Received: 24 September 2023 | Revised: 16 July 2024 | Accepted: 24 July 2024

DOI: 10.1002/spy2.451

RESEARCH ARTICLE

WILEY

Secure semi-automated GDPR compliance service with restrictive fine-grained access control

Max Hashem Eiza[1] | **Vinh Thong Ta**[2] | **Qi Shi**[1] | **Yue Cao**[3]

[1]School of Computer Science and Mathematics, Liverpool John Moores University, Liverpool, UK

[2]Department of Computer Science, Edge Hill University, Ormskirk, UK

[3]School of Cyber Science and Engineering, Wuhan University, Wuhan, China

Correspondence
Max Hashem Eiza, School of Computer Science and Mathematics, Liverpool John Moores University, Liverpool, UK.
Email: M.HashemEiza@ljmu.ac.uk

Abstract

Sharing personal data with service providers is a contentious issue that led to the birth of data regulations such as the EU General Data Protection Regulation (GDPR) and similar laws in the US. Complying with these regulations is a must for service providers. For users, this compliance assures them that their data is handled the way the service provider says it will be via their privacy policy. Auditing service providers' compliance is usually carried out by specific authorities when there is a need to do so (e.g., data breach). Nonetheless, these irregular compliance checks could lead to non-compliant actions being undetected for long periods. Users need an improved way to make sure their data is managed properly, giving them the ability to control and enforce detailed, restricted access to their data, in line with the policies set by the service provider. This work addresses these issues by providing a secure semi-automated GDPR compliance service for both users and service providers using smart contracts and attribute-based encryption with accountability. Privacy policies will be automatically checked for compliance before a service commences. Users can then upload their personal data with restrictive access controls extracted from the approved privacy policy. Operations' logs on the personal data during its full lifecycle will be immutably recorded and regularly checked for compliance to ensure the privacy policy is adhered to at all times. Evaluation results, using a real-world organization policy and example logs, show that the proposed service achieves these goals with low time overhead and high throughput.

KEYWORDS

access control, blockchain, compliance, GDPR, personal data

1 | INTRODUCTION

In our fast-paced digital world, people share their personal data with service providers to access various services. However, many services involve a complex network of multiple providers, meaning that users' data is often shared with third parties *TPs*. The data access/sharing is governed under specific regulations such as the EU General Data Protection Regulation (GDPR) and similar regulations in the US (e.g., Health Insurance Portability and Accountability Act (HIPAA)).

The focus of this work is the GDPR, which for service providers, compliance with it is not optional as any breach would result in hefty fines.

Through its 99 articles, the GDPR sets out the rights of data subjects (*DS*) (i.e., data owner) and the responsibilities of data controllers/processors who store, process or transfer personal data. It also ensures that data are collected legally for legitimate purposes. Given its textual/abstract nature, the GDPR rules are open to interpretation in terms of how organizations satisfy them and/or demonstrate compliance. Moreover, GDPR compliance checks are only carried out when there is a legal request from the GDPR supervisory authority.

Service providers develop their privacy policies to adhere to GDPR while collecting/storing/processing personal data. Under the accountability principle in Article (5)2 of the GDPR, they should also be able to withstand any auditing process by demonstrating compliance with these regulations. Since these auditing processes are irregular, two main questions arise here. First, how can a service provider (*SP*) be trusted to demonstrate compliance via a centralized architecture that is controlled solely by the *SP*? Second, how can users verify that the *SP* is complying with the regulations by observing their privacy policies before/after collecting the data? Moreover, the lack of transparency of *SP′s* operations on the data adds more challenges to the mix.

Hence, there is a need for a solution that can achieve the following objectives: (1) ensure, and automatically verify, conformance between a *SP′s* privacy policy and GDPR rules, (2) empower *DSs* to grant fine-grained access to their data that can be time limited and restrictive to those who are identified by the *SP′s* privacy policy, (3) carry out regular compliance checks on the actual actions carried out on the data during its full lifecycle, and (4) address the lack of trust among auditors, *SPs*, *TPs* and *DSs* when carrying out an audit to provide irrefutable evidence to prove any violation.

This article presents the desired solution as a secure semi-automated compliance and auditing service to ensure GDPR compliance during the data lifecycle using smart contracts *SCs* and ciphertext-policy attribute-based encryption (CP-ABE) with accountability. To address the absence of trust among parties, a blockchain-based architecture is developed. The proposed service semi-automatically ensures and enforces compliance against a set of GDPR rules before/after data collection in two stages. The first stage happens before data collection commences when the *SP′s* privacy policy is checked against GDPR rules automatically using a policy engine. If it is a pass, the data collection starts where *DSs* use CP-ABE to embed fine-grained access to their data based on the verified privacy policy. Finally, regular compliance checks during the data lifecycle, against the *SP′s* privacy policy, are carried using *SCs* functions based on the data operations' logs, which will be stored immutably on the chain. Furthermore, privacy policies are also stored on-chain. The proposed architecture decouples data storage from the blockchain network (i.e., the data itself is stored off the chain) to reduce the storage and computation overhead.

The main contribution of the article is as follows:

1. A novel consortium blockchain network (called *CBN* from now on) architecture for transparent and reliable auditing compliance to address the lack of trust among auditors, *SPs*, *TPs* and *DSs*. Fine-grained access control policies are enforced via implemented *SC* functions to control access to the data operations' logs and carry out regular compliance checks.
2. A bespoke on/off-chain communications protocol among *DSs*, *SP*, *TP*, and the off-chain data storage server *DSt* with close observation of the GDPR Auditor *GA*. The protocol allows *DSs* to place restrictive fine-grained controls on any data field before sharing thanks to ABE.
3. An industry standard *CBN* based on Hyperledger Fabric (HLF) platform that implements a real-world privacy policy from the energy sector to show that our solution achieves a feasible implementation, practical efficiency, and auditing accuracy.

The article is structured as follows: Section 2 discusses the related works in the literature. Section 3 presents the preliminaries for this article. The system model and specifications of the auditing service is detailed in Section 4. Section 5 presents the semi-automated auditing service design and algorithms while the full system deployment and operations are provided in Section 6. Section 7 evaluates the developed auditing service via a smart meter case study. Finally, Section 8 concludes the article and discusses future directions.

2 | RELATED WORKS

In terms of personal data management, and sharing data on cloud storage servers, works such as References 1–5 focused on secure sharing and storage of electronic medical records (EMR) and personal data using techniques such as

certificateless signature, key escrow-free attribute based encryption, and blockchain. These techniques were used to solve the issues of untrusted cloud servers[4] and collusion attacks between malicious users and revoked users.[5] For instance, Sun et al. proposed a blockchain-based framework for EMRs with fine-grained access control that combines a distributed storage system Inter-Planetary File System (IPFS), Ethereum, and CP-ABE.[1] On the other hand, Yan et al. proposed a novel remote data possession checking (RDPC) protocol based on homomorphic hash function.[3] The proposed protocol aimed to give data owners dependable means to check the possession of their files, which are outsourced to remote cloud servers. The authors also introduced an operation record table (ORT) to track operations on file blocks such as block modification, block insertion and block deletion. The scheme was proven secure against forgery attacks, replay attacks and replace attacks based on a typical security model.

Guo et al. developed a revocable blockchain-aided attribute-based encryption with escrow-free (BC-ABE-EF) scheme.[2] The scheme utilized a distributed consortium blockchain that contains consensus nodes to replace the key authority and solve the key escrow problem. The distributed consortium blockchain is used to collaborate with users to implement the key generation function via a secure two-party computation protocol. Moreover, to resist collusion attacks between malicious users and revoked users, each user's certificate is calculated by a group manager (GM) and embedded in the user's key. The GM is one of five participants in the proposed data management system in addition to data owner, data user, blockchain, cloud server and decryption cloud server. The performance analysis showed promising results in terms of storage overhead and calculation cost in comparison to other schemes. However, the scheme is not implemented on an existing blockchain to examine how the throughput of the blockchain may affect the performance of the proposed scheme.

The authors in Reference 6 proposed PrivySharing, a blockchain-based framework for privacy-preserving and secure IoT data sharing. They adopted a permissioned blockchain to separate different users' data into different channels. Users can control who has access to their data using ACLs. The authors implemented the proposed framework with different data types such as health and surveillance data. Compliance with some GDPR requirements is illustrated via ACLs. Nonetheless, the data were stored on-chain where nodes have to maintain copies of multiple ledgers depending on how many channels they are member of. With the large amount of data from IoT devices, this becomes infeasible. Moreover, while compliance with some GDPR requirements is shown, it is not clear how these requirements are audited during the data lifecycle.

Amato et al. focused on law compliance of *SCs* in IoT environments in Reference 7. They proposed a formal model based on multiagent logic and ontological description of contracts for validating law compliance of *SCs*. They demonstrated the benefits of their methodology using a case study where a *SC* is used by insurance companies to collect data from drivers to adapt premium prices. The GDPR requirement, that biometric data must be stored encrypted, was used to demonstrate whether the *SCs* comply or not. The authors did not provide an implementation for their solution to show how efficient and practical it will be.

In Reference 8, Javed et al. presented PETchain, a blockchain-based privacy enhancing technology. PETchain leverages IPFS where users encrypt their data and upload it to the IPFS network. The authors also introduced the trusted execution environment (TEE) notion to allow *SPs* to process users' encrypted data without accessing the raw data. Users own and deploy PETchain *SC*, which contains the key to the stored data, the *SP′s* address, and the URL of the TEE executor. The authors deployed the *SC* on Ethereum and showed that PETchain achieved high levels of Transactions Per Second (TPS) by having low block time and a high gas limit. The evaluation focused on gas consumption and TPS rather than the system compliance with GDPR.

Wang et al.[9] proposed a secure and auditable private data sharing (SPDS) scheme for smart grid using blockchain for trust-free private data computation and data usage tracking. They used TEE off-chain to process confidential users' data. Moreover, they developed contracts based on the contract theory to encourage users to participate. The data access policy is based on ACLs while the limited memory capacity of the TEE executor causes a considerable performance degradation. The evaluation focused on the benefits payoff between the SPDS and linear contract and fixed price schemes. Hence, the performance of SPDS implementation is not investigated.

In terms of GDPR compliance using blockchain and *SCs*, Heiss et al.[10] developed a blockchain-based approach to detect violations of consent declarations in multi-service setups based on off-chain computed violation claims utilizing non-interactive zero-knowledge (NIZK) proofs. Their approach focuses on supporting *SPs* to fulfil the GDPR requirement of consent collection and management. It also helps to detect wrong access to personal data by *TPs′* APIs. However, the proposed approach needs further investigation in terms of its security and ability to detect GDPR violations.

In Reference 11, the authors proposed a lightweight blockchain-based GDPR-compliant personal data management platform. The platform provides mechanisms for *DSs* to control and manage their personal data as well as publicly

accessible and immutable evidences, which are useful for a *SP* to prove the agreements made with a *DS*. These pieces of evidence can be used by a supervisory authority for auditing to ensure the *SP* is complying with their agreements. In addition to blockchain and *SCs*, the authors utilized eXtensible Access Control Markup Language (XACML) to express access control on the *DS*′*s* data. The *SCs* were deployed on Ethereum to test their efficiency and cost to run (i.e., gas consumption). The evaluation focused on the number of interactions required within the system to perform different operations such as *get processing consent* and *access data*. However, samples of the tested policies, expressed in XACML, and the full data lifecycle operation and logs were not included. Finally, the authors stated that deploying the proposed system on an existing blockchain and studying all the incurred costs in a real setting were left for their future work.

Truong et al. proposed a GDPR-compliant personal data management platform that leverages blockchain and *SCs*.[12] The platform allows *DSs* to ensure that only designated parties can access and process their personal data via ACLs. All activities related to data access are logged into the blockchain to verify any violation of GDPR rules. The platform is developed using HLF where the data decoupling concept is followed. If access is granted, the platform issues a token to the *SP* that is validated by the resource server before returning the requested data. The data access policy is checked when consent is given for data access to the data processor or a *TP* rather than checking against the agreed policy with the data controller. Hence, policy checks are carried out when data access is requested rather than prior to any data collection. The platform does not differentiate between data collection consent and data processing consent nor offers regular checks against the agreed policy with the *SP*. Finally, the data stored on the resource server is not encrypted.

A lightweight blockchain-based GDPR compliant personal data management platform is proposed in Reference 13. The platform allows the data controller to specify the data collection policy including storage periods and purposes. If the *DS* accepts it, then consent is given to the data controller via a *SC*. When a *TP* requests access to the data, the data controller prepares a new purpose *SC* that is sent to the *DS* for approval before consent is given. The authors stated that the platform is implemented on a Ganache local blockchain to validate its feasibility.[14] However, the results of their tests especially regarding GDPR compliance were not provided.

2.1 | Comparison with this work

The main aspects in which this work differs from other works in the literature are accountability for *SPs*, fine-grained access control, semi-automated compliance and auditing, and a practical implementation.

Unlike the simple ACL based policy, our approach enforces compliance with a more fine-grained policy, covering five representative stages of the data lifecycle with a particular focus on the data controller (i.e., *SP*), and its privacy policy. Our approach will automatically enforce that data are always being collected, used, stored, deleted, and transferred in compliance with the agreed privacy policy of the *SP*. For example, while the approach proposed in Reference 12 only verifies if a *TP* has access right to certain data, our approach also verifies if the data controller is permitted to collect certain types of data for certain purposes. As part of the data collection sub-policy, our approach also verifies if consent has been collected by the *SP* before collecting certain types of data. In addition, it enforces data storage and deletion specified in the data storage and retention sub-policies. For example, suppose the *SP* has not yet fulfilled the retention delay policy for a certain type of data. In that case, data deletion will be enforced automatically at the end of the retention delay period set in the policy. With regards to data transfer, our approach enforces that only the set of *TPs*, which have been given the rights in the data transfer sub-policy, can access the data. Finally, one of the main strengths of our approach is the semi-automated compliance verification engine that is capable of proving that a (collection, usage, storage, deletion, transfer) request sent by the *SP*, *DS* or *TP* conforms with the "agreed" privacy policy of the *SP*. This way, the enforcement carried out by *SCs* will be based on mathematically proved compliance, reducing the chance of incorrect/false enforcement.

3 | PRELIMINARIES

3.1 | Privacy policy language

To verify data operations' logs against the privacy policy of the *SP* based on automated proofs, we propose a variant of a privacy policy language, which is an extended version of DataProVe.[15] A privacy policy is defined on a data type θ, and is composed of five sub-policies as follows:

$$\pi(\theta) = (\pi_{\text{col}}, \pi_{\text{use}}, \pi_{\text{str}}, \pi_{\text{del}}, \pi_{\text{acc}})$$

1. Data collection sub-policy $\pi_{\text{col}} = (ccons, cpurps)$, where $ccons \in \{Y, N\}$ specifies whether a consent is required to be collected from the *DS* or not for a data type θ, and *cpurp* is a set of collection purposes. A purpose has the form $SP : act : \theta$, which specifies that a piece of data of type θ is collected by the *SP* to perform an action *act*. These capture the consent and purposes limitation requirements of the GDPR.
2. Data usage sub-policy $\pi_{\text{use}} = (ucons, upurps)$, with a usage consent requirement, $ucons \in \{Y, N\}$, and *upurps*, a set of usage purposes. These capture Articles 6 and 30(1)(b) of the GDPR.
3. Data storage sub-policy $\pi_{\text{str}} = (scons, where)$, with a storage consent requirement, $scons \in \{Y, N\}$, and *where* is a set of places where a piece of data of type θ can be stored. These partially capture the storage limitation principle in Article 5(1)(e) of the GDPR.
4. Data deletion sub-policy $\pi_{\text{del}} = (fromwhere, delay)$. *fromwhere* is the location from where a piece of data of type θ can be deleted. This strongly depends on the storage locations, *where*, as defined in the storage sub-policy π_{str}. *delay* is the delay value for retention. This value can be a specific numerical time value (e.g., 1 day (1d), 5 years (5y)). These two elements partially capture Articles 5(1)(e) and 17(1)(a,e) of the GDPR.
5. We extend the data access sub-policy π_{acc} in DataProVe[15] to include the conditions and purposes of data access. $\pi_{\text{acc}} = \{entity1(args1), \ldots, entityn(args\_n)\}$ defines that *entity1*, … , *entityn* can access the data besides the arguments *args1*, … , *argsn*, respectively. Here $args = (acons, conditions, purps, transfer)$, which contains an access consent requirement $acons \in \{Y, N\}$, *conditions* that set out the conditions that must be met for accessing the data (e.g., a certain time interval), *purps* is a set of purposes for the data access, and $transfer \in \{Y, N\}$ indicates whether the data can be transferred to that entity as a result of this access or not. These partially capture the requirement of transferring data to a *TP* in Article 45(1) of the GDPR.

3.2 | Blockchain and smart contracts

Blockchain technology ensures data consistency among distributed nodes in a Peer-to-Peer network without a trusted *TP* by utilizing the Distributed Ledger Technology (DLT).[16] Changes are only committed to the ledger if nodes reach a consensus on the validity of these changes. Blockchains can be either (1) permissionless where virtually any node can anonymously participate in the network (e.g., Bitcoin) or (2) permissioned where participants' identities are verified before they can join the network (e.g., Hyperledger Fabric (HLF)[17] and R3 Corda[18]). Permissioned blockchains do not require mining activities thus, consensus can be reached quicker and with less computational power than in permissionless blockchains. Our system is based on the corporate blockchain concept, which is a sub-type of permissioned blockchains. A corporate blockchain is designed for use within a single organization or a consortium of trusted participants. The integrity of data stored on-chain can be accessed and verified by members of the consortium, thereby addressing the (lack of) trust issues among them. As discussed later in Section 4, we define the following participants: *SP*, *DSt*, *TPs*, and *GA* as part of the consortium where we utilize HLF as our *CBN*. The *SCs* are implemented using Node.js.[19]

To set up the *CBN*, each participant deploys their peer node *P*, which is responsible for transaction validation and storage. These transactions represent different events such as a data access request or processing event. Transactions are submitted by a peer node (e.g., the *SP*'s peer), which will get validated based on consensus rules and *SC* logic, which are written to enforce *SP*'*s* privacy policy. Validated transactions (i.e., data access requests and processing events) are then permanently added to the ledgers, creating an immutable and transparent record.

3.3 | Public key with keyword search (PEKS)

The public key encryption with a keyword search (PEKS) scheme allows searching over encrypted data where the data owner can authorize the search ability to a recipient *R* using a set of encrypted keywords associated with the encrypted data, without compromising its security.[20] The PEKS scheme can be generally described as follows:

- $Setup(\lambda) \rightarrow (PK_R, SK_R)$: The setup algorithm takes a security parameter λ and outputs a public key PK_R and a private key SK_R for the recipient *R*.
- $Enc(PK_R, KW) \rightarrow (S_{PEKS})$: The encryption algorithm takes PK_R and a set of keywords $KW = \{kw_1, kw_2, \ldots, kw_n\}$ and generates a searchable encryption S_{PEKS} of *KW*. The encryption algorithm is run by the data owner.

- $Trapdoor(SK_R, kw_i) \rightarrow (T_{kwi})$: To search for the encrypted data, R generates a trapdoor T_{kwi} that is associated with the keyword kw_i using its private key SK_R.
- $Test(S_{PEKS}, T_{kwi}) \rightarrow True/False$: Upon receiving T_{kwi}, the storage server performs a keyword search process and returns *True* if $kw_i \in KW$ or *False* otherwise. If the output is *True*, the ciphertext associated with the keyword kw_i is returned to R for decryption.

3.4 | Ciphertext-policy attribute-based encryption (CP-ABE) with accountability and proxy re-encryption (PRE)

CP-ABE allows the data owner to embed an access policy inside the ciphertext. Only those whose attributes match the specified access policy π_{acc} can decrypt and access the data.[21] Furthermore, to ensure user accountability and address user key abuse problem[22] (i.e., when a malicious user could simply reveal his/her key to unauthorized users to gain access to the encrypted data), user identity information ID is embedded into the user's secret key. It also include private information, which is unknown to the issuer entity (aka the authorization center), called key family number (KFN) to mitigate the problem of authorization center key abuse.

Complementing the CP-ABE, in the proxy re-encryption (PRE) scheme, an authorized entity, aka delegator, can delegate its search ability to other entities, aka delegatees, by re-encrypting the data without revealing its private key. Similar to CP-ABE, the delegator can ask the data storage to re-encrypt the data under a different access policy that is dedicated to the delegatee. Hence, the CP-ABE scheme with keyword search and PRE can be generally described as follows[22,23]:

1. $Setup(\lambda, \mathcal{U}) \rightarrow (PK_{ABE}, MK_{ABE})$: The setup algorithm takes a security parameter λ and a universe of attributes $\mathcal{U}$ and outputs a public key PK_{ABE} and a master private key MK_{ABE}.
2. $KeyGen(MK_{ABE}, AS, ID) \rightarrow (SK_{AS})$: The key generator algorithm takes MK_{ABE}, a valid attribute set AS that satisfies π_{acc}, and user's identity information ID to generate a private key SK_{AS}. The private key SK_{AS} embeds the user ID.
3. $Enc(data, \pi_{\text{acc}}, kw) \rightarrow (CT_{ABE})$: The encryption algorithm encrypts the data with the access policy π_{acc} and a keyword kw and generates a ciphertext CT_{ABE}.
4. $TokenGen(SK_{AS}, kw') \rightarrow (\tau_{kw'})$: The token generator algorithm takes the private key SK_{AS} and a keyword kw' and generates a search token $\tau_{kw'}$ for kw'.
5. $Test(CT_{ABE}, \tau_{kw'}) \rightarrow True/False$: Upon receiving $\tau_{kw'}$, the storage server performs a keyword search process on CT_{ABE} under kw and returns *True* if $kw' = kw$ or *False* otherwise. Note that only those with SK_{AS} that satisfies π_{acc} can then decrypt the data.
6. $Dec(CT_{ABE}, SK_{AS}, ID) \rightarrow (data)$: The decryption algorithm takes the ciphertext CT_{ABE}, the secret key SK_{AS}, which contains the attribute set AS, and ID. It outputs $data$ if and only if SK_{AS} satisfies π_{acc}.
7. $RKeyGen(SK_{AS}, \pi'_{\text{acc}}, kw', ID', KFN) \rightarrow (RK)$: The key re-generator algorithm takes the private key of the delegator SK_{AS}, a new access policy π'_{acc}, a new keyword kw', the delegatee's ID' and the KFN to generate a re-encryption key RK. Note that AS satisfies π_{acc} but not necessarily π'_{acc}. Moreover, if $kw \neq kw'$, it means the keyword in the ciphertext will be updated during the re-encryption phase.
8. $ReEnc(CT_{ABE}, RK) \rightarrow (CT_{PRE})$: The re-encryption algorithm encrypts the ciphertext CT_{ABE} with the re-encryption key RK and generates a ciphertext CT_{PRE} under a new access policy π'_{acc} and a new keyword kw'.
9. $Trace(SK_{suspected}) \rightarrow (ID, KFN)$: Assuming $SK_{suspected}$ is correctly formed, the algorithm checks if ID is equal to that of the user who owns $SK_{suspected}$. If it is not, the algorithm indicates that the user with ID is dishonest. If $KFN \neq \phi$, it checks whether the KFN is equal to that of the user who owns ID. If not, it indicates that the authority is dishonest.
10. Using *TokenGen* and *Test* defined above, the delegatee can decrypt CT_{PRE} if it satisfies π'_{acc}.

4 | SYSTEM MODEL AND SPECIFICATIONS

In the following, we explain the system components in Figure 1 and their roles in the proposed service.

- *CA* is a fully trusted entity that is responsible for generating certificates and initializing the system security parameters as to be explained later.

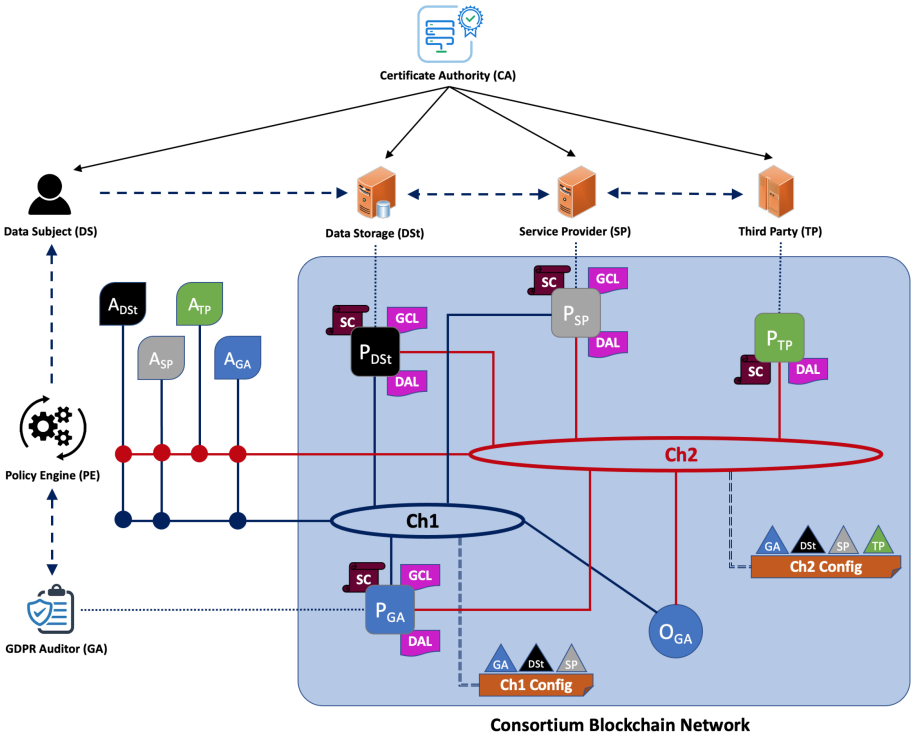

FIGURE 1 System model.

- *DS* refers to the owner of the personal data that will be collected by *SP* to access its services. The required personal data is determined by *SP*'*s* privacy policy.
- *GCL* refers to the GDPR compliance ledger where all the operations' logs, conducted by *DSt* and *SP*, on *DS*'*s* data during the data lifecycle are recorded. These records are used by the *GA* for auditing purposes.
- *DAL* is data access ledger where all the operations, conducted by *TP* on *DS*'*s* data are recorded. These records are also used by the *GA* for auditing purposes.
- P_{DSt}, P_{SP}, P_{TP} and P_{GA} are peer nodes that belong to *DSt*, *SP*, *TP* and *GA*, respectively. These nodes allow access to the ledgers on can commit changes to them if consensus is reached.
- *Ch*1 is a communication channel for nodes that belong to *GA*, *DSt* and *SP*. This way, these organizations can communicate and transact privately without the involvement of *TP*, which does not need access to the *GCL* on *Ch*1. Hence, the ledger records are only available for those who are admitted to *Ch*1 as determined by its configuration *Ch*1 *Config*. Any changes to the channel configuration (e.g., adding a new node) must be approved by these organizations before it is committed.
- *Ch*2 is a communication channel for nodes that belong to *GA*, *DSt*, *SP* and *TP*. This way, *TP* will only have access to the *DAL* on *Ch*2.
- *DSt* is a storage server where personal data are stored encrypted. It is part of the *CBN* and has access to *Ch*1 and *Ch*2 and their ledgers. It can be independent or managed by *SP*. If *SP* collects the *DS*'*s* personal data, this will be stored on *DSt* encrypted using PEKS while the encryption key is encrypted using CP-ABE. The *DSt* receives requests from *SP* and *TP* to transfer the required *DS*'*s* data. All the logs of these operations, which do not contain any identifying or personal data related to the *DS*, are recorded on the chain for auditing purposes. The format and contents of these logs are explained in Section 4.4.
- *SP* refers to the service provider who has a specific privacy policy π. It determines the purposes of collecting and processing *DS*'*s* data. It is also part of the *CBN* and has access to the ledgers on both *Ch*1 and *Ch*2. All the logs of requests/operations carried out by *SP* are recorded on the chain.

- *TP* is a third party that may act as a data processor on behalf of the *SP* or request access to *DS′s* data for certain purposes. *DS′s* data can be transferred to *TP* if π allows such operation. Since the *TP* has limited access to *DS′s* data, it has access only to *Ch*2 and its ledger, which records logs for access requests from *TP*.
- *PE* is the policy engine that implements the privacy policy language defined in Section 3.1 and runs the compliance checks automatically to ensure a privacy policy and/or operations comply with the GDPR.
- *GA* is the auditor that ensures the *SP* privacy policy π complies with GDPR, and all the operations during the data lifecycle still comply with π. *GA* does that first by passing π to the *PE*, which automatically proves if π fully complies with GDPR. Later on, *GA* periodically audits the current operations by checking the logs from the ledgers on *Ch*1 and *Ch*2 to *PE*. Note that *GA* manages the *CBN* via its orderer node O_{GA}.
- *SC* is the smart contract that implements functions to check the compliance of the data operations' logs against each of the five sub-policies in the privacy policy π. *SC* definition is committed and installed on all nodes. Note that *SC* functionality has nothing to do with invoking the actual operations on *DS′s* data. Moreover, its role is different from *PE* functionality, which checks the *SP′s* privacy policy compliance against GDPR before the system starts.
- O_{GA} is the orderer node that is responsible for ordering transactions, creating a new block of ordered transactions, and distributing a newly created block to all peers.
- A_{DSt}, A_{SP}, A_{TP} and A_{GA} are applications that allow *DSt*, *SP*, *TP* and *GA* to interact with *SC* and invoke its functions to update the ledger records, respectively. Note that A_{TP} does not have access to *SC* on *Ch*1.

4.1 | Security model and design goals

Although the focus of this work is GDPR compliance, the following security considerations are important for the integrity of the system model in Figure 1. The *CA* is fully trusted by all participants in the system. The *DSt* is honest but curious, which means it will perform all the operations as dictated by the requests it receives but curiously tries to infer additional information about the data available to it. The *SP* and *TP* are rational but might not handle *DS′s* personal data following all the aspects of the privacy policy π (e.g., a piece of data is collected without seeking consent first, data is not deleted when its purpose is no longer valid, etc.) The *GA* is semi-trusted by all the parties in the system to conduct GDPR compliance and auditing as long as it can provide irrefutable evidence of compliance or breach. As mentioned above, all communications and transactions are encrypted in Figure 1 using PKI. In addition, *DS′s* data confidentiality is guaranteed as it is stored encrypted using *PEKS* on *DSt*. None of *DS*'s data is stored on the *CBN*.

To achieve the desired outcomes, the system should achieve the following design goals:

1. *Semi-automated GDPR compliance checks* to alleviate the problem of irregular auditing and ensure continuous conformance between GDPR rules and privacy policies during the full data lifecycle.
2. *Transparency*. This refers to all the actions that are carried out by all parties on *DS′s* personal data from collection all the way up to deletion.
3. *Data owner enforced fine-grained access control*. The *DS* should be able to enforce fine-grained access control on his/her data according to the agreed privacy policy.
4. *Efficiency*. The time overhead of compliance checks and continuous auditing during the data lifecycle should be feasible and does not impair the system functionalities.
5. *Accountability and traceability*. These features are required to identify entities who accessed or delegated access to *DS′s* personal data. It will also allow to identify any entity who could leak and/or reveal their attribute keys deliberately to unauthorized entities to access the encrypted data. This addresses the entity key abuse problem in both CP-ABE and PRE schemes.

4.2 | Threat model

In Figure 1, any system entity can be considered as an adversary if it misbehaves or deviates from its legitimate operations. The *DSt* is honest in its operations to return search results on personal data but is curious to find any distinctive properties that can identify *DSs*. It is assumed that *DSt* might collude with revoked entities to help them gain unauthorized access to *DS′s* data. Other entities including *SP*, *TP* and *GA* are all semi-trusted in the sense that they always perform their

assigned operations correctly. However, they might collude directly or indirectly for illegal access operations. They may also misbehave independently (e.g., due to a security breach). Finally, the *CBN* is resistant to unauthorized access to application channels and their ledgers. Note that issues such as compromised Membership Service Provider, *CA* and ordering service are outside the scope of this work.

4.3 | Policy specification

Based on the policy language in Section 3.1, for automated processing and verification purposes, a policy $\pi(\theta)$ of a *SP* (identified with *SP_ID*) for a data type θ can be defined in JSON format as follows.

{
"***SP_ID***": "",
$\pi(\theta)$: {
"π_{col}": {"***ccons***": "*Y*"/"*N*",
"***cpurps***": ["*cpurp*1", ..., "*cpurpn*"]},
"π_{use}": {"***ucons***": "*Y*"/"*N*"
"***upurps***": ["*upurp*1", ..., "*upurpn*"]},
"π_{str}": {"***scons***": "*Y*"/"*N*"
"***spurps***": ["*spurp*1", ..., "*spurpn*"],
"***splaces***": ["*spl*1", ..., "*spln*"]},
"π_{del}": {"***delay***": "*delayval*",
"***dplaces***": ["*dpl*1", ..., "*dpln*"]},
"π_{acc}": {
"***entity***": {"*acons*": "*Y*"/"*N*"
"*conditions*": ["*cond*1", ..., "*condn*"],
"*purps*": ["*purp*1", ..., "*purpn*"],
"*transfer*": "*Y*"/"*N*" }}
}
}

SP should define the policy for each data type θ used by the service. A sub-policy can be empty if it does not apply to a given data type (e.g., π_{str} is empty for the data type that would not be stored). The specification of the policy can be done either manually or automatically. Proposing an automated policy specification approach is beyond the scope of this article. The purpose elements in $\pi(\theta)$ have the format of $who : [act] : datatype\_to$, which captures the purpose of collecting, using, storing, and accessing/transferring a piece of data of type θ is for an entity *who* to carry out the action *act* and get a piece of data of type *datatype_to* as a result. For example, for $\theta = energy$, a usage purpose can be $SP : [calculate] : bill$ where the energy consumption is used to calculate the bill.

This policy specification is relatively simple and focuses on the relevant aspects of the GDPR such as consent collection, deletion, storage and purpose limitation. However, there are other aspects of the GDPR including processing special category data, and conditional transfer of data outside the EU (considering contractual aspects) that are subject to be addressed in future extensions of the policy language. Given the complexity of the GDPR, to the best of our knowledge, no existing policy language addresses/covers all rules in it.

4.4 | Log events

A set of log events is defined on a given data type θ to capture the events linked to that data type in system operations' logs. A log of θ, denoted by $\mathcal{L}(\theta)$, is a set of log event sequences defined on θ. The log of a service during a full system run is the set of all logs on all data types in Θ:

$$\mathcal{L}(service) = \bigcup_{\forall \theta \in \Theta} \mathcal{L}(\theta).$$

Specifically, a log $\mathcal{L}(\theta)$ is defined as a sequence of events of four types: "write a consent," "write," "send" and "read" a piece of data. Below is the definition of $\mathcal{L}(\theta)$ that can contain a sequence of these four types of events.

```
{
    "Data_Type": "θ",
    "WriteConsent": {
      "consent": "ccons"/ "ucons"/ "scons"/ "acons",
      "ref_to": "LOCATION", "t": "TIME_STAMP",
      "who": "ENTITY_ID",
      "ID": "UNIQUE_EVENT_ID", "cond": "condition"},
    "Write": {
      "datatypeto": "θ′", "ref_to": "LOCATION",
      "ref_from": "LOCATION", "t": "TIME_STAMP",
      "who": "ENTITY_ID",
      "ID": "UNIQUE_EVENT_ID" },
    "Send": {
      "from": "ENTITY_ID", "to": "ENTITY_ID",
      "datatypeto": "θ′", "ref_to": "LOCATION",
      "ref_from": "LOCATION", "t": "TIME_STAMP",
      "ID": "UNIQUE_EVENT_ID" },
    "Read": {
      "ref_from": "LOCATION", "t": "TIME_STAMP",
      "who": "ENTITY_ID",
      "ID": "UNIQUE_EVENT_ID", "reason": "condition"}
}
```

For consent collection, the log event WriteConsent(*consent*, θ, *ref_to*, *t*, *who*, *ID*, *cond*) specifies that a collection, usage, storage, or access consent for the data type θ has been collected and written into a location with the reference address *ref_to* at time *t* by an entity *who* and satisfies a condition *cond*. *ID* is the unique ID of the log event. Note that *cond* is mainly used for access consent collection.

For the collection, usage, storage, and access purposes of format *who* : [*act*] : *datatype_to*, where $act \in \{calculate, create, \ldots, generate\}$, we define the log event Write(*datatype_to*, *ref_to*, *ref_from*, *t*, *who*, *ID*). This event says that a piece of data of $datatype\_to = \theta'$ is written to *ref_to* after it has been generated from data of type θ by an entity *who*, where *ref_from* denotes the reference address of the data of type θ. For instance, Write(*electricbill*, *DSt*-456, *DSt*-123, November 10, 2021 3:08 p.m., *SP*, *E*092) that has an event ID *E*092 captures that an electric bill was written by *SP* at 3:08 p.m. on November 10, 2021 into the location *DSt*-456, and electricity data (i.e., $\theta = electricity$) from the location *DSt*-123 was used to generate *electricbill*.

For the usage purposes of format *who*:[*act*]:*datatype_to*, where $act \in \{send, notify, share\}$ we define the log event Send(*from*, *to*, (*datatype_to*, *ref_to*), *ref_from*, *t*, *ID*) in which a piece of data of type $datatype\_to = \theta'$ that was generated from/related to a piece of data of type θ is made accessible by an entity *from* to an entity *to* at time *t*. Here, *ref_to* and *ref_from* are the location references where θ' is written, and where θ is stored, respectively. Note that θ' can be the same as θ (e.g., *from* sends a copy of the data).

The deletion action can be modeled by the log event Write(*ND*, *ref_to*, "-," *t*, *who*, *ID*), which captures that a piece of data of type θ has been deleted at time *t* by an entity *who*. Here, *ND* denotes the so-called undefined data type, which means that the data value in the location *ref_to* will be overwritten with a *ND* value. Finally, "-" means that *ref_from* is not defined in this case.

The store action is captured by the log event Write(*datatype_to*, *ref_to*, *ref_from*, *t*, *who*, *ID*). For example, a store event can be, for example, when the data is stored by a *TP* after receiving a transferred data, or when the data of type *datatype_to* has been generated using another type of data that needs to be stored. We also define a log event Write(*datatype_to*, *ref_to*, "-", *t*, *who*, *ID*) when *datatype_to* is only received and then stored without being transferred or generated from other data.

The data access action can be modeled with the log event Read(*ref_from*, *t*, *who*, *ID*, *reason*) and captures that an entity *who* accessed some data of type θ from *ref_from* at *t* for a specific *reason*. Finally, the transfer of a piece of data of type θ by an entity *from* to an entity *to* can be captured by Send(*from*, *to*, (θ, *ref_to*), *ref_from*, *t*, *ID*). This is a special case

of the *Send* event when $datatypeto = \theta$. In this case, *ref_to* is the reference to the location where the data will be stored after being transferred. Note that only a copy of the data will be sent and stored in the new place in case of data transfer. The location of the original data remains unchanged.

4.5 | System workflow

A full system workflow is presented here according to Figure 2. Details related to *CBN* internal operations such as channel configurations, adding peers to channels, etc. are omitted for simplicity.

1. The *CA* generates the system public parameters that are available for all the participants including the security parameters, PKI certificates for each participant, and hash functions that will be used throughout the system. It publishes these parameters including its public key.
2. The system participants form the *CBN*, install *SC* on their peer nodes and approve its definition, and install the applications A_{DSt}, A_{SP}, A_{TP} and A_{GA} to interact with the ledgers on *CBN*.
3. The *SP* populates $\pi(\theta)$ for each data type $\theta \in \Theta$, where Θ is the set of all data types required by the service according to *SP*′*s* privacy policy. After that, the *SP* packages all the privacy policies and calls the *SC* initialization function to submit them to *CBN*. In addition to checking the policy format, this achieves transparency since all data privacy policies follow this agreed/populated specification.
4. Before the *SP*′*s* privacy policy is committed on the ledger, the *GA* runs the submitted policy through the policy engine *PE* to ensure its compliance with GDPR before any data collection takes place. This guarantees an early detection of any breach. More details on how the automated verification is run can be found in Reference 15. If the compliance check is successful, the record is committed on the chain. The *GA* also communicates the result to the *DS* to start using the service in sub-step 4*a* in Figure 2. Otherwise, if the compliance check fails, the *GA* notifies the *SP* to fix the identified issues and submits it again in sub-step 4*b*.
5. After receiving the *GA*′*s* notification, the *DS* uses CP-ABE to encrypt and embed fine-grained access permissions on his/her personal data using the *SP*′*s* policy and the keywords that allow searching the encrypted data. If they exist, the *DS* gives *TPs* access rights using PRE techniques where *SP* is the delegator and *TP* is the delegatee. Finally, it uploads the encrypted data to the *DSt*. Note that *PRE* encryption is outsourced to *DSt*, which acts as a proxy, to save *DS*′*s* computation resources. Furthermore, re-encryption according to PRE will only go ahead when there is a request to access the data by *TP*. This is called lazy re-encryption mechanism,[24] which reduces computing costs.

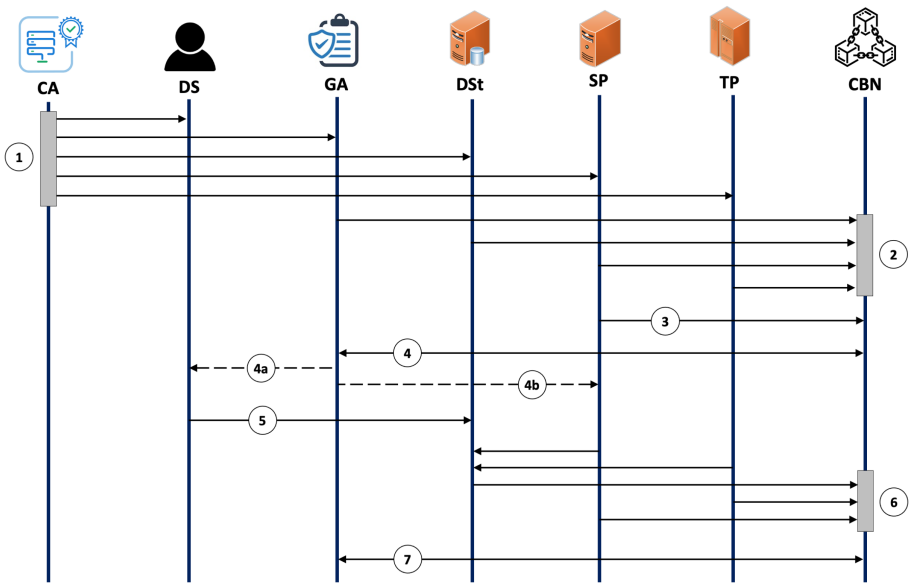

FIGURE 2 System workflow.

6. Operations on *DS′s* data take place according to the data lifecycle. For every operation, a set of log events $\mathcal{L}(\theta)$ will be recorded on *CBN* as a result of requests received by *DSt* from *SP* and/or *TPs*. These logs will be used for compliance checks later by the *GA*.
7. At any time during the system run, the *GA* chooses to run auditing and compliance checks to ensure the operations on the *DS′s* data comply with the approved data policies $\pi(\theta)$ for each $\theta \in \Theta$ in Step 4. These checks are carried out using the algorithms defined in the next Section 5.2, which are implemented as *SCs*. Should the *GA* finds any violation, the system run terminates and all participants are notified.

5 | SEMI-AUTOMATED GDPR COMPLIANCE SERVICE DESIGN AND ALGORITHMS

5.1 | Compliance between a log and a policy

We discuss five compliance properties between a log and a policy used in the verification algorithms, as follows:

1. **Consent collection compliance.** Besides $\pi_{\text{col}}.ccons = Y$, if there is a log event Write(θ, *ref_to*, "-", *t*, *who*, *ID*) in $\mathcal{L}(\theta)$, then there must be a corresponding log event WriteConsent(*consent*, *ref_to′*, *t′*, *who*, *ID*, *cond*) in $\mathcal{L}(\theta)$, where $t' \leq t$, $\theta \in \Theta$, and *ref_to′* ≠ "-."
2. **Usage purpose compliance.** (Below, ∨ stands for or)
   - For a usage purpose of a form *who* : *calculate* ∨ *create* ∨ *compute* ∨ *extract* ∨ *generate* : θ', the log complies with the usage purpose for θ if there is a log event Write(θ', *ref_to*, *ref_from*, *t*, *who*, *ID*) at *t*, where θ' is derived from data type θ.
   - For a usage purpose of a form *who* : *send* ∨ *notify* ∨ *share* : θ', the log complies with the purpose for θ if there is a log event Send(*from*, *to*, θ', *ref_to*, *ref_from*, *t*, *ID*) where θ' is derived from θ.
3. **Storage place compliance.** If there is a log event Write(θ, *ref_to*, *ref_from*, *t*, *who*, *ID*) or Write(θ, *ref_to*, "-", *t*, *who*, *ID*), then *ref_to* must comply with π_{str} of $\pi(\theta)$ (i.e., *ref_to* ∈ $\pi_{\text{str}}.splaces$ where $\pi_{\text{str}} \in \pi_\theta$).
4. **Deletion compliance and enforcement.** If there is a log event Write(θ, *ref_to*, "-", t_1, *who*, *ID*), there must be an event Write(*ND*, *ref_to*, "-," t_2, *who*, *ID*), where $t_2 - t_1 \leq \pi_{\text{del}}.delay$. If there is no Write(*ND*, *ref_to*, "-," t_2, *who*, *ID*) by the time the delay elapsed, the deletion will be enforced from *ref_to* by the service entity *SYS*, and that will produce a Write(*ND*, *ref_to*, "-," t_2, *SYS*, *ID*) log.
5. **Access compliance.** If there is a log event Read(*ref_from*, *t*, *who*, *ID*, *reason*), then *who* is an entity defined in π_{acc} of $\pi(\theta)$. In addition, the conditions in $\pi_{\text{acc}}.entity.conditions$ and the purposes in $\pi_{\text{acc}}.entity.purps$ must be satisfied. This means that *reason* must be a subset of/equal to *conditions*. Regarding the access consent, if *acons* = *Y* in π_{acc}, there must be a log WriteConsent(*consent*, *ref_to*, *t*, *who*, *ID*, *cond*), where *consent* = *acons* and *ref_to* complies with π_{acc}.

5.2 | Verification procedure

In the log for a given service $\mathcal{L}$(*service*), the logs for each datatype supported/used by the service will be stored and managed separately to improve search efficiency. Logs only contain a reference (i.e., *ref_to* or *ref_from*) to the corresponding entry stored in the *DSt*. Each entry in *DSt* is a triplet (*value*, *loc_ref*, *t*), where *value* is the concrete value of the datatype (e.g., if *datatype* = *name*, then *value* can be "Peter"), *loc_ref* is the concrete location/address where the data can be found physically, and *t* is the last modified time. In addition, *value* = *ND* if the data has been deleted, therefore, the value of *loc_ref* prior to deletion is kept for a certain period of time for accountability and auditing purposes. For verifying compliance of each sub-policy, we define a dedicated *SC* function namely *SmartCollection*(), *SmartUsage*(), *SmartStorage*(), *SmartDeletion*(), and *SmartTransfer*().

We start with Algorithm 1 that presents the details of *SmartCollection*(*Consent*) that is responsible for checking $\mathcal{L}$(*service*) in the *CBN* for verifying compliance regarding obtaining data collection consents.

In the following, we explain the algorithm steps in details as shown in its flowchart in Figure 3.

1. Iterate through each predefined time interval $\bigtriangleup T$.

Algorithm 1. SmartCollection (Consent)

for each $\bigtriangleup T$ interval **do**
  for $\forall\theta \in \Theta$ **do**
    if $\pi(\theta)_{\text{col}}.ccons = Y$ **then**
      if $\exists$ Write(θ, *ref_to*, "-", t, *who*, *ID*) $\in \mathcal{L}(\theta)$ **then**
        if $\exists$ WriteConsent(*consent*, *ref_to*$'$, t', *who*, *ID*, *cond*) $\in \mathcal{L}(\theta)$, where $t' \leq t \wedge consent = ccons \wedge ref\_to \neq$ "–" **then**
          Return Compliance/Do Nothing
        else
          Return Violation
        end if
      end if
    end if
  end for
end for

2. Iterate through each data type θ in the set Θ.
3. Check the consent status (*ccons*) of the data type θ.
   - If the condition is true, proceed to the next step.
   - If the condition is false, end the current data type's processing and move to the next data type.
4. Check whether a *Write* action with the specified parameters exists in the log $\mathcal{L}(\theta)$ for the data type θ.
   - If it exists, proceed to the next step.
   - If it does not, end the current data type's processing and move to the next data type.
5. Check if a *WriteConsent* entry exists in $\mathcal{L}(\theta)$, satisfying the following conditions:
   - The timestamp t' of *WriteConsent* is less than or equal to t.
   - The consent value is the same as the "*ccons*" value.
   - The *ref_to* field is not equal to "-." This condition is required to ensure that the consent has not been deleted from the location *ref_to* (i.e., the consent has not been revoked).
   - If these conditions are met, return compliance or do nothing.
   - If any condition is not met, return a violation.

Note that *SmartUsage*() checks $\mathcal{L}$(*service*) for verifying compliance regarding data usage consent and usage purposes. The verification for *SmartUsage*() is similar to *SmartColection*(*Consent*), but with the log event *WriteConsent* containing usage consent instead of collection consent, and $\pi(\theta)_{\text{use}}.ucons = \text{Y}$.

Algorithm 2 implements the *SmartCollection*(*Purpose*) function. In the following, we explain the steps as shown in its flowchart in Figure 4.

1. Iterate through each predefined time interval $\bigtriangleup T$.
2. Iterate through each data type θ within the set Θ.
3. Check if there is a collection purpose (*cpurps*) for the data type θ that involves actions like *calculate*, *create*, etc., performed by *who* to generate another data type θ'.
   - If it exists, check if there exists a *Write* action with parameters (θ'', *ref_to*, *ref_from*, t, *who*, *ID*) in the log $\mathcal{L}(\theta)$, where θ'' is different from θ'.
     - If the above condition is met, return a violation.
4. Otherwise, check if the collection purpose (*cpurps*) for of the data type θ involves actions like *Send*, *Notify*, or *Share* to another data type θ'.
   - If it exists, check if there exists a *Send* action with parameters (*from*, *to*, θ'', *ref_to*, *ref_from*, t, *ID*) in the log $\mathcal{L}(\theta)$, where θ'' is different from θ'.
     - If the above condition is met, return a violation.

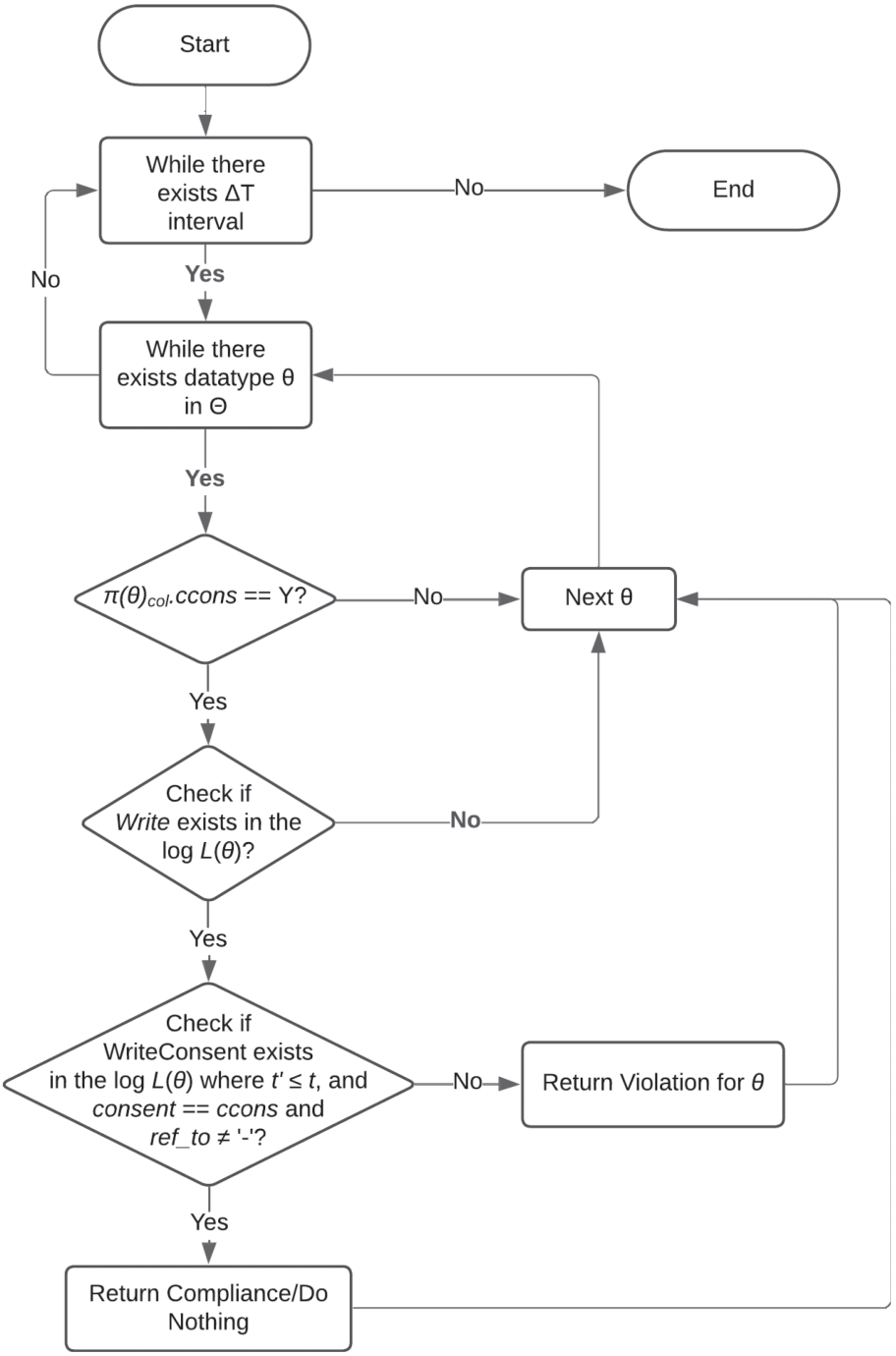

FIGURE 3 Flowchart of SmartCollection (Consent).

SmartStorage(*Places*) in Algorithm 3 checks $\mathcal{L}$(*service*) for verifying compliance regarding data storage as follows. The following steps are shown in its flowchart in Figure 5:

1. Iterate through each predefined time interval $\triangle T$.
2. Iterate through each data type θ within the set Θ.
3. Check if there is a *Write* action with these parameters (θ, *ref_to*, *ref_from*, *t*, *who*, *ID*) or (θ, *ref_to*, "-," *t*, *who*, *ID*) in the log $\mathcal{L}(\theta)$.

- If it exists, verify if *ref_to* parameter of the *Write* action is included in the storage places (*places*) defined for θ. This ensures that the storage location for the 'Write' action is authorized.

Algorithm 2. SmartCollection (Purpose)

for each $\triangle T$ interval **do**
  for $\forall\theta \in \Theta$ **do**
    if $\pi(\theta)_{\text{col}}.cpurps = who{:}[act]{:}\theta'$, where $act \in \{calculate, create, etc.\}$ **then**
      if $\exists$ Write(θ'', *ref_to*, *ref_from*, *t*, *who*, *ID*) $\in \mathcal{L}(\theta)$ where $\theta'' \neq \theta'$ **then**
        Return Violation
      end if
    else if $\pi(\theta)_{\text{col}}.cpurps = to{:}[act]{:}\theta'$, where $act \in \{send, notify, share\}$ **then**
      if $\exists$ Send(*from*, *to*, θ'', *ref_to*, *ref_from*, *t*, *ID*) $\in \mathcal{L}(\theta)$ where $\theta'' \neq \theta'$ **then**
        Return Violation
      end if
    end if
  end for
end for

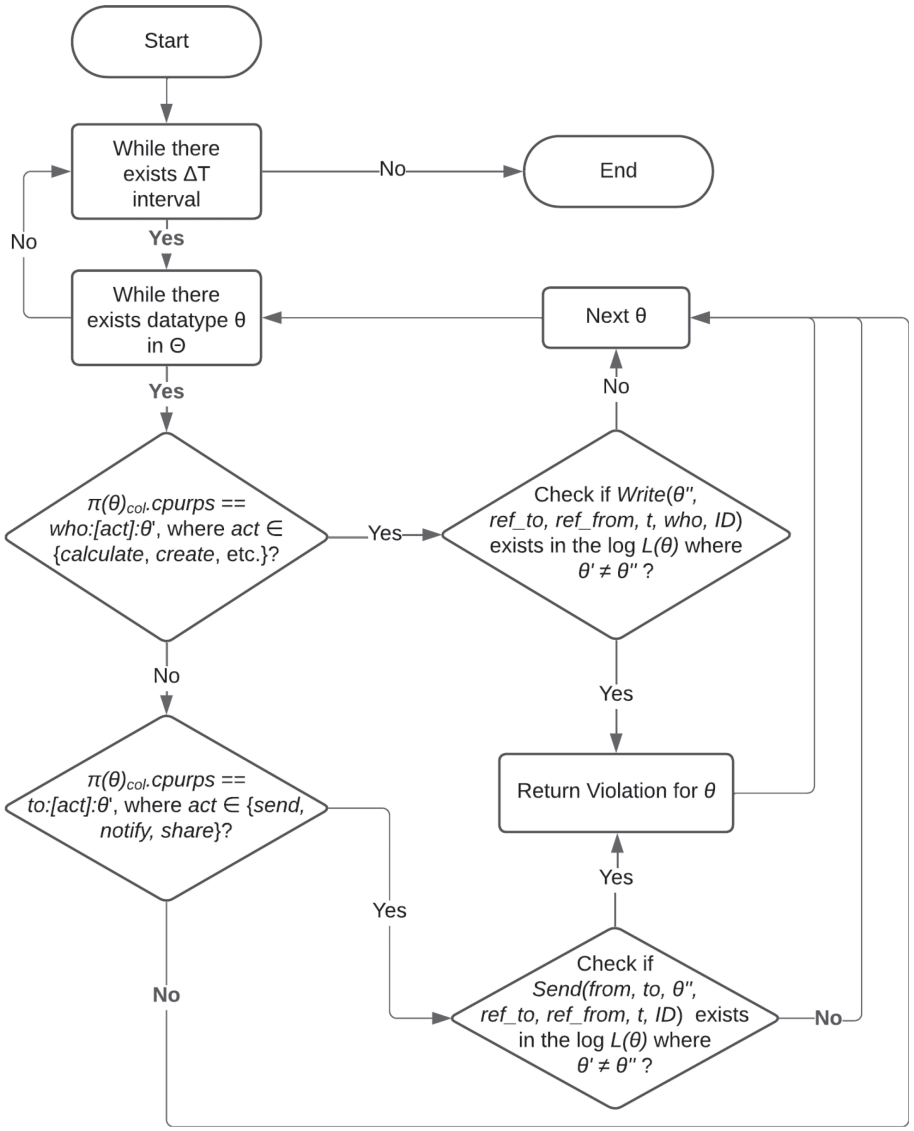

FIGURE 4 Flowchart of SmartCollection (Purpose).

Algorithm 3. SmartStorage (Places)

for each $\triangle T$ interval **do**
  for $\forall\theta \in \Theta$ **do**
    if ($\exists$ Write(θ, *ref_to*, *ref_from*, *t*, *who*, *ID*) $\vee$ Write(θ, *ref_to*, "-," *t*, *who*, *ID*)) $\in \mathcal{L}(\theta)$ **then**
      if $ref\_to \in \pi(\theta)_{\text{str}}.places$ **then**
        Return Compliance/ Do Nothing
      else
        Return Violation
      end if
    end if
  end for
end for

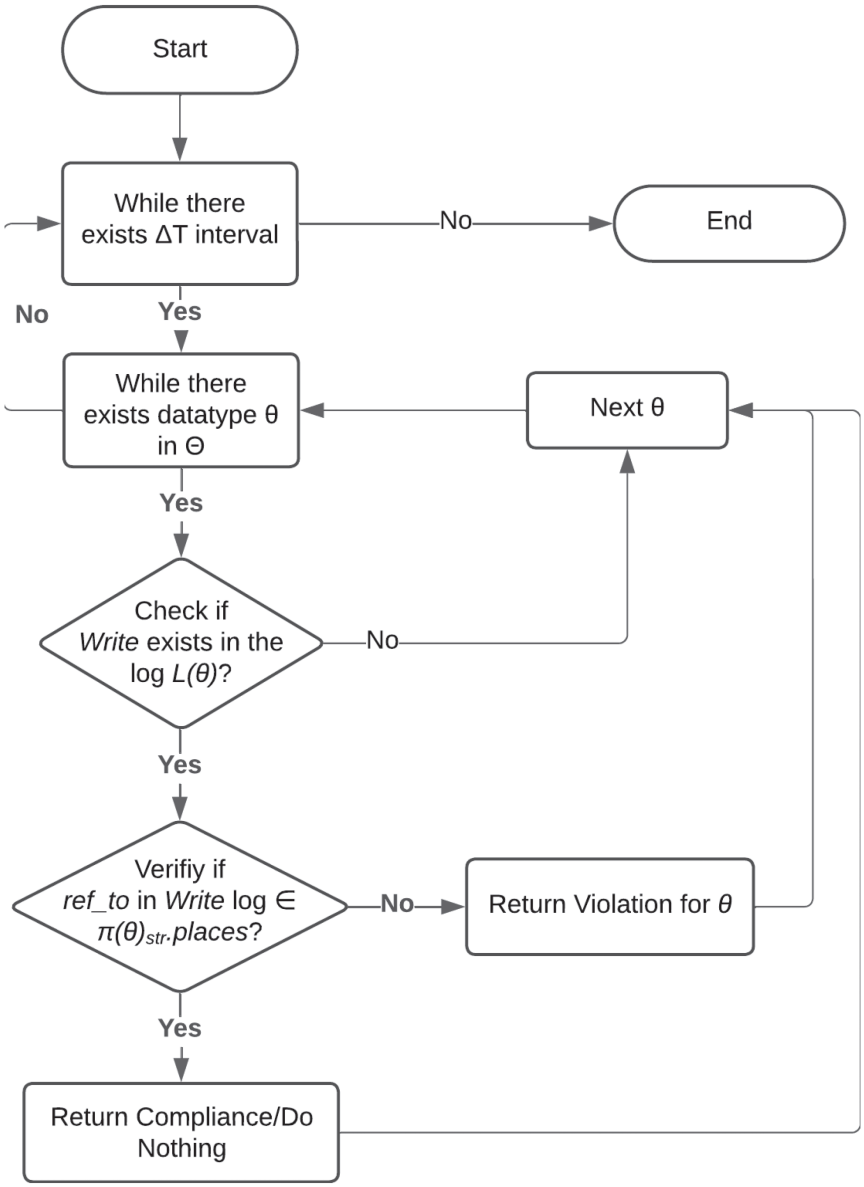

FIGURE 5 Flowchart of SmartStorage (Places).

- If the verification is successful, return compliance or do nothing.
- Otherwise, return a violation that indicates that the *Write* action does not comply with the authorized storage locations.

SmartDeletion(*Delay*) captures the compliance verification and automated deletion enforcement as defined in Algorithm 4. Note that *SYS* is a system identity that enforced a specific action where compliance was not verified. The flowchart for this algorithm is illustrated in Figure 6:

1. Initialize delay value $\pi(\theta)_{\text{del}}.delay$ to *delayval* for the deletion process.
2. Iterates through each predefined time interval $\triangle T$.
3. Iterate through each data type θ within the set Θ.
4. Check if there is a *Write* action with parameters (θ, *ref_to*, "-," t_1, *who*, *ID*) in the log $\mathcal{L}(\theta)$. This identifies if any write operation related to the entity θ occurred.
5. Check if there is a deletion *Write* action with parameters (*ND*, *ref_to*, "-," t_2, *who*, *ID*) in the log $\mathcal{L}(\theta)$, where $t_2 - t_1 \leq$ *delayval*. This verifies if a deletion *Write* action occurred within the delay period.
   - If it exists, return Compliance/Do Nothing. This indicates the data has been deleted within the allowed delay period, and hence is compliant.
   - Otherwise, if the delay value *delayval* has elapsed, enforce the deletion of data at *ref_to*, add a log event *Write(ND, ref_to*, "-," t_3, *SYS, ID)* in $\mathcal{L}(\theta)$ to be stored in the ledger, and Return violation as the deletion is enforced by the system.

For *SmartTransfer*(*Thirdparties*), we define Algorithm 5 that is illustrated via its flowchart in Figure 7 and explained below.

1. Iterate through each predefined time interval $\triangle T$.
2. Iterate through each data type θ within the set Θ.
3. Verify if the policy $\pi(\theta)_{\text{acc}}$ (i.e., access control permissions for θ) is empty. This checks if there are any access control permissions defined for θ.
4. Check if there exists a *Read* action with parameters (*ref_from*, *t*, *who*, *ID*, *reason*) in the log $\mathcal{L}(\theta)$. This verifies if there was a read operation related to θ.
5. If the *Read* action exists, verify its conditions as follows.
   - If *who* is an entity in $\pi(\theta)_{\text{acc}}$ and the conditions specified in *who.conditions* are satisfied: Return compliance or do nothing.
   - Otherwise, return violation.
6. Otherwise, check if there exists a *Send* action with parameters (*from*, *to*, *datatype_to*, *ref_to*, *ref_from*, *t*, *ID*) in the log $\mathcal{L}(\theta)$. This verifies if there was a send operation related to θ.

Algorithm 4. SmartDeletion (Delay)

Let $\pi(\theta)_{\text{del}}.delay = delayval$.
for each $\triangle T$ interval **do**
  for $\forall\theta \in \Theta$ **do**
    if $\exists$ Write(θ, *ref_to*, "-", t_1, *who*, *ID*) $\in \mathcal{L}(\theta)$ **then**
      if $\exists$ Write(*ND*, *ref_to*, "-", t_2, *who*, *ID*), where ($t_2 - t_1 \leq delayval$) **then**
        Return Compliance/Do Nothing
      else if *delayval* has elapsed **then**
        Enforce data deletion at *ref_to*.
        Add *Write(ND, ref_to, "-," t_3, SYS, ID)* to $\mathcal{L}(\theta)$
        Return Violation - Deletion is enforced
      end if
    end if
  end for
end for

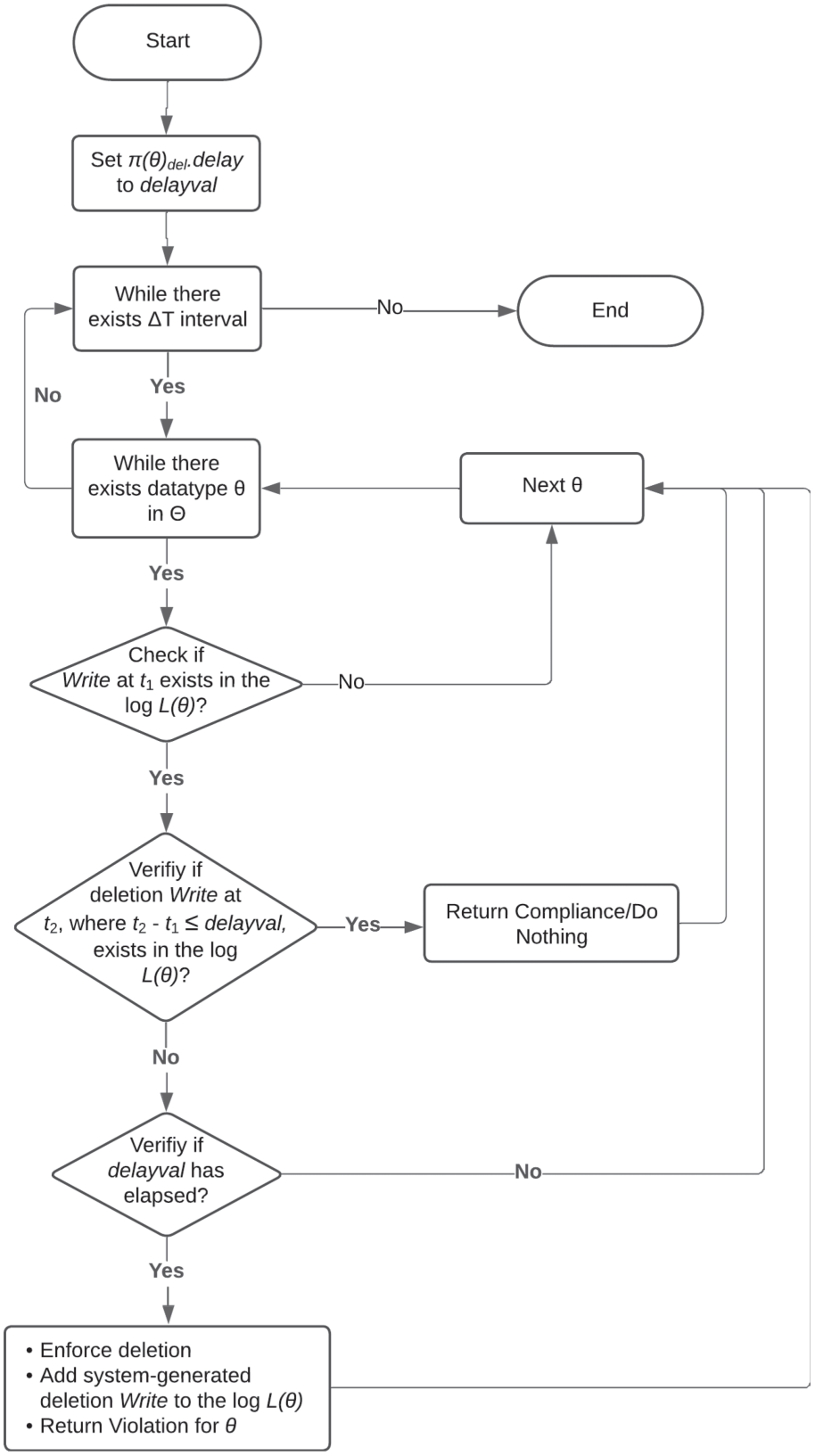

FIGURE 6 Flowchart of SmartDeletion (Delay).

Algorithm 5. SmartTransfer (Thirdparties)

for each $\triangle T$ interval **do**
  for $\forall\theta \in \Theta$ **do**
    if $\pi(\theta)_{\text{acc}} \neq \emptyset$ **then**
      if $\exists$ Read(*ref_from*, *t*, *who*, *ID*, *reason*) $\in \mathcal{L}(\theta)$ **then**
        if *who* is an entity in $\pi(\theta)_{\text{acc}}$ AND the conditions in *who.conditions* are satisfied **then**
          Return Compliance/Do Nothing
        else
          Return Violation
        end if
      else if $\exists$ Send(*from*, *to*, *datatype_to*, *ref_to*, *ref_from*, *t*, *ID*) **then**
        if *to* is an entity in $\pi(\theta)_{\text{acc}}$ AND the conditions in *to.conditions* are satisfied AND *to.transfer* = Y **then**
          Return Compliance/Do Nothing
        else
          Return Violation
        end if
      end if
    end if
  end for
end for

7. If the *Send* action exists, verify its conditions as follows.
   - If *to* is an entity in $\pi(\theta)_{\text{acc}}$, the conditions in *to.conditions* are satisfied, and *to.transfer* = Y: Return compliance or do nothing.
   - Otherwise, return violation.

Finally, Algorithm 6 checks the compliance of a privacy policy against the GDPR, which is implemented in the *PE*. Given its simplicity, there is no flowchart for this algorithm. Note that we only consider consent as a legal basis for data processing, however, in the GDPR, there are also other types of legal basis such as legitimate, vital or public interest.

6 | THE FULL SYSTEM DEPLOYMENT AND OPERATIONS

6.1 | System setup

Let $(\mathbb{G}_1, \cdot)$ and $(\mathbb{G}_2, \cdot)$ be two cyclic groups of prime order q and $e : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$ be an efficient admissible bilinear map, which has the properties of bilinearity, computability and non-degeneracy. The *CA* chooses a random generator $P \in \mathbb{G}_1$, a random master key $s \in \mathbb{Z}_q^*$, and four collision resistant hash functions: H_1: $\{0, 1\}^* \rightarrow \mathbb{G}_1$, H_2: $\mathbb{G}_2 \rightarrow \{0, 1\}^*$, H_3: $\{0, 1\}^* \rightarrow \mathbb{Z}_q^*$ and H_4: $\{0, 1\}^* \times \mathbb{G}_2 \rightarrow \mathbb{Z}_q^*$. The *CA* then sets $P_{pub} = sP$ as its public key and publishes the system parameters $(\mathbb{G}_1, \mathbb{G}_2, q, e, P, P_{pub}, H_1, H_2, H_3, H_4, SEnc(\cdot))$ where $SEnc(\cdot)$ is a secure symmetric encryption algorithm.

Algorithm 6. PolicyCompliance

for each $\pi_* \in \{\pi_{\text{col}}, \pi_{\text{use}}, \pi_{\text{str}}, \pi_{fw}, \pi_{\text{del}}\}$ **do**
  if GDPRCompliance(π_*) = *False* **then**
    Return Violation
  else if for all π_*: GDPRCompliance(π_*) = *True* **then**
    Return Compliance
  end if
end for

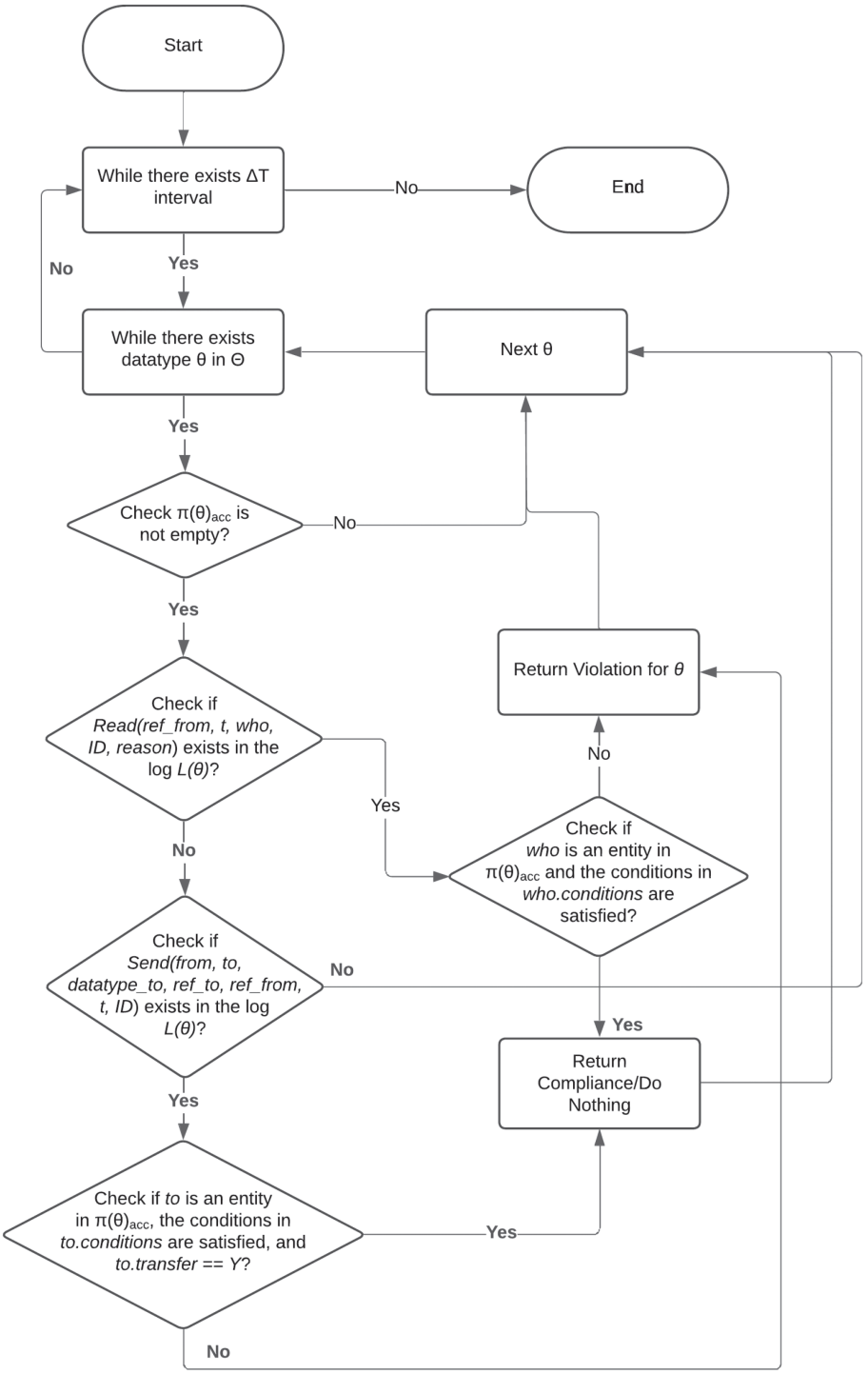

FIGURE 7 Flowchart of SmartTransfer (Thridparties).

After receiving the system public parameters, GA, SP, DSt and TP form the CBN and establish $Ch1$ and $Ch2$. The SC is installed and approved on all peer nodes. Then, all the applications A_{DSt}, A_{SP}, A_{TP} and A_{GA} are installed to interact with SC on the CBN. Note that each organization can have its own CA as long as it is recognized by the system and uses its parameters. Therefore, details of acquiring keys and certificates from the CA for each organization are omitted for simplicity. Instead, we focus on the CP-ABE operations during the submission of $DS's$ encrypted personal data and access to this data during the data lifecycle. Finally, we assume a secure digital signature scheme (e.g., ECDSA) is used to sign the exchanged messages.

6.2 | Service registration

The *SP* uses the policy specification for each data type θ that is needed for its service. The *SP* will generate $\mathcal{P} = \bigcup_{\forall\theta\in\Theta} \pi(\theta)$ the representation of its privacy policy (step 3 in the system workflow in Section 4.5). Then, the *SP* defines the attribute universe $\mathcal{U}$ that will be used during the CP-ABE of personal data by the *DS*. $\mathcal{U}$ is defined by the *SP* because it is related to the service itself and any *TP* that requires access to the data for processing. It can contain specific attributes that will identify the attribute set for those who will be granted access according to the access policy (e.g., $\mathcal{U}$ = {"credit agency," "fraud prevention," "law enforcement," etc.} is an attribute universe for potential *TPs* a *SP* might use while providing a service to their customers).

Once $\mathcal{U}$ and $\mathcal{P}$ are ready, the *SP* invokes *init*() function in *SC* via its application A_{SP} and passes $\mathcal{U}$ and $\mathcal{P}$. Before the result is committed, the *GA* passes $\mathcal{P}$ to *PE* that verifies if the policy complies with the GDPR. In our case, we focus on compliance with the rules in the GDPR that are closely related to the sub-policies of *SP*. Specifically, we focus on the collection of consents, purpose minimization and consistency, retention delay, and right of access. The basis of the verification relies on the algorithm we proposed in the DataProVe tool, which uses logic inference rules to prove three types of conformance relations (privacy, data protection, and functional conformance). DataProVe feasibility has been demonstrated in Reference 15 and more information can be found on GitHub.[25] If $\mathcal{P}$ complies with GDPR (the considered rules), the transaction is approved and committed on *CBN*. The *GA* then sends $\mathcal{P}$ and $\mathcal{U}$ to the *DS* to start using the service (step 4a of the system workflow).

6.3 | Encryption and submission of personal data

Utilizing the CP-ABE with accountability and PRE schemes, which were explained in Section 3.4, we provide details on how the *DS* encrypts his/her personal data to enforce restrictive fine-grained access control and automatic revocation of access rights.

1. $Setup(\lambda, \mathcal{U}) \rightarrow (PK_{ABE}, MK_{ABE})$: The *DS* requests the *CA* to run the setup algorithm using a security parameter λ and the universe of attributes $\mathcal{U}$ received from the *GA*. The *CA* outputs a public key PK_{ABE} and a master private key MK_{ABE}. The MK_{ABE} will be used later to produce a secret key SK_{AS} for an entity E that is associated with a set of attributes $AS \subset \mathcal{U}$.
2. $KeyGen(MK_{ABE}, E_{ID}, AS) \rightarrow (SK_{AS})$: Based on $\mathcal{P}$, for each entity E in the access policy π_{acc}, the *DS* extracts a set of attributes *AS* that describes E and its access rights under π_{acc}. *AS* include each entity's ID E_{ID}, conditions for access (e.g., consent is given, the purpose is approved, etc.), and any relevant information that identifies those entities. For instance, AS = {"*SP_ID*" $\wedge$ "*whocanaccess*" $\wedge$ "$(ent1 : [cond1, \ldots, condn])$"}). This way, each authorized entity E will have its private key SK_{AS} that implicitly includes its *AS* and E_{ID}. Note that the *SP* is an example of these entities. Again, *KeyGen* is run by the *CA* based on *DS*'*s* request and the resulted secret key(s) are delivered securely to those entities.
3. $SEnc(data, S_r) \rightarrow Enc_C$: The *DS* chooses a random symmetric key $S_r \in \mathbb{Z}_q^*$ and encrypts his/her data using the symmetric encryption algorithm $SEnc(\cdot)$. For simplicity, we assume all the data requested by *SP* is encrypted under the same key S_r. It is possible for the *DS* to encrypt his/her data under different keys where each will be associated with a different entity. However, this adds more complexity to the system and requires further investigation, which is left for future works.
4. The *DS* uses the public key of the *SP* (i.e., the recipient) to produce a searchable encryption of the keyword set *KW*, which identifies the data types required by the *SP*, as follows: $Enc(PK_{SP}, KW) \rightarrow (S_{PEKS})$. The set *KW* should be concise to avoid delays that may occur during the search process.
5. Using CP-ABE, the *DS* encrypts the symmetric key S_r under *AS*, which defines the access policy, as follows: $Enc(PK_{ABE}, S_r, AS) \rightarrow (CT_{ABE})$.
6. The *DS* signs the tuple $\{Enc_C, S_{PEKS}, CT_{ABE}\}$ using his/her certificate and generates a digital signature σ_{DS} and finally uploads $\{Enc_C, S_{PEKS}, CT_{ABE}, \sigma_{DS}, H_1(T)\}$ to *DSt* where T is a timestamp.

This process can achieve restrictive fine-grained access control to *DS*'*s* data since the access policy to decrypt mirrors that in π_{acc} and can be extended to have more contextual conditions. The automatic revocation of access rights is achieved via "$(ent1 : [cond1, \ldots, condn])$" in *AS* since these conditions can include a time seal that expires when E no longer needs access to *DS*'*s* data. The direct application of this automatic revocation is the *delay* value in π_{del} that indicates the

data must be deleted thus, access is no longer permitted. This design gives *DS* a lot of flexibility to enforce access controls according to the *SP* policy.

6.4 | Full data lifecycle operations and logs

Assuming that *DSt* acts as a proxy, *SP* is the delegator and *TP* is the delegatee for the purpose of PRE scheme, the first step for the *SP* is to execute its collection policy π_{col} to collect *DS*′*s* data from the *DSt*. However, before doing so, the *SP* should register the collection of relevant consents and purposes following the details in the policy $\mathcal{P}$. For each data type θ, the *SP* invokes the following event *WriteConsent*(*consent*, *GAL*, Θ, t, *SP*), which specifies that a consent of type *consent* has been collected by the *SP* and written to the ledger *GAL* for the set of data types in Θ at time t. After that, the *SP* executes $TokenGen(SK_{SP}, kw')$ to generate a search token $\tau_{kw'}$ and sends it to *DSt*. The later executes $Test(S_{PEKS}, \tau_{kw'})$ and returns true if $\tau_{kw'} \in KW$ or false otherwise. If the *Test* output is true, the *SP* receives the corresponding tuple $\{Enc_C, S_{PEKS}, CT_{ABE}, \sigma_{DS}, H_1(T)\}$. After that, *SP* uses its key SK_{AS}, which is generated and associated with *AS* in Section 6.3, to decrypt the symmetric encryption key as follows: $Dec(CT_{ABE}, SK_{AS}) \rightarrow S_r$. Finally, it obtains *DS*′*s* data as follows $SDec(Enc_C, S_r) \rightarrow data$.

Following the successful collection of *DS*′*s* data, the *SP* can invoke the events related to collection, usage, transfer, and storage actions. All these operations generate logs that are recorded on *GAL* for auditing and compliance checks later. If a *TP* access is required according to the *SP*′*s* privacy policy, the following steps must be taken by the *SP*:

1. As a delegator, *SP* runs $RKeyGen(SK_{AS}, \pi'_{\text{acc}}, kw', TP_{ID}, KFN) \rightarrow (RK)$ using its secret key SK_{AS}, a new access policy π'_{acc} that gives a *TP* access to specific data types, a new keyword kw' to generate re-encryption key *RK*, *TP*′*s* identity and *KFN*. The *SP* can choose kw' different from the keyword set that was used earlier by the *DS* to encrypt in Section 6.3. This allows the *SP* to grant access to *TPs* under different keywords and update them if necessary.
2. The *SP* shares the re-encryption key *RK* with *DSt* to encrypt the CT_{ABE} and generates a new ciphertext CT_{PRE} that can be accessed by the *TP*.
3. To ensure compliance, the *SP* invokes the event Send(*SP*, *TP*, (θ', *DSt*), (θ, *DSt*), t) to specify that *SP* is sending data of type θ', which was generated from type θ, to *TP* from locations in *DSt* at time t. This log will be written on both ledgers *GAL* and *DAL*.

The *TP* can follow the same steps (i.e., $TokenGen(SK_{TP}, kw')$ to generate a search token and sends it to *DSt* to retrieve CT_{PRE} for decryption). Similarly, the *TP* invokes relevant log events to record logs only on *DAL* according to the system model in Figure 1.

7 | PERFORMANCE EVALUATION AND ANALYSIS

7.1 | The scenario

In this section, we took a real world energy service and its relevant privacy policy*. For demonstration purposes, we consider a simplified policy, which is a "subset" of OVO privacy policy to show how our compliance service works. The *SP* collects the following data:

1. The customer's *DS* contact details *contact* (e.g., full name, email, property address, phone number, etc.)
2. Account data *account* (e.g., unique account number, national insurance number, photos of the meter, etc.)
3. Financial data *financial* (e.g., bank account details.)
4. Energy consumption data *energy* (e.g., meter readings.)

The privacy policy of *SP* is then defined as follows:

1. $\pi(contact) = (\pi_{\text{col}}, \pi_{\text{use}}, \pi_{\text{acc}})$, where:
   - $\pi_{\text{col}} = \{Y, \{[SP] : create : [account]\}\}$

- $\pi_{use} = \{Y, \{[SP] : send : [notification, bill] : [DS]\}\}$
- $\pi_{acc} = \{otherSP(Y, [switch], [otherSP] : send : [prediction] : [DS], Y\}$

2. Similarly, for $\pi(account)$, we have:
   - $\pi_{col} = \{Y, \{[SP] : create : [account]\}\}$
   - $\pi_{use} = \{Y, \{[SP] : create : [bill]\}\}$
   - $\pi_{acc} = \{marketingunit(Y, [loyaltyprogram], [DS] : receive : [voucher], N\}$
3. For the financial data, $\pi(financial)$, we have:
   - $\pi_{col} = \{Y, \{[SP] : create : [account]\}\}$
   - $\pi_{use} = \{Y, \{[SP] : receive : [payment]\}\}$
   - $\pi_{acc} = \{creditcheckunit(Y, [-], [creditcheckunit] : create : [creditprofile], Y\}$
4. Finally, for meter reading, $\pi(energy)$, we have:
   - $\pi_{col} = \{Y, \{[SP] : create : [bill]\}\}$
   - $\pi_{use} = \{Y, \{[SP] : calculate : [fee]\}\}$
   - $\pi_{acc} = \{marketingunit(Y, [loyaltyprogram], [DS]: receive: [voucher], N), otherSP(Y, [switch], [otherSP]: calculate: [prediction], Y)\}$
5. For all the four data types above, the deletion policy is $\pi_{del} = \{6y, DSt\}$, which means that all $DS's$ data must be deleted from *DSt* within 6 years after the contract period starts assuming the contract is for 3 years.
6. For the four data types above, the storage policy is $\pi_{str} = \{Y, \{\{[SP] : create : [bill]\}, [SP] : calculate : [fee], [marketingunit] : send : [voucher], [otherSP] : calculate : [switch], [otherSP]: calculate: [prediction]\}, DSt\}$.

7.2 | Efficiency metrics

The performance evaluation is done on an Ubuntu 20.04 server with Intel Xeon 6248 2.5 GHz, 64 GB RAM and 1 TB storage. The *CBN* is setup using HLF v2.5 with four peer nodes, two channels' configurations and one orderer node. Moreover, there are four applications to interact with the *CBN* that are available for each system's participant as depicted in Figure 1. These applications and the *SCs* are written in Node.js. Furthermore, Crypto++ library 8.7[26] and cpabe toolkit[27] are used for cryptographic operations. The performance of the *CBN* is measured using Hyperledger Caliper v0.5.0 benchmarking tool.[28] The following efficiency metrics are considered:

1. Compliance check overhead including the initial $SP's$ policy check and the periodic operation logs checks later.
2. Transaction latency that measures the time from when a transaction is submitted to when the result becomes available on the *CBN*.
3. Transaction throughput which is the rate at which valid transactions are committed on *CBN*. It is expressed as transactions per second (TPS).

Note for Caliper, the benchmark runs using 10 workers processes where the number of submitted transactions (TXs) is 1000 at a send rate that varies between 25 and 500 TPS. Finally, the time overhead needed for the system setup (e.g., create the system public parameters, form the *CBN*, configure/create *Ch*1 and *Ch*2 on the *CBN*, install *SC* on the peer nodes, and populate the specifications for the privacy, etc.) is not considered here. The reason for that is it only happens once before the service starts and has no impact on the operations later. Once the *CBN* is up and running, the system starts when a *SP* registers its privacy policy, and it gets checked for compliance to start the data lifecycle. Hence, our performance evaluation starts from point 3 of the system workflow in Section 4.5.

7.3 | Security and privacy analysis

In this section, we evaluate the proposed system against the design goals we identified in Section 4.1, except for efficiency, which is explained in more details in the next Section 7.4. For demonstration purposes, we assume a set of example operation logs.

7.3.1 | Semi-automated GDPR compliance checks

In the proposed system, these checks are achieved by enforcing compliance automatically via the *PE* by the *GA* before the service can run. When *SP* upload its privacy policies into *CBN* via calling the *SC* initialization function, Algorithm 6 in Section 5.2 is automatically triggered before the result is committed to the ledgers and the *DS* receives the green light to use the service. Otherwise, if the *SP*′*s* policy fails to comply with GDPR, the *GA* asks the *SP* to fix the identified issues and submit it again. Moreover, periodically, the *GA* checks the operation logs to ensure continuous compliance. This assures the *DS* that their data is handled accordingly without having to perform any extra tasks.

7.3.2 | Security and fine-grained access control

Once the *SP*′*s* privacy policy $\mathcal{P}$ is approved, the *DS* can start using the service, where the contact details will be collected according to $\pi(contact)$, and based on $\pi(account)$, the national insurance number is needed. As discussed in point 5 of the system workflow in Section 4.5, these *DS*′*s* personal data are encrypted using PEKS and CP-ABE algorithms according to the attributes $\mathcal{U}$ required by the *SP*. Therefore, even if *DSt* is compromised, adversaries will not be able to read the data unless they have been given access in the policy π_{acc}. Moreover, it is infeasible for *DSt* to identify the *DS* via the stored data given the restrictive access on the encrypted data. Other entities such as *TP* can only access pieces of specific data types if and only if *SP* shares a new re-encrypted ciphertext CT_{PRE} with *TP*. As pointed out earlier in our threat model, system entities do not collude directly or indirectly for illegal access operations. Therefore, *DS*′*s* data security can be guaranteed in this context.

7.3.3 | Transparency

The transparency of the proposed system can be illustrated in the following examples. After the registration phase and data collection, operations logs will be generated and recorded as a trace of log events according to the specification defined in Section 4.4. Each new log event generated by *DSt* and *SP* will be recorded on the *GCL* ledger, while log events related to data access from *TP* will be recorded on the *DAL* ledger. For instance, when *SP* collects a piece of contact data (e.g., full name) from a *DS* at 1:01 p.m. on March 2, 2022, the following *write* log event will be generated and committed to the *GCL* ledger.

$\mathcal{L}(contact, E145) =$
{"***Write***": {
"***datatypeto***": "*contact*", "***ref_to***": "*DSt-Ref1234*",
"***ref_from***": "-", "***t***": "*02/03/2022 13:01*",
"***who***": "*SP*", "***ID***": "*E145*"}
}

This log specifies that a piece of contact data was written by *SP* into the location *DSt-Ref1234* at 1:01 p.m. on March 2, 2022. The log has the unique ID *E*145. In this case *ref_from* is not specified as the data has not been generated from any other data. Later on, let us assume that there is another *write* log recorded into the *GCL* ledger as follows:

$\mathcal{L}(contact, E235) =$
{"***Write***":{
"***datatypeto***": "*advideo*", "***ref_to***": "*DSt-Ref5567*",
"***ref_from***": "*DSt-Ref1234*", "***t***": "*12/04/2022 10:14*",
"***who***": "*SP*", "***ID***": "*E235*"}
}

This log event specifies that an advertisement video was written by *SP* into *DSt-Ref5567* at 10:14 a.m. on April 12, 2022. The value of *ref_from* is *DSt-Ref1234*, which means that the video was generated using the contact data or contains the contact data, which is stored at $DSt - Ref1234$. Let us assume *GA* invokes the compliance check at 10:30 a.m. on April 12, 2022, then based on Algorithm 2, a violation of $\pi(contact)$ is detected, as a piece of contact data is not allowed to be used

to create an advertisement video by *SP*. Next, we consider another example related to the consent collection requirement. Let us assume that at 2:45 p.m. on January 2, 2022 the following log event is committed to *GCL*:

$\mathcal{L}(energy, E056) =$
{"***Write***": {
"***datatypeto***": "*energy*", "***ref_to***": "*DSt-Ref0045*",
"***ref_from***": "-", "***t***": "*02/01/2022 14:45*",
"***who***": "*SP*", "***ID***": "*E056*"}
}

This log event captures the collection of a piece of meter reading (i.e., data type *energy*). When *GA* invokes the compliance check the next day at 11:10 a.m. (i.e., on January 3, 2022), since, there is no log event *WriteConsent* for *energy* in *DSt*, a violation is detected based on Algorithm 1. Finally, regarding the *data deletion*, as pointed out in the *SP* policy before, the contract period is 3 years and data must be deleted from *DSt* within 6 years based on π_{del}. Violation of the privacy policy can happen if after the log event $\mathcal{L}(contact, E145)$ above, the *GCL* contains any log event such as:

$\mathcal{L}(contact, E333) =$
{"***Write***":
{
"***datatypeto***": "ND", "***ref_to***": "*DSt-Ref1234*",
"***ref_from***": "-", "***t***": "*time*",
"***who***": "*SP*", "***ID***": "*E333*",
}
}

where *time* is beyond *01/03/2028#13:00*. In this case, the contact data stored at the location $DSt - Ref1234$ will be automatically deleted by *DSt*, which takes the role of the system identity *SYS*.

7.3.4 | Accountability and traceability

During the encryption and delegating access operations, the use of entity's ID E_{ID} and *KFN* allows tracking of users (i.e., *DSs*) and system entities' activities. For instance, if a *DS*, who owns a private key SK_{AS}, is suspected of being dishonest and/or colluding with an entity *E*, the $Trace(SK_{AS})$ function is used to check if the *ID* matches E_{ID}. In case of granting access via PRE scheme, the $Trace(SK_{AS})$ functions also checks if *KFN* is equal to that of the entity who owns E_{ID}.

7.3.5 | Efficiency analysis

In this section, we focus our efficiency analysis on the compliance checks overhead, and *CBN* performance in terms of average latency and throughput as explained in Section 7.2. Moreover, we include a comparison with current schemes in the literature that consider GDPR compliance. The comparison with these schemes is based on the functionalities they offer and their features. The rationale behind these choices is due to the fact that most of the schemes in the literature, except for Reference 12, do not provide enough details of their implementation (e.g., policy language specification and/or compliance checks algorithms for *SC* implementation). Hence, we are unable to implement their schemes on the *CBN* and perform a fair comparison with ours.

Scope and limitations

The existing cryptographic schemes we used in our proposed system have already been benchmarked against their counterparts (e.g., in Reference 22). However, for the sake of completeness, we should mention that the operations time overhead in the proposed system is mainly related to authentication in terms of message signing and verification. It occurs only when the *DS* encrypts and submits his/her data to *DSt*. Using AES/CBC (256-bit key) as *SEnc*(.) with the processing speed of 455 MB/s and SHA-512 with the processing speed of 231 MB/s, it takes the *DS* approximately 0.02 ms to

encrypt 10 KB personal data file. These processing speeds are obtained on the same Ubuntu 20.04 server with Intel Xeon 6248 2.5 GHz mentioned above in Section 7.2. On the other hand, the time needed to perform the encryption operation $Enc(PK_{SP}, KW) \rightarrow (S_{PEKS})$ and $Enc(PK_{ABE}, S_r, AS) \rightarrow (CT_{ABE})$ are approximately 37 and 62 ms, respectively, where *AS* has four attributes. The measured time values stay relatively flat even when the number of attributes reaches 100 as shown in the performance analysis of CP-ABE with accountability in Reference 22, which we have utilized in our scheme. Finally, with their computational resources, the time needed for decryption by the *DSt* and *SP* are negligible.

Efficiency analysis of compliance checks

The initial compliance check against *SP′s* policy, which is defined in Section 7.1, takes on average 37 ms. Here, we only verify the GDPR requirements for consent collection, purpose minimization and consistency, data retention delay, and data access, which are related to the proposed sub-policy specifications. For the scenario above, the system generates four log files; one for each data type which contains approximately 33–45 numbers of *Writeconsent*, *Write*, *Send*, and *Read* events. After 30 runs, the compliance verification of the generated logs against the *SP′s* policy took on average 200 ms. Table 1 shows the average time it takes the *GA* to run its compliance checks against $\mathcal{L}(service)$ based on the transactions rate on *CBN*. It also shows the performance of the read operation on *CBN* (i.e., *getLogs*). As pointed out before, the transaction rate varies between 25 and 500 TPS for *getLogs* (i.e., when the *GA* retrieves these logs for continuous auditing and conformance checks against *SP′s* policy).

Scalability analysis

Table 2 shows the results for the write operation *recordLogs* (i.e., when *SP*, *TP*, and *DSt* interact with *CBN* to record their logs) for the same workload (i.e., send rates) in Table 1. Note that the average latency increases significantly for the write operations on *CBN* as the workload increases since new records are created on *CBN* hence, consensus must be reached first among peer nodes therefore, causing extra waiting time. Nonetheless, recording logs is not linked to data operations that are taking place regardless. Therefore, our proposed system is feasible since the compliance check of logs against the approved policy is not required to take place simultaneously (i.e., it takes place offline after the logs are recorded).

Finally, in a larger real-world environment where the full privacy policy is included along with larger number of system participants (e.g., more *TPs* and *DSs*), the following considerations should be taken into account. First, as data operations become very frequent and generate logs at high rate (e.g., more than 200 TPS), the compliance checks will be delayed. In this case, fine tuning of the periodical compliance checks, which are carried out when the main operations are done, and acceptable delays should be investigated. Second, since more peer nodes will join the *CBN*, this will have an impact on the average latency since consensus among peer nodes will take longer to be achieved. In this case, measures such

TABLE 1 Transactions latency, throughput and compliance check overhead on *CBN—getLogs*.

	Send rate	Average latency	Throughput	Compliance check
getLogs	25.2 TPS	20 ms	25.2 TPS	220 ms
	50.4 TPS	20 ms	50.4 TPS	220 ms
	200.8 TPS	40 ms	200.3 TPS	240 ms
	399.8 TPS	40 ms	398.4 TPS	240 ms
	497.5 TPS	40 ms	495.1 TPS	240 ms

TABLE 2 Transactions latency and throughput on *CBN—recordLogs*.

	Send rate	Average latency	Throughput
recordLogs	25.2 TPS	70 ms	25.2 TPS
	50.4 TPS	80 ms	50.3 TPS
	200.8 TPS	600 ms	200.6 TPS
	399.8 TPS	1170 ms	310.3 TPS
	497.5 TPS	1500 ms	317.1 TPS

TABLE 3 Comparison of features and functionalities.

Relevant work	Collection consent	Processing consent	Policy language	Logs	Conformance check	Number of SCs
Our scheme	Yes	Yes	Formal	Yes	Semi-automated	6
11	Yes	Yes	XACML	Unknown	Manual	Undetermined
12	Yes	Yes	Access control list	Unknown	Manual	2

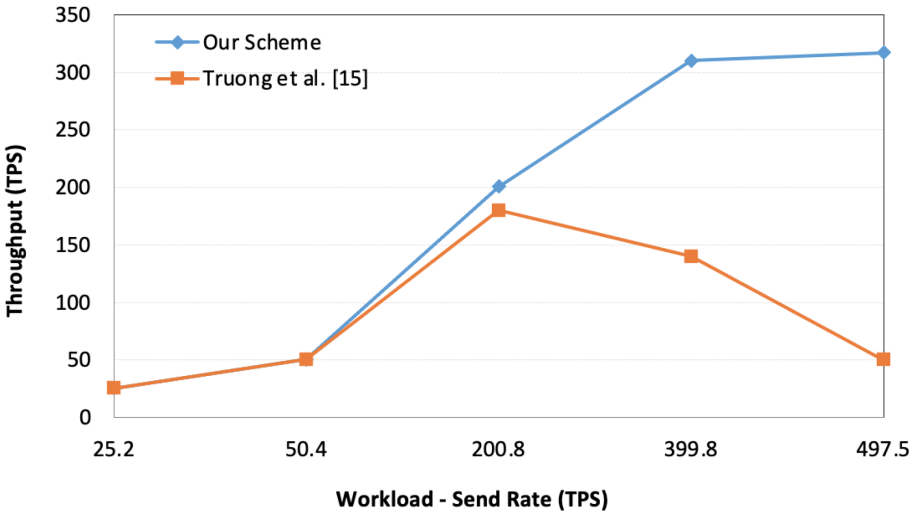

FIGURE 8 *recordLogs* performance: Throughput.

as different consensus thresholds should be considered to minimize that delay. Third, although the initial compliance check for the full privacy policy will take longer, it does not impact the system performance since it is done before the data lifecycle. These considerations are left for our future work.

Comparison with related works

In this section, we choose the works by Daudén-Esmel et al.[11] and Truong et al.[12] to perform this comparison given their proper documentation and similarity, in terms of the main objectives and functionalities, to this work. Table 3 compares the different features and functionalities offered by these works. It can be noted in Table 3 that all schemes differentiate between data collection and data processing consents. In terms of the utilized policy language, only our work uses a bespoke privacy policy language designed for fine-grained specification of data protection and privacy requirements. Our proposed policy language is defined on data types, and supports a systematic policy specification, as its syntax and semantics cover five sub-policies capturing a representative data life-cycle from collection till deletion.

In terms of generating logs during the data lifecycle, it is unclear how other schemes generate and use these logs for compliance checks. Therefore, the corresponding feature is left as unknown in Table 3. At least, we can confirm that our generated logs are used to perform conformance checks semi-automatically while for other schemes, it is done manually using the logs (i.e., records) on the ledger. A conformance check compares a policy and a log of events based on the privacy properties. Specifically, if an entity does not have the right to have certain type of data or link two types of data, then in the logs, during the data lifecycle, this entity cannot have or link those types of data.

Finally, our scheme defines six *SCc* each dedicated to a different stage of the full data lifecycle from collection, storage and transfer, to deletion. This ensures separate processing and logging of different events during the data lifecycle to check for compliance later. In the scheme proposed by Truong et al.,[12] there are only two SCc; one for authentication, authorization and access control, and another for access validation and logging. This design approach clearly underperformed in terms of average latency as shown in Figure 9 before. On the other hand, the number of *SCc* in the scheme proposed by Daudén-Esmel et al.[11] is undetermined since a new *SC* is created every time a new data processor requests

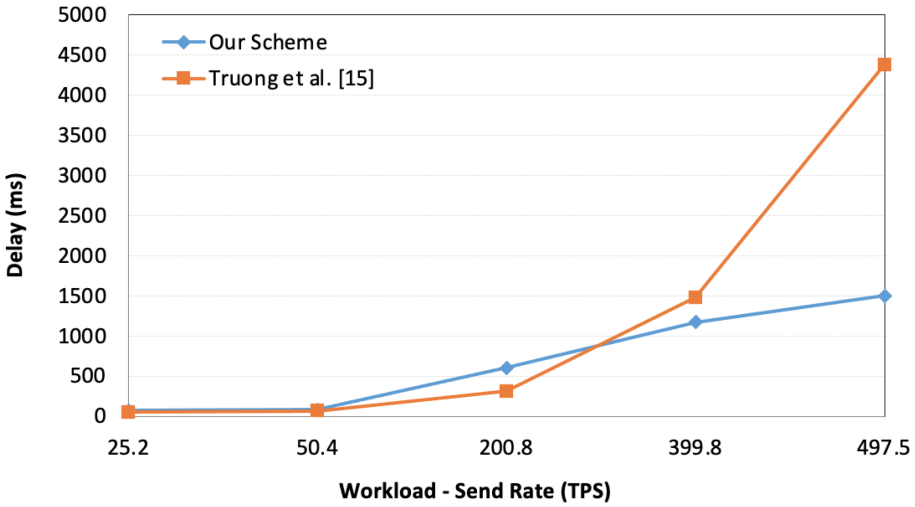

FIGURE 9 *recordLogs* performance: Delay.

to process some *DS*′*s* personal data collected by a data controller. This will result in many *SCs* that could result in high computing and processing overhead.

In terms of performance, we compare the overhead of the write operation only with that in the scheme proposed by Truong et al.[12] The reason for this is the importance of the write operation in these systems in comparison to the read operation, which does not cause delays since it does not require creating new blocks, reaching consensus, etc. Unlike the read operations, write operations represent a bottleneck where blocks are created and are put in a queue waiting to be verified. Figure 8 shows that our scheme outperforms that in Truong et al.[12] when the number of transactions increases beyond 200 TPS. In fact, our scheme throughput continues to have a success rate of over 63% when transactions reach 497.5. Thanks to the design of different *SCs* that handle different operations during the data lifecycle, we can keep the throughput relatively high even when the workload increases. Note that the scheme in Reference 12 only have two *SCc*, which are not clearly dedicated to the data lifecycle operations. This point will be explained further in the next sub-section in conjunction with Table 3.

The advantage of our design approach, where we have six *SCs* each dedicated to a different stage of the data lifecycle, can be seen clearly in Figure 9. The delay reaches a maximum of 1500 ms in our scheme when the workload reaches its maximum capacity of 497.5 TPS. In comparison, the scheme in Reference 12 reaches nearly 4500 ms for the same maximum workload.

8 | CONCLUSION AND OUTLOOK

In this article, we proposed a novel semi-automatic GDPR compliance service with restrictive fine-grained access control. The service is developed on top of a permissioned blockchain and uses smart contracts to enforce *SP*'s privacy policy compliance with GDPR before and during the full data lifecycle. Operations' logs on personal data are checked regularly to ensure users' data are handled according to the privacy policy, which is used to enforce access control on the data using attribute-based encryption with keyword search and data sharing. The proposed service was evaluated using a real world privacy policy of an energy *SP*. The analysis showed that our service achieved transparency and continuous GDPR compliance checks during the full data lifecycle with low compliance time overhead and high throughput.

For future work, we will extend the proposed policy language to cater to all GDPR requirements and develop a control panel for *SP*s to publish their privacy policies on the system. The control panel is envisaged to digest privacy policies and produce the format required by the system. This can be part of the A_{SP} application functionalities which *SPs* use to interact with *SCs*. It can be developed in isolation of the system and used to call *SCs*' functions via their APIs. On the other hand, from users' perspective, all the operations of CP-ABE are carried by the application provided by the *SP* for their service. Some operations such as PRE are outsourced to the storage server *DSt*, as mentioned in Section 4.5 to ensure users' experience is not affected.

DATA AVAILABILITY STATEMENT

Data sharing is not applicable to this article as no new data were created or analyzed in this study.

ENDNOTE

*OVO energy, https://sse.co.uk/about-us/legal/privacy-policy.

ORCID

Max Hashem Eiza https://orcid.org/0000-0001-9114-8577
Vinh Thong Ta https://orcid.org/0000-0003-0399-9633

REFERENCES

1. Sun J, Ren L, Wang S, Yao X. A blockchain-based framework for electronic medical records sharing with fine-grained access control. *PLoS One*. 2020;15(10):e0239946.
2. Guo Y, Lu Z, Ge H, Li J. Revocable blockchain-aided attribute-based encryption with escrow-free in cloud storage. *IEEE Trans Cloud Comput*. 2023;72(7):1901-1912.
3. Yan H, Li J, Han J, Zhang Y. A novel efficient remote data possession checking protocol in cloud storage. *IEEE Trans Inf Forensics Secur*. 2017;12(1):78-88.
4. Li J, Yan H, Zhang Y. Certificateless public integrity checking of group shared data on cloud storage. *IEEE Trans Serv Comput*. 2021;14(1):71-81.
5. Zhang R, Li J, Lu Y, Han J, Zhang Y. Key escrow-free attribute based encryption with user revocation. *Inform Sci*. 2022;600:59-72.
6. Makhdoom I, Zhou I, Abolhasan M, Lipman J, Ni W. PrivySharing: a blockchain-based framework for privacy-preserving and secure data sharing in smart cities. *Comput Secur*. 2020;88:101653.
7. Amato F, Cozzolino G, Moscato F, Moscato V, Xhafa F. A model for verification and validation of law compliance of smart contracts in IoT environment. *IEEE Trans Industr Inform*. 2021;17(11):7752-7759.
8. Javed IT, Alharbi F, Margaria T, Crespi N, Qureshi KN. PETchain: a blockchain-based privacy enhancing technology. *IEEE Access*. 2021;9:41129-41143.
9. Wang Y, Su Z, Zhang N, et al. SPDS: a secure and auditable private data sharing scheme for smart grid based on blockchain. *IEEE Trans Industr Inform*. 2021;17(11):7688-7699.
10. Heiss J, Ulbricht M-R, Eberhardt J. Put your money where your mouth is—towards blockchain-based consent violation detection. *IEEE International Conference on Blockchain and Cryptocurrency (ICBC), Toronto, ON*. IEEE; 2020.
11. Daudén-Esmel C, Castellà-Roca J, Viejo A. Blockchain-based access control system for efficient and GDPR-compliant personal data management. *Comput Commun*. 2024;214:67-87.
12. Truong NB, Sun K, Lee GM, Guo Y. GDPR-compliant personal data management: a blockchain-based solution. *IEEE Trans Inf Forensics Secur*. 2019;15:1746-1761.
13. Dauden-Esmel C, Castella-Roca J, Viejo A, Domingo-Ferrer J. Lightweight blockchain-based platform for GDPR-compliant personal data management. *2021 IEEE 5th International Conference on Cryptography, Security and Privacy (CSP)*. IEEE; 2021:68-73.
14. BC_GDPR-Compliant_PDManagement_System. Accessed November 8, 2021. https://github.com/toful/BC_GDPR-Compliant_PDManagement_System
15. Thong Ta V, Hashem Eiza M. DataProVe: fully automated conformance verification between data protection policies and system architectures. *Proc Priv Enhanc Technol*. 2022;2022(1):565-585.
16. Hu B, Zhang Z, Liu J, et al. A comprehensive survey on smart contract construction and execution: paradigms, tools, and systems. *Patterns*. 2021;2(2):100179.
17. Hyperledger Fabric. 2020. Accessed November 13, 2021. https://www.hyperledger.org/wp-content/uploads/2020/03/hyperledger_fabric_whitepaper.pdf
18. Hearn M, Brown RG. Corda: a distributed ledger. August 2019. Accessed November 13, 2021. https://www.r3.com/wp-content/uploads/2019/08/corda-technical-whitepaper-August-29-2019.pdf
19. Androulaki E, Barger A, Bortnikov V, et al. Hyperledger fabric: a distributed operating system for permissioned blockchains. *Proceedings of the Thirteenth EuroSys Conference*. ACM; 2018:1-15.
20. Boneh D, Crescenzo GD, Ostrovsky R, Persiano G. Public key encryption with keyword search. *Advances in Cryptology-EUROCRYPT*. Springer-Verlag; 2004.
21. Bethencourt J, Sahai A, Waters B. Ciphertext-policy attribute based encryption. *2007 IEEE Symposium on Security and Privacy*. IEEE; 2007:300-314.
22. Li J, Zhang Y, Ning J, Huang X, Sen Poh G, Wang D. Attribute based encryption with privacy protection and accountability for CloudIoT. *IEEE Trans Cloud Comput*. 2022;10(2):762-773.
23. Ge C, Susilo W, Liu Z, Xia J, Szalachowski P, Fang L. Secure keyword search and data sharing mechanism for cloud computing. *IEEE Trans Dependable Secure Comput*. 2021;18(6):2787-2800.
24. Li J, Shi Y, Zhang Y. Searchable ciphertext-policy attribute-based encryption with revocation in cloud storage. *Int J Commun Syst*. 2017;30(1):e2942.
25. Dataprove. https://github.com/Dataprove/Dataprovetool/

26. Crypto++ Library 8.7, Crypto++ community. Accessed May 13, 2023. https://www.cryptopp.com
27. Bethencourt J, Sahai A, Waters B. Advanced crypto software collection—ciphertext-policy attribute-based encryption. Accessed May 13, 2023. https://acsc.cs.utexas.edu/cpabe/
28. Hyperledger Caliper, Fabric v0.5.0. Accessed May 13, 2023. https://hyperledger.github.io/caliper/v0.5.0//fabric-config/new/

How to cite this article: Hashem Eiza M, Thong Ta V, Shi Q, Cao Y. Secure semi-automated GDPR compliance service with restrictive fine-grained access control. *Security and Privacy*. 2024;e451. doi: 10.1002/spy2.451