}$. In the real setting, we see from the global execution sequence of SEPAR that (1) they are not involved in any consensus, (2) but are able to observe the state of the datastore, (3) have received the task $t$, and (4) may receive an abort from their platform if they contribute to $t$, while $t$ has already been solved. From (2) and (4), they are able to learn the ending events of crowdworking processes, from (3) they learn $t$, and from (1) they do not learn any other information. As a result, they learn (END, $t$), which is contained in $\delta^{\pi}_{\mathcal{R}_{\rightarrow}I}$. Second, we focus on the subset of participants that are platforms. In the ideal settings, they learn nothing beyond the information contained in $\delta^{\pi}_{\mathcal{R}_{\rightarrow}I}$. In the real setting, (1) they receive the tasks from the requesters, (2) they participate to the cross-platform consensuses and to the global consensuses in addition to (3) observing the same information as the workers. From (1) they learn $t$ and $r$ for each process $\pi \in \Pi$. From (2), they learn $p$, BEGIN, and END because: the cross-platform consensus discloses the initiating platform and the starting event, and the global consensus discloses the initiating platform together with the ending event. From (3), they learn the ending event and the task. As a result, such platforms learn about all $\pi \in \Pi$ the following information (BEGIN, END, $r, p, t$), which is exactly $\delta^{\pi}_{\mathcal{R}_{\rightarrow}I}$.

13Including a generation timestamp into the public part of tokens, for supporting the validity periods of constraints, would not leak information about $\pi$ beyond its probable execution timeframe, which is already captured by the BEGIN and END events allowed to leak in the $\delta^{\Pi}$-confidentiality model. Indeed, the timestamps would be generated by the registration authority independently from $\pi$.

14The verifications performed by the worker and the requester involved in $\pi$ are performed on the instance of the datastore stored on $p$. Indeed, the block resulting from the global consensus contains all the necessary information both for performing the verification and for checking that it results from the global consensus.
Disclosure set $\delta_{RI}^\pi$. The computational indistinguishability requirement between the real setting and the ideal setting is trivially satisfied for the $\delta_{RI}^\pi$ set of disclosures because $\delta_{RI}^\pi$ contains all the information about the execution of the crowdworking process $\pi$ so $\sigma$ does not (and cannot) disclose more information.

**Lemma 1** (Detection of malicious behaviors (constraints)) A crowdworking process $\pi$ executed over participants $W$, $P$, and $R$ by an instance of SEPAR $\sigma$, completes successfully without rising a legitimate alert if and only if it does not jeopardize any constraint.

**Proof:** (sketch) We have to show first that the constraint tokens allocated to participants can be spent, and second that participants cannot spend more.

**Participants can spend their tokens.** In order to prevent participants from spending their constraint tokens, an attacker has three main possibilities. First, she can try to acquire tokens belonging to another participant and to spend them (i.e., relay attack). However, this rises an ALERT with certainty. Indeed, if a token is spent by an illegitimate participant, it is stored on the datastore and is thus accessible to the legitimate participant who is able to detect it through the nonce and to rise an ALERT to the registration authority (including the group signatures stored along the token). Second, the attacker (platform only) could try to misuse tokens by spending them in a way that was not intended by the legitimate owner (i.e., relay attack). However, this would be detected by participants as well because the signature of the task would not be valid. Third, the attacker (platform only) may abort the process after having received tokens but before performing the global consensus (i.e., illegitimate invalidation). However, after a timeout, the other involved participants simply send an ALERT to the registration authority and prove that their tokens were requested by the platform (signatures of the requests for tokens and signatures), and therefore that the platform behaves illegitimately.

**Participants cannot spend more.** First, an attacker may produce additional tokens (i.e., forge attack). However, the public parts of tokens must contain valid signatures produced by the registration authority. Second, an attacker may try to spend a token more than once (i.e., replay attack). However, the nonce of a token that must be spent must not already be in the datastore. Finally, an attacker may simply omit sending any token. However, the public parts of tokens are required for the successful completion of the global consensus.

**Lemma 2** (Detection of malicious behaviors (certificates)) A participant can produce a certificate about a crowdworking process $\pi$ executed over participants $W$, $P$, and $R$ by an instance of SEPAR $\sigma$, if and only if (1) she was involved in $\pi$ and (2) $\pi$ completes successfully.

**Proof:** (sketch) The correctness of our token-based system for answering to requests for certificates is trivial to demonstrate. First, certificates about a crowdworking process $\pi$ that completed successfully can always be produced by the participants involved in $\pi$. Indeed, the certificate tokens are produced by the registration authority and sent to all participants (i.e., the number of certificate tokens is correct), and only the successful crowdworking processes store the certificate tokens in the datastore. Second, participants cannot spend a certificate token more than once because the nonce of a token that must be spent must not already be in the datastore. Third, participants cannot produce any certificate token by themselves because their public parts must contain valid signatures produced by the registration authority.

**Theorem 2** (Privacy (inferences and malicious behaviors)) For all sets of crowdworking processes $\Pi$ executed over participants $W$, $P$, and $R$ by an instance of SEPAR $\sigma$, then it holds that $\sigma$ is $\delta^{11}$-private against covert adversaries.

**Proof:** Theorem 1 shows that the execution of SEPAR satisfies $\delta^{11}$-privacy against covert adversaries restricted to inferences, while Lemma 1 and Lemma 2 shows that malicious behaviors aiming at jeopardizing the guarantees token-based system are detected and consequently prevented within SEPAR. As a result it follows directly that the execution of SEPAR satisfies $\delta^{11}$-privacy against covert adversaries.

### 6 Coping with Distribution

SEPAR is a multi-platform crowdworking system where multiple globally distributed platforms collaborate with each other to process crowdworking tasks. To realize such distributed collaborations and due to the unique features of permissioned blockchains such as transparency and provenance, which are needed by crowdworking applications, SEPAR is deployed on a permissioned blockchain to implement the persistent
In this section, we first present the distributed blockchain ledger of SEPAR and then, show how SEPAR establishes consensus on the order of transactions within and across different platforms.

### 6.1 Blockchain Ledger

In a blockchain, transactions are recorded in an append-only data structure, called **Blockchain ledger**. The blockchain ledger in SEPAR includes all submission, claim, and verification transactions of all internal as well as cross-platform tasks. To ensure data consistency, an ordering among transactions in which a platform is involved is needed. The total order of transactions in the blockchain ledger is captured by chaining the transactions (blocks) together, i.e. each transaction block includes a sequence number or the cryptographic hash of the previous transaction block. Since SEPAR supports both internal and cross-platform tasks and more than one platform are involved in each cross-platform transaction, similar to [2,3], the ledger is formed as a **directed acyclic graph (DAG)** where the **nodes** of the graph are transaction blocks (each block includes a single transaction) and **edges** enforce the order among transaction blocks. In addition to submission, claim, and verification transactions, a unique initialization transaction (block), called the **genesis** transaction is also included in the ledger.

Fig. 3(a) shows a blockchain ledger created in the SEPAR model for a blockchain infrastructure consisting of four platforms $p_1$, $p_2$, $p_3$, and $p_4$. In this figure, $\lambda$ is the genesis block of the blockchain, $t_i$’s are submission transactions, $t_i c_j$ is the $j$-th claim transaction of task $t_i$, and $t_i v$ is the verification transaction of task $t_i$. In Fig. 3(a), $t_{10}$, $t_{20}$, $t_{30}$, and $t_{40}$ are internal submission transactions of different platforms. In SEPAR, as can be seen, the internal transactions of different platforms can be appended to the ledger in parallel. $t_{10} c_1$, $t_{10} c_2$, …. and $t_{40} c_2$ are the corresponding claim transactions. As shown, $t_{10}$ requires 3 contributions (thus 3 claim transactions) whereas each of $t_{20}$, $t_{30}$, and $t_{40}$ needs two contributions. $t_{10} v$, $t_{20} v$, $t_{30} v$, and $t_{40} v$ are also the verification transactions. $t_{11} c_1$ is a cross-platform submission among platforms $p_1$ and $p_2$. Similarly, $t_{31} c_1$ is a cross-platform submission among platforms $p_3$ and $p_4$. Here, $t_{11} c_1$ needs a single contribution and $t_{31} c_1$ requires two contributions. Note that the claim transactions of a cross-platform task might be initiated by different platforms and as mentioned earlier, the order of these claim transactions is important (to recognize the $n$ first claims). Finally, $t_{22} c_2$ is a cross-platform task among platforms $p_2$, $p_3$, and $p_4$ that is processed in parallel to the internal task $t_{12}$ of platform $p_1$. 

![Blockchain Ledger Diagram]
The introduced blockchain ledger includes all transactions of internal as well as cross-platform tasks initiated by all platforms. However, due to the data privacy requirement, each platform must access only a subset of these transactions, i.e., the transactions in which the platform is involved. One way to achieve data privacy is to encrypt all transactions using cryptographic techniques and keep an identical blockchain ledger on every platform. However, the considerable overhead of such techniques results in low performance [4], and in addition, each platform will store many transactions that are not relevant to the platform. As a result and for the sake of performance, in SEPAR, the entire blockchain ledger is not maintained by any platform and each platform only maintains its own view of the blockchain ledger including (1) all submission and claim transactions of its internal tasks, (2) all submission and claim transactions of the cross-platform tasks that the platform is involved in them, and (3) verification transactions of all tasks. Note that verification transactions are replicated on every platform to enable all platforms to check the satisfaction of constraints. The blockchain ledger is indeed the union of all these physical views.

Fig. 3(b)-(e) show the views of the blockchain ledger for platforms $p_1$, $p_2$, $p_3$, and $p_4$ respectively. As can be seen, each platform $p_i$ maintains only submission and claim transactions of all internal tasks as well as cross-platform tasks that $p_i$ is involved in them and verification transactions of all tasks. For example and as shown in Fig. 3(b), platform $p_1$ maintains all transactions of its two internal tasks $t_{10}$ and $t_{12}$. These are either submission transactions, i.e., $t_{10}c_1$, $t_{10}c_2$, $t_{10}c_3$, $t_{12}c_1$, $t_{12}c_2$, or verification transactions, i.e., $t_{10}v$, $t_{12}v$. Platform $p_1$ also maintains cross-platform transactions that $p_1$ is involved in, i.e., $t_{11,21}$, $t_{11,21}c_1$, and $t_{11,21}v$. Finally, $p_1$ maintains the verification transactions of all other tasks within the system, i.e. $t_{20}v$, $t_{30}v$, $t_{40}v$, $t_{31,41}v$, and $t_{22,32,42}v$. Note that, since there is no data dependency between the verification transactions of the tasks that a platform is not involved in and the transactions of the tasks that a platform is involved in, the verification transactions might be appended to the ledgers in different orders, e.g., $t_{20}v$ (of platform $p_2$) and $t_{40}v$ (of platform $p_4$) are appended to the ledger of platforms $p_1$ and $p_3$ in two different orders.

6.2 Consensus in SEPAR

In SEPAR, each platform consists of a (disjoint) set of nodes (i.e., replicas) where the platform replicates its own view of the blockchain ledger on those nodes to achieve fault tolerance. Nodes follow either the crash or Byzantine failure model. In the crash failure model, nodes operate at arbitrary speed, may fail by stopping, and may restart, however, in the Byzantine failure model, faulty nodes may exhibit arbitrary, potentially malicious, behavior. Nodes of the same or different platforms need to establish consensus on a unique order in which entries are appended to the blockchain ledger. To establish consensus among the nodes, asynchronous fault-tolerant protocols have been used. Fault-tolerant protocols use the State Machine Replication (SMR) algorithm [23] to guarantee safety where nodes agree on an ordering of incoming transactions. Crash fault-tolerant protocols guarantee safety in an asynchronous network using $2f+1$ nodes to overcome the simultaneous failure of any $f$ nodes while in Byzantine fault-tolerant protocols, $3f+1$ nodes
are usually needed to provide the safety property in the presence of $f$ malicious nodes. Figure 4 shows the crowdworking infrastructure of Figure 1 where each platform consists of 4 replicas (assuming Byzantine failure model and $f = 1$) and replicas use a blockchain to store data.

Completion of a crowdworking task, as discussed earlier, requires a single submission, one or more claim, and a verification transaction. For an internal task of a platform, submission and claim transactions are replicated only on the nodes of the platform, hence, local consensus among nodes of the platform on the order of the transaction is needed. For a cross-platform task, on the other hand, submission and claim transactions are replicated on every node of all (and only) involved platforms. As a result, cross-platform consensus among the nodes of all involved platforms is needed. Finally, verification transactions will be appended to the blockchain of all platforms, therefore, all nodes of every platform participate in a global consensus protocol. In this section, we show how local, cross-platform, and global consensus are established in the presence of crash-only or Byzantine nodes.

6.2.1 Local Consensus

Processing a submission or a claim transaction of an internal task requires local consensus where nodes of a single platform, independent of other platforms, establish agreement on the order of the transaction. The local consensus protocol in SEPAR is pluggable and depending on the failure model of nodes, i.e., crash-only or Byzantine, a platform uses a crash fault-tolerant protocol, e.g., Paxos [24], or a Byzantine fault-tolerant protocol, e.g., PBFT [12]. Figure 5 shows the normal case operation of both Paxos and PBFT protocols.

The local consensus protocol is initiated by a pre-elected node of the platform, called the primary. When the primary $p$ receives a valid internal transaction (either submission or claim), it initiates a local consensus algorithm by multicasting a message, e.g., accept message in Paxos or pre-prepare message in PBFT, including the requested transaction to other nodes of the platform. To provide a total order among transactions, the primary also assigns a sequence number to the request. Instead of a sequence number, the primary can also include the cryptographic hash of the previous transaction block in the message. If the transaction is a claim transaction, the primary includes the cryptographic hash of the corresponding submission transaction and any previously received claim transactions for that particular task (if any). The nodes of the platform then establish agreement on a total order of transactions using the utilized consensus protocol and append the transaction to the blockchain ledger.

6.2.2 Cross-Platform Consensus

Submission and claim transactions of a cross-platform task must be appended to the blockchains of all involved platforms in the same order to ensure data consistency. To process such transactions, therefore, consensus among the nodes of all (and only) involved platforms is needed. SEPAR addresses the lack of trust in the collaboration between platforms, by using an asynchronous Byzantine fault-tolerant protocol to establish consensus on the order of cross-platform transactions. Since the number of nodes of each platform depends on the utilized consensus protocol within the platform (i.e. crash fault-tolerant protocols require $2f + 1$ whereas Byzantine fault-tolerant protocols require $3f + 1$ nodes), the required number of matching replies from each platform, i.e., the quorum size, to ensure the safety of protocol depends on the failure model of nodes of the platform. We define local-majority as the required number of matching replies from the nodes of a platform. For a platform with crash-only nodes, local-majority is $f + 1$ (from the total $2f + 1$ nodes), whereas for a platform with Byzantine nodes, local-majority is $2f + 1$ (from the total $3f + 1$ nodes).

SEPAR processes cross-platform transactions in four phases: prepare, propose, accept, and commit. Upon receiving a cross-platform (submission or claim) transaction, the (pre-elected) primary node of the (recipient)
Algorithm 1 Cross-Platform Consensus

1: init():
2: \( r := \text{node\_id} \)
3: \( p_i := \text{the platform that initiates the consensus} \)
4: \( \pi(p) := \text{the primary node of cluster} \)
5: \( P := \text{the set of involved platforms} \)
6: \( \pi(P) := \text{the primary nodes of clusters in} \ P \)
7: upon receiving valid transaction \( m \) and \( (r == \pi(p_i)) \) do:
8: \( \text{multicast } \{\langle \text{PREPARE, } h_i, d \rangle_{\pi(p_i)}, m \} \text{ to } \pi(P) \)
9: \( \text{multicast } \{\langle \text{PROPOSE, } h_i, d \rangle_{\pi(p_i)}, m \} \text{ to all nodes of } p_i \)
10: upon receiving valid \( \mu := \{\langle \text{PREPARE, } h_i, d \rangle_{\pi(p_i)}, m \} \) and \( r == \pi(p_j) \) do:
11: if \( r \) is not involved in any uncommitted request \( m' \) where \( m \) and \( m' \) intersect in some other platform \( p_k \) do:
12: \( \text{multicast } \{\langle \text{PROPOSE, } h_j, d, r \rangle_{\pi(p_j)}, \mu \} \text{ to all nodes of } p_j \)
13: upon receiving valid \( \langle \text{PREPARE, } h_i, d \rangle_{\pi(p_i)}, m \) and \( r \in p_i \) do:
14: \( \text{multicast } \{\langle \text{ACCEPT, } h_i, h_j, d, r \rangle_{\pi(p_j)}, \mu \} \text{ to } P \)
15: upon receiving valid \( \langle \text{PROPOSE, } h_i, d \rangle_{\pi(p_i)}, m \) and \( r \in p_j \) do:
16: \( \text{multicast } \{\langle \text{ACCEPT, } h_i, h_j, d, r \rangle_{\pi(p_j)}, \mu \} \text{ to } P \)
17: upon receiving valid matching \( \{\text{ACCEPT, } h_i, h_j, d, r \rangle_{\pi(p_j)}, \mu \} \) from local-majority of every platform \( p_j \) in \( P \) do:
18: \( \text{multicast } \{\langle \text{COMMIT, } h_i, h_j, ..., h_k, d, r \rangle_{\pi(P)}, \mu \} \text{ to } P \)
19: \( \text{append the transaction block to the ledger} \)

platform initiates the consensus protocol by multicasting a \text{prepare} message to the primary node of all involved platforms. Each primary node then assigns a sequence number to the request and multicasts a \text{propose} message to every node of its platform. During the \text{accept} and \text{commit} phases, all nodes of every involved platform communicate to each other to reach agreement on the order of the cross-platform transaction.

Algorithm 1 presents the normal case of \text{cross-platform consensus} in SEPAR. Although not explicitly mentioned, every sent and received message is logged by nodes. As shown in lines 1-6 of the algorithm, \( p_i \) is the platform that initiates the transaction, \( \pi(p) \) represents the primary node of platform \( p_i \). \( P \) is the set of involved platforms in the transaction where \( \pi(P) \) represents their current primary nodes (one node per platform).

Once the primary \( \pi(p_i) \) of the initiator platform \( p_i \) receives a valid submission or claim transaction, as presented in lines 7-8, the primary node assigns sequence number \( h_i \) to the request and multicasts a \text{signed prepare} message \( \langle \langle \text{PREPARE, } h_i, d \rangle_{\pi(p_i)}, m \rangle \) to the primary nodes of all involved platforms where \( m \) is the received message (either submission or claim) and \( d = D(m) \) is the digest of \( m \). The sequence number \( h_i \) represents the correct order of the transaction block in the initiator platform \( p_i \). If the transaction is a claim transaction, the primary includes the cryptographic hash of the corresponding submission transaction as well. As shown in line 9, the primary node also multicasts a \text{signed propose} message \( \langle \langle \text{PROPOSE, } h_i, d \rangle_{\pi(p_i)}, m \rangle \) to the nodes of its platform where \( d = D(m) \) is the digest of \( m \).

As indicated in lines 10-12, once the primary node of some platform \( p_j \) receives a \text{prepare} message \( \mu \) from the primary of the initiator platform, it first validates the message. If node \( r \) is currently waiting for a \text{commit} message of some cross-platform transaction \( m' \) where the involved platforms of the two requests \( m \) and \( m' \) intersect, the node does not process the new transaction \( m \) before the earlier transaction \( m' \) gets committed. This ensures that requests are committed in the same order on different platforms. Otherwise, it assigns sequence number \( h_j \) to the message and multicasts a \text{signed propose} message \( \langle \langle \text{PROPOSE, } h_j, d \rangle_{\pi(p_j)}, \mu \rangle \) to the nodes of its platform. The primary node \( \pi(p_j) \) also piggybacks the \text{prepare} message \( \mu \) to its \text{propose} message to enable the node to access the request and validate the \text{propose} message. The primary node \( \pi(p_j) \), as presented in line 13, multicasts a signed \text{accept} message \( \langle \text{ACCEPT, } h_i, h_j, d \rangle_{\pi(p_j)} \) to every node of all involved platforms.

Upon receiving a \text{propose} message Once a node \( r \) of an involved platform \( p_i \) receives a \text{propose} message, as indicated in lines 8-10, it validates the signature and message digest (if the node belongs to the initiator platform \( i = j \), it also checks \( h_i \) to be valid (within a certain range)) since a malicious primary might multicast a request with an invalid sequence number. In addition, if the node is currently involved in an
uncommitted cross-platform request $m'$ where the involved platforms of two requests $m$ and $m'$ overlap in some other platform, the node does not process the new request $m$ before the earlier request $m'$ is processed. This is needed to ensure requests are committed in the same order on different platforms. The node then multicasts a signed accept message including the corresponding sequence number $h_i$ (that represents the order of $m$ in platform $p_j$), and the digest $d = D(m)$ to every node of all involved platforms.

As presented in lines 18-19, each node waits for valid matching accept messages from a local majority (i.e., either $f + 1$ or $2f + 1$ depending on the failure model) of every involved platform with $h_i$ and $d$ that matches the propose message which was sent by primary $p_i$. We define the predicate $accepted-local_{p_j}(m, h_i, h_j, r)$ to be true if and only if node $r$ has received the request $m$, a propose for $m$ with sequence number $h_i$ from the initiator platform $p_i$ and accept messages from a local majority of an involved platform $p_j$ that match the propose message. The predicate $accepted(m, h, r)$ where $h = [h_i, h_j, ..., h_k]$ is then defined to be true on node $r$ if and only if $accepted-local_{p_j}$ is true for every involved platform $p_j$ in cross-platform request $m$. The order of sequence numbers in the predicate is an ascending order determined by their platform ids. The propose and accept phases of the algorithm basically guarantee that non-faulty nodes agree on a total order for the transactions. When $accepted(m, h, v, r)$ becomes true, node $r$ multicasts a signed commit message $\langle commit, h, d, r\rangle_{\sigma_r}$, to all nodes of every involved platforms.

Finally, as shown in lines 20-21, node $r$ waits for valid matching commit messages from a local majority of every involved platform that matches its commit message. The predicate $committed-local_{p_j}(m, h, r)$ is defined to be true on node $r$ if and only if $accepted(m, h, r)$ is true and node $r$ has accepted valid matching commit messages from a local majority of platform $p_j$ that match the propose message for cross-platform transaction $m$. The predicate $committed(m, h, v, r)$ is then defined to be true on node $r$ if and only if $committed-local_{p_j}$ is true for every involved platform $p_j$ in cross-platform transaction $m$. The committed predicate indeed shows that at least $f + 1$ nodes of each involved platform have multicast valid commit messages. When the committed predicate becomes true, the node considers the transaction as committed. If all transactions with lower sequence numbers than $h_i$ have already been committed, the node appends a transaction block including the transaction as well as the corresponding commit message to its copy of the ledger.

Figure 9 shows the normal case operation for SEPAR to execute two concurrent cross-platform transactions in the presence of (a) crash-only and (b) Byzantine nodes where each transaction accesses two disjoint platforms. The network consists of four platforms where each platform includes either three or four nodes ($f = 1$).

In addition to the normal case operation, SEPAR has to deal with two other scenarios. First, when the primary node fails. Second, when nodes have not received a quorum of matching accept messages from the local-majority of every involved platform due to conflicting accept messages. Indeed, the primary nodes of different platforms might multicast their propose messages in parallel, hence, different overlapping platforms might receive the messages in different order. Furthermore, nodes might assign inconsistent sequence numbers since they have not necessarily received the latest propose message from the primary of their own platform. We use the techniques presented in SharPer [3] to address these two situations.
Algorithm 2 Global Consensus

1: init():
2: \( r := \text{node}_i \text{d}_i \)
3: \( p_j := \text{the platform that initiates the consensus} \)
4: \( \pi(p) := \text{the primary node of cluster } p \)

5: upon receiving valid transaction \( m \) and \( (r == \pi(p)) \)
6: multicast \( \langle \langle \text{PREPARE}, h_i, d, \pi(r), m \rangle \rangle \) to the primary node of every cluster
7: multicast \( \langle \langle \text{PROPOSE}, h_i, d, \pi(r), m \rangle \rangle \) to all nodes of \( p_i \)

8: upon receiving valid \( m = \langle \langle \text{PREPARE}, h_i, d, \pi(r), m \rangle \rangle \) and \( r == \pi(p) \)
9: if \( r \) is not involved in any uncommitted request \( m' \) where \( m \) and \( m' \) intersect in some other platform \( p_k \)
10: multicast \( \langle \langle \text{PROPOSE}, h_j, d, r, \pi(p_j), \mu \rangle \rangle \) to all nodes of \( p_j \)
11: multicast \( \langle \langle \text{ACCEPT}, h_i, h_j, d, r, \pi(p_j), \mu \rangle \rangle \) to all nodes

12: upon receiving valid \( \langle \langle \text{PROPOSE}, h_i, d, \pi(r), m \rangle \rangle \) and \( r \in \pi(p_i) \)
13: multicast \( \langle \langle \text{ACCEPT}, h_i, d, r, \pi(r) \rangle \rangle \) to all nodes

14: upon receiving valid \( \langle \langle \text{PROPOSE}, h_j, d, r, \pi(p_j), \mu \rangle \rangle \) and \( r \in \pi(p_j) \)
15: multicast \( \langle \langle \text{ACCEPT}, h_i, h_j, d, r, \pi(p_j), \mu \rangle \rangle \) to all nodes

16: upon receiving valid matching \( \langle \langle \text{ACCEPT}, h_i, h_j, d, r, \pi(p_j), \mu \rangle \rangle \) from local-majority of two-thirds of platforms
17: multicast \( \langle \langle \text{COMMIT}, h_i, h_j, d, r, \pi(p_j), \mu \rangle \rangle \) to all nodes

18: upon receiving valid \( \langle \langle \text{COMMIT}, h_i, h_j, d, r, \pi(p_j), \mu \rangle \rangle \) from local-majority of two-thirds of platforms
19: append the transaction block to the ledger

6.2.3 Global Consensus

The verification transactions include group signatures and all tokens that are consumed by different participants to perform a particular task. In SEPAR and in order to enable all platforms to check constraints, verification transactions are appended to the blockchains of all platforms. To do so, a Byzantine fault-tolerant protocol is run among all nodes of every platform where the protocol needs agreement from the local majority of the nodes of two-thirds of the platforms. The local majority, similar to cross-platform consensus, is defined based on the utilized consensus protocol within each platform. However, there are two main differences between cross-platform consensus and global consensus. First, in cross-platform consensus only the involved platforms participate, whereas, in global consensus, every platform verifies transactions by checking the group signatures and consumed tokens. Second, cross-platform consensus requires agreement from every platform, whereas, in global consensus, agreement from only two-thirds of platforms is needed. In fact, in cross-platform consensus, there might be some dependency between cross-platform transactions and internal ones, thus, to ensure data consistency, every involved platform must agree on the order of the cross-platform transaction. However, in global consensus, the goal is to verify the correctness of the transaction and as soon as two-thirds of platforms verify that (assuming at most one-third of platforms might behave maliciously), the transaction can be appended to the blockchain ledger.

Algorithm 2 shows the normal case of global consensus in SEPAR where a Byzantine protocol is run among all nodes of every platform (in contrast to cross-platform consensus where only the involved platforms participate). The protocol, similar to cross-platform consensus, process a transaction in four phases of prepare (lines 5-6), propose (lines 7-10), accept (lines 11-15), and commit (lines 16-19), however, each node waits for matching accept and commit messages from the local majority of only two-thirds of the platforms (as shown in lines 16 and 18).

Figure 7 presents the normal case operation of global consensus in SEPAR. Here all platforms include crash-only nodes where \( f = 1 \) and the network consists of four platforms.

Similar to cross-consensus, global consensus also addresses primary failure and conflicting transactions in a similar way as SharPer [3].

6.2.4 Correctness Arguments

A consensus protocol has to satisfy four main properties [9]: (1) agreement: every correct node must agree on the same value (Lemma 3), (2) Validity (integrity): if a correct node commits a value, then the value must have been proposed by some correct node (Lemma 4), (3) Consistency (total order): all correct nodes commit the same value in the same order (Lemma 5), and (4) termination: eventually every node commits some value (Lemma 6). The first three properties are known as safety and the termination property is known.
as liveness. In an asynchronous system, where nodes can fail, as shown by Fischer et al. [18], consensus has no solution that is both safe and live. Therefore, SEPAR guarantees safety in an asynchronous network, however, similar to most fault-tolerant protocols, deals with termination (liveness) only during periods of synchrony using timers.

**Lemma 3 (Agreement)** If node $r$ commits request $m$ with sequence number $h$, no other correct node commits request $m'$ ($m \neq m'$) with the same sequence number $h$.

The propose and accept phases of both cross-platform and global consensus protocols guarantee that correct nodes agree on a total order of requests. Indeed, if the accepted($m, h, r$) predicate where $h = [h_i, h_j, ..., h_k]$ is true, then accepted($m', h, q$) is false for any non-faulty node $q$ (including $r = q$) and any $m'$ such that $m \neq m'$. This is true because $(m, h, r)$ implies that accepted-local$_{p_i}(m, h_i, h_j, r)$ is true for each involved platform $p_i$ and a local majority ($f + 1$ crash-only or $2f + 1$ Byzantine node) of platform $p_j$ have sent accept (or propose) messages for request $m$ with sequence number $h_j$. As a result, for accepted($m', h, q$) to be true, at least one non-faulty nodes needs to have sent two conflicting accept messages with the same sequence number but different message digest. This condition guarantees that first, a malicious primary cannot violate the safety and second, at most one of the concurrent conflicting transactions, i.e., transactions that overlap in at least one platform, can collect the required number of messages from each overlapping platform.

**Lemma 4 (Validity)** If a correct node $r$ commits $m$, then $m$ must have been proposed by some correct node $\pi$.

In the presence of crash-only nodes, validity is ensured since crash-only nodes do not send fictitious messages. In the presence of Byzantine nodes, however, validity is guaranteed mainly based on standard cryptographic assumptions about collision-resistant hashes, encryption, and signatures which the adversary cannot subvert them. Since the request as well as all messages are signed and either the request or its digest is included in each message (to prevent changes and alterations to any part of the message), and in each step $2f + 1$ matching messages (from each Byzantine platform) are required, if a request is committed, the same request must have been proposed earlier.

**Lemma 5 (Consistency)** Let $P_{\mu}$ denote the set of involved platforms for a request $\mu$. For any two committed requests $m$ and $m'$ and any two nodes $r_1$ and $r_2$ such that $r_1 \in p_i$, $r_2 \in p_j$, and $\{p_i, p_j\} \in P_m \cap P_{m'}$, if $m$ is committed before $m'$ in $r_1$, then $m$ is committed before $m'$ in $r_2$.

As mentioned in both cross-platform and global consensuses, once a node $r_1$ of some platform $p_i$ receives a propose message for some transaction $m$, if the node is involved in some other uncommitted transaction $m'$ where $m$ and $m'$ overlap, node $r_1$ does not send an accept message for transaction $m$ before $m'$ gets committed. In this way, since committing request $m$ requires accept messages from a local majority of every (involved) platform, $m$ cannot be committed until $m'$ is committed. As a result the order of committing messages is the same in all involved platforms. It should be noted that in such a case we might face a deadlock where different platforms might need to re-initiate their transactions after some predefined time. To prevent any further deadlock, SEPAR define different waiting times for different platforms.

**Lemma 6 (Termination)** A request $m$ issued by a correct client eventually completes.
SEPAR deals with termination (liveness) only during periods of synchrony using timers. To do so, three scenarios need to be addressed. If the primary is non-faulty and accept messages are non-conflicting, following the normal case operation of the protocol, request \( m \) completes. If the primary is non-faulty, but accept messages are conflicting, the request will be re-initiated. Finally, SEPAR includes a routine to handle primary failures. SEPAR, as explained before, handles conflicting messages and primary failures in a similar way as SharPer [3].

7 Experimental Evaluations

In this section, we conduct several experiments to evaluate SEPAR. We have implemented a blockchain-based multi-platform crowdworking system. For the purpose of this evaluation, and as explained earlier, we do not focus on the description of tasks and contributions (both are modeled as arbitrary bitstrings). In addition, certificate tokens, as explained earlier, are very similar to \( \left( \left( w, p, r, \theta \right) \right) \) tokens except for the private part that has no significant impact on the performance and the number of interaction phases which is even less than constraint tokens. Therefore, we only focus on constraint tokens in the experiments. To implement group signatures, as discussed in Section 2, we use the protocol proposed in [11]. The experiments were conducted on the Amazon EC2 platform. Each VM is c4.2xlarge instance with 8 vCPUs and 15GB RAM, Intel Xeon E5-2666 v3 processor clocked at 3.50 GHz. When reporting throughput measurements, we use an increasing number of tasks submitted by requesters running on a single VM, until the end-to-end throughput is saturated, and state the throughput and latency just below saturation.

7.1 Token Generation

In the first set of experiments, we measure the performance of token generation in SEPAR for different types of constraints. We consider constraints with a single specified participant (e.g., \( \left( \left( w, *, *, \theta \right) \right) \)), two specified participants (e.g., \( \left( \left( *, p, r, \theta \right) \right) \)), and three specified participants (e.g., \( \left( \left( w, p, r, \theta \right) \right) \)). As shown in Figure 8(a), SEPAR is able to generate tokens in linear time. SEPAR generates each token in 0.7ms, hence, generating 1 million tokens in 12 minutes. This is an acceptable amount of time since token generation is executed periodically, e.g., every week or every month. Note that since tokens of different constraints can be generated in parallel, SEPAR can easily parallelize the token generation routine in order to improve the throughput. As can be seen in Figure 8(b), the type of constraints, i.e., the number of specified participants, also does not affect the performance and the token generation throughput and latency is constant in terms of the number of participant. However, it should be noted that a more complicated constraint, i.e., a constraint with more specified participants, requires more tokens to be generated.

7.2 Impact of Cross-Platform Tasks

In the second set of experiments, we measure the performance of SEPAR for workloads with different percentages of cross-platform tasks. We consider four different workloads with (1) no cross-platform tasks, (2) 20% cross-platform tasks, (3) 80% cross-platform tasks, and (4) 100% cross-platform tasks. We also assume that two (randomly chosen) platforms are involved in each cross-platform tasks and completion of each task requires a contribution coming from a randomly chosen worker. The system includes four platforms and each task has to satisfy two randomly chosen constraints. We consider two different networks with crash-only and Byzantine nodes. When all nodes follow crash-only nodes, as presented in Figure 9(a), SEPAR is able to process 8600 tasks with 400 ms latency before the end-to-end throughput is saturated (the penultimate point), if all tasks are local. Note that even when all tasks are local, the verification transaction of
In the next set of experiments, we measure the performance of SEPAR with different types of constraints. We consider four different scenarios where each task has to satisfy (1) no constraints (i.e., basic scenario), (2) a one-specified constraint, (3) a two-specified constraint, and (4) a three-specified constraint. The system consists of four platforms and the workload includes 90% intra- and 10% cross-platform tasks (the typical settings in partitioned databases [34, 35]) where two (randomly chosen) platforms are involved in each cross-platform tasks. As before, completion of each task requires a single contribution. To measure the overhead of group signatures and tokens, we compare the results with the basic scenario where there is no constraints in the system, thus, there is no need to exchange and validate tokens and signatures. When nodes follow the crash failure model and the system has no constraints, as can be seen in Figure 10(a), SEPAR is able to process 7000 tasks with 390 ms latency before the end-to-end throughput is saturated (the penultimate point). Adding constraints to the tasks results in more phases of communication between different participants to exchange tokens and signatures, however, SEPAR is still able to process 6200 tasks (the penultimate point) with 450 ms latency (only 11% and 15% overhead in terms of the throughput and latency respectively). The number of participants in each constraint, on the other hand, does not significantly affect the performance of SEPAR. This is expected, because more participants results in only increasing the number of (parallel) tokens and signature exchanges and the consensus protocols and other communication phases are not affected. Similarly, in the presence of Byzantine nodes and as shown in Figure 10(b), SEPAR is able to process 6140 tasks with 409 ms latency with no constraints and 5331 task (13% overhead) with
467 ms (14% overhead) latency with one-specified constraint. As before, the number of participants does not significantly affect the performance.

It should be noted that increasing the number of constraints, SEPAR still demonstrates similar performance as shown in this experiment (increasing the number of participants in each constraints). Indeed, adding more constraints results in adding more tokens and possibly more participants and signatures, however, it does not affect the consensus protocols and other communication phases.

7.4 Varying the Number of Platforms

In the last set of experiments, we measure the scalability of SEPAR in crowdsourcing systems with different number of platforms. We measure the performance of SEPAR in networks including 1 to 5 platforms for both crash-only and Byzantine nodes (assuming $f = 1$ in each platform). Each task has to satisfy on average two randomly chosen constraints, two (randomly chosen) platforms are involved in each cross-platform tasks, completion of each task requires a single contribution, and the workloads include 90% intra- and 10% cross-platform tasks. Note that in the scenario with a single platform, all tasks are intra-platform. As shown in Figure 11(a), in the presence of crash-only nodes, the performance of the system improves by adding more platforms, e.g., with five platform, SEPAR processes 6600 tasks with 400 ms latency whereas in a single platform setting, SEPAR processes 3300 task with the same latency. While adding more platforms improves the performance of SEPAR, the relation between the increased number of platforms and the improved throughput is non-linear (the number of platforms has been increased 5 times while the throughput doubled). This is expected because adding more platforms while increases the possibility of parallel processing of local tasks, makes the global consensus algorithm (which is needed for every single task) more expensive. In the presence of Byzantine nodes, SEPAR demonstrates similar behavior, e.g., processes 5500 tasks with 470 ms latency with 5 platforms.

8 Conclusion

In this paper, we introduce SEPAR, a multi-platform crowdworking system that enforces global regulations in a privacy-preserving and transparent manner. SEPAR consists of two main components. First, a token-based system that enables official institutions to express legal regulations in simple and unambiguous terms, guarantees the satisfaction of global constraints by construction, and allows participants to prove to external entities their involvement in crowdworking tasks, all in a privacy-preserving manner. Second, a permissioned blockchain that provides transparency using distributed ledgers shared across multiple platforms and enables collaboration among platforms through a suite of distributed consensus protocols. To the best of our knowledge, SEPAR is the first to address the problem of enforcing global regulation over multi-crowdworking platforms. We prove the privacy requirements of the token-based system as well as the correctness of the consensus protocols and conduct an extensive experimental evaluation to measure the performance and scalability of SEPAR.

References

[1] Private data collections: A high-level overview. https://www.hyperledger.org/blog/2018/10/23/private-data-collections-a-high-level-overview.
[2] M. J. Amiri, D. Agrawal, and A. El Abbadi. Caper: a cross-application permissioned blockchain. *Proceedings of the VLDB Endowment*, 12(11):1385–1398, 2019.

[3] M. J. Amiri, D. Agrawal, and A. El Abbadi. Sharper: Sharding permissioned blockchains over network clusters. *arXiv preprint arXiv:1910.00765*, 2019.

[4] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, et al. Hyperledger fabric: a distributed operating system for permissioned blockchains. In *EuroSys Conference*, page 30. ACM, 2018.

[5] G. Ateniese, J. Camenisch, M. Joye, and G. Tsudik. A practical and provably secure coalition-resistant group signature scheme. In *Annual International Cryptology Conference*, pages 255–270. Springer, 2000.

[6] Y. Aumann and Y. Lindell. Security against covert adversaries: Efficient protocols for realistic adversaries. In *Theory of Cryptography Conference*, pages 137–156. Springer, 2007.

[7] A. Azaria, A. Ekblaw, T. Vieira, and A. Lippman. Medrec: Using blockchain for medical data access and permission management. In *International Conference on Open and Big Data (OBD)*, pages 25–30. IEEE, 2016.

[8] J. Berg, M. Furrer, E. Harmon, U. Rani, and M. S. Silberman. Digital labour platforms and the future of work : Towards decent work in the online world. Technical Report ISBN 978-92-2-031025-0, International Labour Organization, 2018.

[9] C. Cachin, R. Guerraoui, and L. Rodrigues. *Introduction to reliable and secure distributed programming*. Springer Science & Business Media, 2011.

[10] C. Cachin and M. Vukolić. Blockchain consensus protocols in the wild. In *DISC*, pages 1–16, 2017.

[11] J. Camenisch and J. Groth. Group signatures: Better efficiency and new theoretical aspects. In *International Conference on Security in Communication Networks*, pages 120–133. Springer, 2004.

[12] M. Castro and B. Liskov. Practical byzantine fault tolerance and proactive recovery. *ACM Transactions on Computer Systems (TOCS)*, 20(4):398–461, 2002.

[13] M. Castro, B. Liskov, et al. Practical byzantine fault tolerance. In *OSDI*, volume 99, pages 173–186, 1999.

[14] E. Cecchetti, F. Zhang, Y. Ji, A. Kosba, A. Juels, and E. Shi. Solidus: Confidential distributed ledger transactions via pvorm. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 701–717, 2017.

[15] J. M. Chase. Quorum white paper, 2016.

[16] D. Chaum and E. Van Heyst. Group signatures. In *Workshop on the Theory and Application of Cryptographic Techniques*, pages 257–265. Springer, 1991.

[17] J. E. Cohen. Law for the platform economy. *UCDL Rev.*, 51:133, 2017.

[18] M. J. Fischer, N. A. Lynch, and M. S. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM (JACM)*, 32(2):374–382, 1985.

[19] D. Hopwood, S. Bowe, T. Hornby, and N. Wilcox. Zcash protocol specification. *GitHub: San Francisco, CA, USA*, 2016.

[20] M. Kenney and J. Zysman. The rise of the platform economy. *Issues in science and technology*, 32(3):61, 2016.

[21] K. Korpela, J. Hallikas, and T. Dahlberg. Digital supply chain transformation toward blockchain integration. In *proceedings of the 50th Hawaii international conference on system sciences*, 2017.

[22] A. Kosba, A. Miller, E. Shi, Z. Wen, and C. Papamanthou. Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In *2016 IEEE symposium on security and privacy (SP)*, pages 839–858. IEEE, 2016.

[23] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, 1978.

[24] L. Lamport et al. Paxos made simple. *ACM Sigact News*, 32(4):18–25, 2001.

[25] A. Liu, Z.-X. Li, G.-F. Liu, K. Zheng, M. Zhang, Q. Li, and X. Zhang. Privacy-preserving task assignment in spatial crowdsourcing. 2017.

[26] A. Liu, W. Wang, S. Shang, Q. Li, and X. Zhang. Efficient task assignment in spatial crowdsourcing with worker and task privacy protection. *GeoInformatica*, 22(2):335–362, 2018.
[27] B. Liu, L. Chen, X. Zhu, Y. Zhang, C. Zhang, and W. Qiu. Protecting location privacy in spatial crowdsourcing using encrypted data. *Advances in Database Technology-EDBT*, 2017.

[28] Y. Lu, Q. Tang, and G. Wang. Zebralancer: Private and anonymous crowdsourcing system atop open blockchain. In *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, pages 853–865. IEEE, 2018.

[29] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2008.

[30] C. on International Labour Legislation. Constitution of the international labour organization, 1919.

[31] G. C. on the Future of Work. Work for a brighter future. Technical Report ISBN 978-92-2-132796-7, International Labour Organization, 2019.

[32] F. participants. Imagine all the people and ai in the future of work. ACM SIGMOD blog post, September 2019.

[33] D. C. Sánchez. Raziel: Private and verifiable smart contracts on blockchains. *arXiv preprint arXiv:1807.09484*, 2018.

[34] R. Taft, E. Mansour, M. Serafini, J. Duggan, A. J. Elmore, A. Aboulnaga, A. Pavlo, and M. Stonebraker. E-store: Fine-grained elastic partitioning for distributed transaction processing systems. *VLDB*, 8(3):245–256, 2014.

[35] A. Thomson, T. Diamond, S.-C. Weng, K. Ren, P. Shao, and D. J. Abadi. Calvin: fast distributed transactions for partitioned database systems. In *SIGMOD International Conference on Management of Data*, pages 1–12. ACM, 2012.

[36] H. To, G. Ghinita, L. Fan, and C. Shahabi. Differentially private location protection for worker datasets in spatial crowdsourcing. *IEEE Transactions on Mobile Computing*, 16(4):934–949, 2016.

[37] H. To, C. Shahabi, and L. Xiong. Privacy-preserving online task assignment in spatial crowdsourcing with untrusted server. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*, pages 833–844. IEEE, 2018.

[38] S. Zhu, Z. Cai, H. Hu, Y. Li, and W. Li. zkcrowd: a hybrid blockchain-based crowdsourcing platform. *IEEE Transactions on Industrial Informatics*, 2019.