GDPR-Compliant Personal Data Management: A Blockchain-Based Solution

Nguyen Binh Truong, *Member, IEEE*, Kai Sun, *Senior Member, IEEE*, Gyu Myoung Lee, *Senior Member, IEEE*, and Yike Guo, *Fellow, IEEE*

***Abstract*— The General Data Protection Regulation (GDPR) gives control of personal data back to the owners by appointing higher requirements and obligations on service providers who manage and process personal data. As the verification of GDPR-compliance, handled by a supervisory authority, is irregularly conducted; it is challenging to be certified that a service provider has been continuously adhering to the GDPR. Furthermore, it is beyond the data owner's capability to perceive whether a service provider complies with the GDPR and effectively protects her personal data. This motivates us to envision a design concept for developing a GDPR-compliant personal data management platform leveraging the emerging blockchain and smart contract technologies. The goals of the platform are to provide decentralised mechanisms to both service providers and data owners for processing personal data; meanwhile, empower data provenance and transparency by leveraging advanced features of the blockchain technology. The platform enables data owners to impose data usage consent, ensures only designated parties can process personal data, and logs all data activities in an immutable distributed ledger using smart contract and cryptography techniques. By honestly participating in the platform, a service provider can be endorsed by the blockchain network that it is fully GDPR-compliant; otherwise, any violation is immutably recorded and is easily figured out by associated parties. We then demonstrate the feasibility and efficiency of the proposed design concept by developing a profile management platform implemented on top of the Hyperledger Fabric permissioned blockchain framework, following by valuable analysis and discussion.**

***Index Terms*— Blockchain, data management, GDPR, personal data, smart contract.**

I. Introduction

THE General Data Protection Regulation (GDPR) legislation came into force in May 2018 in all European Union (EU) countries. The GDPR is a major update to the data privacy regulations released in 1995, which is before the proliferation of cloud platforms and social media, let alone the scale of today's data usage. The provision of the GDPR is to ensure that personal data "can only be gathered legally, under strict conditions, for a legitimate purpose"; as well as to bring full control back to the data owners.[1]

As the GDPR requirements are highly abstract, it is open to interpretation. In fact, each organisation has its own way to satisfy the new regulations; and to demonstrate the compliance. Supposedly, each EU member state provides a supervisory authority who is responsible for monitoring the GDPR-compliance. Organisations are required to demonstrate compliance only in case of suspicion of a violation or when a Data Subject (i.e., the owner of data, denoted as DS) lodges a complaint with the supervisory authority. In this regard, the challenge of complying with the GDPR is not because of lacking technical solutions for tackling down the GDPR requirements nor providing required mechanisms; it is because such solutions are designed and implemented under a centralised client-server architecture mindset. Due to the irregular verification of GDPR compliance, critical concerns on the lack of transparency have been imposed accordingly. In particular, it is unachievable for a Service Provider (SP) to prove that it has been continuously adhering to the GDPR using existing centralised solutions. Moreover, it is beyond the DS's capability to perceive whether an SP fully complies with the GDPR and effectively protects her data. For these reasons, GDPR-compliant personal data management is a well-suited scenario for the emerging blockchain technology (BC) to come into play. A BC platform implementing Smart Contracts (SCs) is expected to be a promising measure for these challenges thanks to its advanced features of decentralisation, transparency, tamper-resistance, and traceability.

Some research articles have stated potentials of the BC as a general-purpose data management and storage [1]–[11]; however, they only provided preliminary methodological exploration or conceptual models without detailed technical analysis and implementation. In these articles, a holistic architecture of decoupling the BC, which is for accounting and auditing data access, from a storage layer, which physically stores data were adopted. Unfortunately, there are lacking of a comprehensive design concept and technical mechanisms to actualise the capability of the BC in personal data management and in complying with the GDPR requirements. In this article, we propose a design concept with technical mechanisms for a BC-based GDPR-compliant personal data management platform, along with a detailed implementation of the profile management

Manuscript received March 19, 2019; revised July 26, 2019 and October 3, 2019; accepted October 10, 2019. Date of publication October 18, 2019; date of current version January 22, 2020. This work was supported by the HNA Research Centre for Future Data Ecosystems at Imperial College London. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Mauro Conti. *(Corresponding author: Nguyen Binh Truong.)*

N. B. Truong, K. Sun, and Y. Guo are with the Department of Computing, Data Science Institute, Imperial College London, London SW7 2AZ, U.K. (e-mail: n.truong@imperial.ac.uk; k.sun@imperial.ac.uk; y.guo@imperial.ac.uk).

G. M. Lee is with the Department of Computer Science, Liverpool John Moores University, Liverpool L3 3AF, U.K. (e-mail:g.m.lee@ljmu.ac.uk).

Digital Object Identifier 10.1109/TIFS.2019.2948287

[1] https://gdpr-info.eu/

system use-case built on top of a permissioned BC framework. The goal of the design concept is to preserve advanced features of BC and SCs in personal data management by leveraging distributed ledger and public-key cryptography technologies for complying with the manifold legal requirements of the GDPR [12]. For this purpose, a BC network is designed to play as the roles of: *(i)* a delegated authentication and authorisation server which is consolidated by a novel concept of decentralised *access token*, *(ii)* an automated access control manager, and *(iii)* an immutable logging system a for parties who desire to access personal data stored in an off-chain Resource Server (RS).

By following the proposed design concept, a personal data management platform ensures that only designated DSs and Data Controllers (DCs) are permitted to create, update and withdraw consents; and only authorised Data Processors (DPs) can process personal data respecting rules defined in corresponding data usage policy agreed between the DSs and the DPs. The platform not only provides mechanisms for DS rights but also plays as a role of a DC for handling personal data processing and demonstrating data accountability. By honestly participating in the BC-based personal data management platform, an SP can be endorsed by the BC network that it is GDPR-compliant. Otherwise any violations are recorded in an immutable distributed ledger as a record of the infringements, which can be then used for the GDPR compliance investigation by supervisory authorities.

We demonstrate the feasibility and effectiveness of the proposed design concept by developing a system for managing personal profiles. The system, which is built on top of the Hyperledger Fabric (HLF) permissioned BC framework[2] and cooperates with an honest RS for data storage, plays as a profile management service for a social networking SP. This system provides clients' rights as well as facilitates the social networking SP's obligations, following by analysis and discussion on the GDPR-compliance, threat models and system performance. It is affirmed that the social networking SP is fully compliant with the GDPR requirements. We believe the proposed approach is a promising solution not only for GDPR-compliant personal data management but also for digital assets governance.

The rest of the article is organised as follows. Section II presents background and related work. Section III describes challenges and motivation. The design concept is proposed in Section IV following by the implementation of the profile management platform in Section V. Section VI provides the analysis and discussion about the platform. The last section concludes our work and outlines future research.

II. Background and Related Work

In this section, relevant background knowledge on GDPR and BC and related work are presented. Table I depicts some of the notions frequently used throughout this article.

A. The GDPR in a Nutshell

The full GDPR are described in detail across 99 articles covering all of the technical and admin principles around how commercial and public organisations process personal data [13]. GDPR lays out the means by which personal data is to be protected which are founded on a set of six core data processing principles: Lawfulness, Fairness, and Transparency; Purpose Limitation; Data Minimisation; Accuracy; Storage Limitation; Integrity and Confidentiality.[3] To preserve such principles, the GDPR clearly differentiates three roles (i.e., DS, DC and DP) and explicitly specifies associated rights and obligations under the EU data protection law. The goal of the GDPR legislation is to provide a DS full control over her personal data by specifying a variety of rights. The GDPR requires that personal data should be managed by a DC that assures the rights of the DS [13]. Such mechanisms enable the DS to impose consents and to arbitrarily withdraw the consents whenever needed. The DS is also able to trace back all activities on her data including who, what, why, when, and how the data is processed. Valid legal consents must be given by the DS to the DC for processing her personal data. The DC then takes appropriate measures to provide the rights of the DS; meanwhile determines the purposes for which and the method in which, the personal data is processed by DPs [14].

Being compliant with the GDPR is not enough, DCs should also be able to demonstrate the compliance to supervisory authorities once required (when a supervisory authority has suspicion of a violation or when a DS lodges a complaint with the supervisory authority). In this case, the supervisory authority shall establish and make public a list of processing operations subjected to Data Protection Impact Assessment and the Privacy Impact Assessment requirements[4]; then file a report of infringements if it is the case.

TABLE I
NOTATION TABLE WITH ENTRIES IN ALPHABETICAL ORDER

Notation	Description
API	Application Programming Interface
BC	Blockchain
BFT	Byzantine Fault Tolerance
C-ID	Complex Identity
CA	Certificate Authority
CRUD	Create-Read-Update-Delete operations
DBMS	Database Management System
DC	Data Controller
DP	Data Processor
DS	Data Subject
GDPR	General Data Protection Regulations
HLF	Hyperledger Fabric Blockchain framework
IdM	Identity Management
MSP	Membership Service Provider
OSN	Ordering Service Node
RS	Resource Server
SC	Smart Contract
SP	Service Provider
TP	Third-Party

B. Blockchain Technology

The BC technology, indeed, is a set of diversified techniques including distributed systems, computer networks, databases, and cryptography playing as the role of a distributed ledger.

[2]https://www.hyperledger.org/projects/fabric

[3]https://gdpr-info.eu/art-5-gdpr/

[4]https://gdpr-info.eu/issues/privacy-impact-assessment/

The BC technology maintains a distributed immutable database constituted from a continuous growing list of blocks so-called a BC which records all transactions between entities in a network. In our article, the acronym *BC* either refers to the technology or a specific chain-of-block database. By nature, a BC is inherently resistant to data modification. Once recorded, information in any given block cannot be altered retroactively as this would invalidate all hashes in the previous blocks in a BC; and break the consensus among nodes in the network. The concept of BC was introduced in Bitcoin in 2008 [15]. Bitcoin is the first cryptocurrency that not only transacts digital currency in a secure manner but also resolves the long-standing problem of "double-spend" without the need for a trusted third-party. BC underpins Bitcoin, but BC is not only Bitcoin. Its usage goes far beyond [16]–[18].

In a BC network, a consensus protocol needs to be implemented to ensure any disruptive action from an adversary will be negated by a majority of participants [2]. The protocol is to decide which player among the participants in the BC network has permission to append a new block; other participants are able to verify the permission and update their local ledgers accordingly; which establishes consensus over the network [19], [20]. Proof of Work (PoW) is the most common consensus model used in public BCs. Unfortunately, PoW is computation-intensive, as it requires powerful nodes (i.e., miners) dedicate to solve a computationally intensive puzzle (i.e., mining), in order to produce a new block to the chain [21]. To overcome latency and throughput bottlenecks of PoW, alternative consensus models have been proposed, including Proof of Stake (PoS) [22], [23], Byzantine fault-tolerant (BFT) variants [24], Proof of Elapsed Time (PoET),[5] and Algorand [25]. Nonetheless, such consensus protocols depend on several assumptions and impose their own disadvantages which results in limited usage in the real-world compared to the PoW-variant mechanisms [20].

C. Smart Contracts

An SC is a computer program deployed onto a BC network. It automatically executes "actions" when necessary "conditions" are met, specifying business logic of a service that participants have agreed to [26]. As a mutual agreement, the content of the SC is accessible to all participants [27]. An SC is a form of decentralised automation that facilitates, verifies, and enforces an agreement in a transaction and records the results (i.e., state changes) into a ledger. All BC frameworks have built-in mechanisms for executing SCs from a simple stack-based scripting system (e.g., Bitcoin) to a Turing-complete system (e.g., Ethereum and Hyperledger). Ethereum is among the first BCs offering Turing-completeness. Its SCs are written in either Solidity, Serpent or LLC, before being compiled to bytecodes and executed in an Ethereum Virtual Machine (EVM) [28]. The EVM keeps track of resources consumed by the execution (i.e., *gas*) and charges to the sender's account as an incentive for miners. Hyperledger does not have its bytecode for SCs. Instead, its SCs are language-agnostic programs which are then compiled into native code, packed, installed and executed inside Docker containers [29]. As a result, this language-agnostic design supports multiple high-level programming languages such as Go and JavaScript [30].

D. Related Work

Besides cryptocurrencies, the use of BC in other areas has been intensively carried out over the last few years. Specifically, prominent features of BC such as immutability, traceability, transparency, and pseudo-anonymity can be preserved for a wide range of decentralised applications (DApps), especially for managing and accounting digital assets. For instance, several projects have utilised BC in supply-chain and logistics to provide provenance tracking mechanisms for products leveraging its immutability and traceability features [31]–[33]. The immutability and transparency features have also been utilised in a cloud data provenance platform called ProvChain [34] in which all data operation history was transparently and permanently recorded into a BC.

Furthermore, SCs deployed in a BC framework provide autonomous functionalities executed in a decentralised manner for a wide range of domain services. Blockstack [35] took advantage of BC for managing domain names to replace the traditional centralised Domain Name System. This work introduced pivotal functionalities including identity and discovery mechanisms deployed on top of the Namecoin platform [36] and integrated with an off-chain storage service. In Blockstack, domain name registration and modification operations were implemented in BC whereas payload and digital signatures were stored in a Kademlia[6] Distributed Hash Table (DHT), which was connected to a virtual-chain that separated off-chain storage and BC operations. Only hashes of "name-data" tuples and state transitions were recorded on-chain. This design of decoupling the storage layer from the BC has paved the way to other studies, particularly in large-scale Internet of Things (IoT) data management [1], [2]. In these studies, data generated from IoT devices were stored in a DHT system and only keys of the data were recorded onto a BC. DHT nodes, responsible for managing IoT data, are required to join the BC network and listen to transactions for sending/retrieving data to/from legitimate IoT devices. BigchainDB [37] further provided a mechanism to balance between on-chain and off-chain storage to achieve advanced features from both BC and distributed databases by using Tendermint,[7] a weak synchronisation BC engine built on a BFT consensus.

Besides general-purpose data storage, BC-based accounting and management mechanisms (e.g., IdM, authorisation, access and permissions control) have also been proposed in a variety of scenarios. Lee proposed a BC-based cloud ID service for IdM [38], which used public-key cryptography for pseudo-identity and a distributed ledger for recording public keys. This study introduced a concept of mutual authentication by combining signatures from a client and an SP for granting access to a service. A fast security authentication scheme based on permissioned BC was proposed by Chen *et al.*

[5] https://sawtooth.hyperledger.org/docs/core/releases/latest/index.html

[6] https://en.wikipedia.org/wiki/Kademlia

[7] https://tendermint.com

in a 5G ultra-dense network [39] by using an optimised Practical BFT (PBFT) consensus protocol called APG-PBFT. APG-PBFT propagated authentication results embedded in BC among a group of access points, resulting in reducing the authentication frequency. In [40], a distributed access control in the IoT was proposed, with operations embedded in an SC on a public BC (i.e., Ethereum). However, most of these studies only presented high-level system design, without technical details to demonstrate the feasibility of their proposed solutions. Some platforms (e.g., [40]) relied on a set of management nodes to play as a hub for access control, which in fact turns into the scenario of centralised management.

A few studies in the literature concerning BC-based personal data management, particularly in supporting SPs to comply with the new GDPR legislation. In [3], Wang *et al.* proposed a fine-grained access control scheme deployed in the Ethereum framework, for personal files stored in a distributed file system called Interplanetary File System (IPFS) [4]. It customised an attributed-based encryption scheme, but the dependency of a centralised trusted private key generator is eliminated by leveraging BC. The main limitation of this system is data owners were responsible for all required tasks, from secret key generation, file encryption, to the establishment of a secure channel for communicating with another party. The Ethereum framework was just used as a medium to execute SCs in which crypto-artifacts were embedded for identity authentication. Zyskind *et al.* [5] proposed another access control scheme for a privacy-preserving personal data sharing platform, taking advantage of immutability and public-key cryptography in BC for identity verification and authorisation mechanisms. Similar ideas were proposed for Electronic Heath Records (EHRs) access control using Ethereum [6], [7] or a permissioned BC [8]. In these works, EHRs were stored off-chain in secure data custodians whereas access control was carried out on a BC using a digital signature scheme. Neisse *et al.* [9] proposed a BC-based approach for data accountability, resulting in GDPR-compliance. They discussed different design choices respecting who create and manage data usage SCs. Similar ideas can be found in [10], [11]. However, in these studies, only the conceptual approach was presented; technical details on platform development were missed out. The challenges including ledger data models and functionalities in SCs have not been addressed.

III. Personal Data Management: Scenarios and Challenges

In this section, we provide an overview of the scenarios and the current solution approach on personal data management which leverages a delegated authentication and authorisation server following the OAuth standardisation [41] (illustrated in Fig. 1). This solution approach is designed under a centralised client-server mindset that imposes unsolvable challenges in complying with the new GDPR requirements and in establishing trust with clients [14].

A. Scenarios

We consider real-world scenarios in which clients allow an SP to collect, manage, process, and (possibly) shares their personal data in exchange for a service. These scenarios specify three roles as follows:

- *End-user*: a client of a service who owns personal data. The end-user allows the SP to collect its data once using the service. In the GDPR terminology, an end-user is a DS.
- *Service Provider (SP)*: an entity that directly collects and manages personal data for its operational and business-related purposes. An SP stores personal data in an RS, which is either a system run by the SP or an independent service. An SP may share collected data with third parties for its benefits. In the context of GDPR, an SP plays both roles of a DC (when the SP shares personal data with a third-party) and a dDP (when the SP processes personal data for its own business).
- *Third-party (TP)*: an entity that provides a service to end-users but has to rely on the SP' infrastructure to develop the service and to acquire desired personal data. In the GDPR terminology, a TP is a DP.

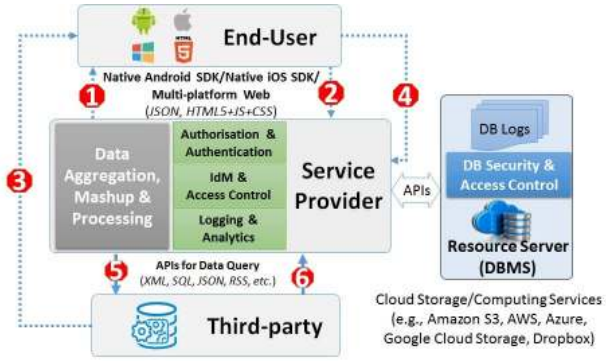

Fig. 1. Personal data management and sharing scheme in the conventional client-server architecture.

As illustrated in Fig. 1, the procedure of granting data access for an SP and a TP is in four steps:

1) A user starts to use a service provided by an SP. The SP asks the user for permission to collect her personal data.
2) The end-user grants a set of permissions to the SP for personal data collection and processing.
3) The TP asks the end-user to access her personal data which is collected and managed by the SP.
4) End-user logs into the service provided by the SP and consents a set of permissions to the TP

Once the permission is granted, the data access procedure is in the fifth and sixth steps in Fig. 1:

5) The SP authenticates and authorises the TP for accessing the data and provides an access token to the TP.
6) The TP then calls associated APIs using the provided token in step-5 to obtain the desired data.

B. Challenges

To meet the new GDPR requirements, conventional solutions on personal data management provide additional measures such as offering end-users mechanisms to fully control their data. Nevertheless, these measures are based

on the client-server architecture which provide limited transparency and are lack of trust. For instance, a majority of SPs follow the $OAuth2$[8] standard for access delegation, which includes IdM, authentication, authorisation, and access control mechanisms that allows end-users to share their personal data with single sign-on in a simplified and secure manner [41]. However, the centralisation of the current approaches poses severe concern [42]: it fully relies on the truthfulness of the SP (i.e., a delegated authentication and authorisation server) as it is the only authority to *(i)* authenticate and authorise participants; and *(ii)* control data access and provenance.

From an end user's perspective, this leads to a lack of transparency and accountability of data management and raise risks of personal data leakage. As all data management mechanisms are operated in a centralised system and under the SP's control, the SP may still be able to hand over personal data to an unauthorised TP without the end-user's knowledge, as far as it is not investigated by supervisory authorities. From an SP's perspective, as investigation from supervisory authority is occasionally carried out, it is challenging for an SP to declare that it has been continuously, securely and legally processing all personal data as required. This is of paramount importance for any SP to build trust with prospective clients. Furthermore, delegated permissions on personal data are not flexible as end-users do not have a fine-granular access control to impose their preferences on data usage except simple conditions predefined by SPs. Indeed, SPs currently provide only options to either "accept all" or opt-out.

Motivated by such challenges, our ultimate goal is to develop a GDPR-compliant personal data management platform by leveraging the state-of-the-art BC and SC technologies. The use of BC with SC provides autonomous operations securely executed in a decentralised manner. Furthermore, the prominent features of the BC technology, namely immutability, traceability, transparency, and pseudo-anonymity, can be effectively utilised to manage personal data fully complying with the GDPR legislation.

IV. Design Concept

In this section, we propose a design concept for a GDPR-compliant personal data management platform, including a high-level system architecture, design guidelines, and detailed functionalities and algorithms.

A. Conceptual Model and System Architecture

1) Assumption: The design of a BC-based platform depends on the security models of the parties involved. In this article, we assume that an RS is "honest-but-curious" whereas SPs follow a malicious model. This means the RS executes required protocols honestly, even though it might be curious about the results it receives after the operations. If an SP correctly follows the required protocols; it will be compliant with the GDPR; otherwise violations will be logged in an immutable ledger as a record of GDPR infringements.

[8]https://oauth.net/2/

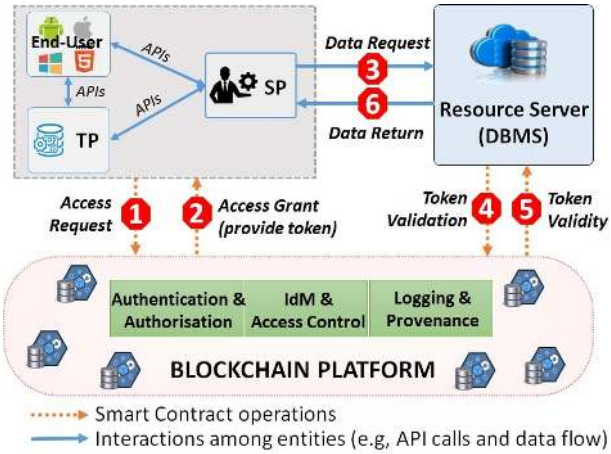

Fig. 2. High-level system architecture of the design concept for a BC-based personal data management platform. The operation flow consists of 6 steps, among which step 1, 2, 4, and 5 are dedicated to granting and validating permissions operated through Smart Contracts. Step 3 and 6 operated via API calls and data-flow from/to an resource server.

2) High-Level System Architecture: A conceptual model of the proposed platform is illustrated in Fig. 2. The inclusive idea is that mechanisms which are related to GDPR compliance are ported to a BC network from a traditional centralised server. In particular, the Authorisation and Authentication, IdM and Access Control; and Logging and Provenance components are implemented in the form of SCs deployed in a BC network. If a BC framework offers Turing-completeness (e.g., Ethereum and Hyperledger Fabric), GDPR-related mechanisms can be conveyed by SCs. As depicted in Fig. 2, all activities on personal data are authenticated and authorised by the proposed BC platform (step 1 and 2). The BC, playing as a role of a delegated authentication and authorisation server, issues an *access token* as "proof of permission" showing that a party has been granted to access a particular dataset. An authorised SP receives the *access token* (step 2) and use it to request desired data from the RS (step 3). The RS interacts with the BC platform to validate the granted access (step 4 and 5) before returns the requested data (step 6). The validation ensures the granted access is still valid and honestly used by the corresponding authorised party.

B. Design Guidelines

1) IdM, Authentication and Authorisation Mechanisms: IdM, authorisation, and authentication mechanisms are of paramount importance in any data management system since they are directly related to security and privacy of the system. In the design concept, an entity in a BC network should be uniquely identified using a public-key (or hash of the public-key) in an asymmetric cryptography key-pair; authentication and authorisation processes should be implemented leveraging public-key cryptography techniques (e.g., digital signatures and encryption). In the case of permissioned BC, an additional access control layer is consolidated by using a Certificate Authority (CA) and a Membership Service Provider (MSP).

2) Design of Distributed Ledgers: Content of a distributed ledger reflects historical and current states of information

recorded in the ledger maintained by the BC network. A personal data management platform should clarify what information and associated data model to be stored in the ledger.

(i) Information required to be tamper-resistant, transparent and traceable should be recorded in a distributed ledger.
- Any personal dataset should be specified by both DS and DC using digital signatures in a distributed ledger;
- Data Usage Policy should be clearly specified and recorded in a distributed ledger;
- Data activities should be logged in a distributed ledger. The logs should contain information about 'who', 'why', 'when', 'what' and 'how' personal data was processed;
- Hash of personal data can be recorded in a distributed ledger for data integrity checking.

(ii) The design of a distributed ledger must ensure:
- Designated nodes in the BC network are able to verify whether an entity is the DS or the DC of a dataset;
- Designated nodes in the BC network should be able to verify whether an entity's activity satisfies the data usage policy as recorded in a distributed ledger

3) Data Usage Policy: The policy specifies data governance measures including rights, permissions, and conditions. The usage policy should be defined in a fine-grain and expressive way using a policy language such as eXtensible Access Control Markup Language (XACML) and Model-based Security Toolkit (SecKit) designated for the IoT domains [43]. By nature, a blockchain-based personal data management following the proposed design concept provides a fine-grained access control capability as an individual user is able to customise her own policy on each dataset by imposing access control preferences recorded onto the ledger.

4) Off-Chain Data Storage: Personal data should be stored off-chain for better scalability and higher efficiency. Moreover, storing personal data directly onto BC, even in an encrypted form, could pose potential privacy leakage and result in non-compliance with the GDPR [44]. Depending on specific scenarios, a conventional DBMS (e.g., Oracle or MongoDB), a storage cloud service (e.g., S3, AWS or Azure), or a distributed storage system (e.g., IPFS [4] or Storj [45]) can be used for data storage. Only reference to the data is stored on-chain (i.e., stored in distributed ledgers). The reference is called $data\_pointer$ that can be a hash,[9] a connection string, an absolute path, or an identifier referring to a dataset; depending on specific off-chain storage system used in the platform.

C. Functionalities, Ledgers Data Model and Algorithms

1) Identity Management: We introduce *complex-identity*, denoted as *c-ID*, to specify a digital asset associated with two or more parties. A *c-ID* can be considered as an extension of asymmetric keys. In the context of the personal data management, a *c-ID* of a dataset m comprises an asymmetric key pair of the DS, an asymmetric key pair the DC, and an asymmetric key pair of the data pointer (denoted as p_m) of m. As the data usage policy depends on the requester's role (i.e., DS, DC, or DP), the way we define *c-ID* specifies the entities associated with m, and simplifies the process of verification. Any digital signature scheme such as Digital Signature Algorithm (DSA) or Elliptic Curve Digital Signature Algorithm (ECDSA)[10] can be used to generate and manage the *c-ID*, which is formally defined as a triple of probabilistic polynomial-time algorithms $(\mathcal{G}, \mathcal{S}, \mathcal{V})$:

- $\mathcal{G}$: a key generator that creates a public-private key pair (pk, sk).
- $\mathcal{S}$: a signing algorithm that takes sk and a message x as inputs and produces a signature $t = S(sk, x)$ as the output.
- $\mathcal{V}$: a signature verifying algorithm that takes pk, x, t as inputs, and outputs $accept$ or $reject$. For all x and (pk, sk), $V(pk, x, S(sk, x)) = accept$.

A *complete c-ID* is defined as a 6-tuple as follows:

$$c\text{-}ID^{comp}_{DS,DC} = (pk_{DS}, sk_{DS}, pk_{DC}, sk_{DC}, pk_{enc}, sk_{enc}) \tag{1}$$

where (pk_{DS}, sk_{DS}), (pk_{DC}, sk_{DC}) and (pk_{enc}, sk_{enc}) are asymmetric key-pairs of DS, DC and p_m, respectively. The *c-ID* is externally observed by nodes in a BC network as a 3-tuple:

$$c - ID^{ext}_{DS,DC} = (pk_{DS}, pk_{DC}, pk_{enc}) \tag{2}$$

The *c-ID* is observed by the DS (or DC) as a 5-tuple:

$$c - ID^{DS}_{DS,DC} = (pk_{DS}, sk_{DS}, pk_{DC}, pk_{enc}, sk_{enc}) \tag{3}$$

$$c - ID^{DC}_{DS,DC} = (pk_{DS}, pk_{DC}, sk_{DC}, pk_{enc}, sk_{enc}) \tag{4}$$

When a DS grants consent to a DP to access m, the private key sk_enc of p_m is shared to the DP through a secure channel. The DP then observes the *c-ID* as a 4-tuple:

$$c - ID^{DP}_{DS,DC} = (pk_{DS}, pk_{DC}, pk_{enc}, sk_{enc}) \tag{5}$$

The $c - ID^{DP}_{DS,DC}$ includes the key-pair (pk_{enc}, sk_{enc}) used to encrypt and decrypt sensitive information, including the data pointer p_m. Thus, only designated nodes are able to decrypt the ciphertext using the shared private key sk_{enc}. As a result, the information is protected from all other players in the system. Normally, RSA (Rivest-Shamir-Adleman) is used for public-key encryption mechanisms in a digital signature scheme such as DSA and ECDSA, formally defined as a 4-tuple $(\mathcal{G}, \mathcal{D}, \mathcal{E}, \mathbb{D})$: the key generator, key distribution, encryption and decryption mechanisms, respectively.

2) Distributed Ledgers Data Model: In the proposed design concept, ledgers are in the form of key-value pair, which is widely used in BC frameworks including Ethereum and HLF. For complex business logic, extra tasks might be required for mapping high-level data structures into key-value pairs. A state is a snapshot of a ledger at a specific time whereas state transitions are a result of transactions for creating, updating or deleting key-value pairs. A ledger contains a full history of state transitions recorded in a BC, thus it is timestamp-sequenced, immutable and tamper-resistant. With the key-value data format, all information can be obtained by

[9]Hash is a type of the $data\_pointer$ used in a content-addressed storage system such as DHT, IPFS, and Stoij.

[10]https://en.wikipedia.org/wiki/Elliptic_Curve_Digital_Signature_Algorithm

```
{"3A_ledger": {
    "key": {
      "owner": pk_DS,
      "controller": pk_DC
    }
    "value" {
      "en_pointer": 3erwf3ese6d5c4...,
      "policy": {
        "rule": {Effect},{Condition},
        "action": "read, update",
        "target": "{pk_1, pk_2, ...}"
      },
      "pk_enc": "fMA0GCSqGSIb3...",
      "hash": "369f2e3e69dc40543...",
      "timestamp": 1549480378
}}}
```

Listing 1. A state of the $3A\_ledger$ in JSON format. Content of the ledger includes *en_pointer*: ciphertext of a data pointer; *pk_enc*: public key used to encrypt the *en_pointer*; *policy*: data usage policy, and *hash* of the data.

```
{"log_ledger": {
    "key": {
      "owner": pk_DS,
      "controller": pk_DC,
      "processor": pk_DP
    }
    "value" {
      "access_token": "aAD0Gdfs234S3...",
      "issued_at": 1549480378,
      "status": "approved",
      "operation": op,
      "scope": []ops,
      "expires_in": 3600,
      "refresh_count": 1,
}}}
```

Listing 2. A state of the *log_ledger* in JSON format. Content of the ledger includes *status*: either *approved* or *rejected*, *operation*: an activity a *DP* used to process the data such as CRUD, *scope*: a set of allowed permissions, *expires_in* and *refresh_count*: dedicated to controlling the *access_token*.

referring to the latest state of the ledger, which is written in the most recent block of the BC. Some frameworks duplicate the latest state of a ledger (i.e., world-state) from a BC to a DBMS for better performance and for supporting advanced query capability (e.g., rich query). For example, either CouchDB[11] or LevelDB[12] are used in the HLF for its world-state database.

Following the design guidelines for distributed ledgers, we specify data models for two separate ledgers used in personal data management: $3A\_ledger$ (Listing 1) and log_ledger (Listing 2). The $3A\_ledger$ is used in authentication, authorisation and access control whereas the log_ledger is used for *access token* validation and logging. Both ledgers are in key-value format in which *keys* in the $3A\_ledger$ and log_ledger are $c-ID_{DS,DC}$ and $c-ID_{DS,DC,DP}$, respectively. The *value* in both ledgers contains information being used in the personal data management and provenance operations.

Note that the content of the ledgers can be seen by corresponding nodes in the BC network, either honest or malicious ones. Therefore, sensitive information should be protected. For instance, asymmetric cryptography is used for pseudo-anonymous identity; and reference to a dataset (i.e., data pointer p_m) is encrypted (Eq. 6).

$$en\_pointer = \mathcal{E}(pk_{enc}, p_m) \tag{6}$$

3) Authentication, Authorisation and Access Control: Public-key cryptography has been commonly used in BC-based systems to authenticate participants involved in a variety of tasks from consensus protocol participation to SC operations. In our design concept, the authentication is achieved by using the algorithm $\mathcal{V}$ in the 3-tuples digital signature scheme $\mathcal{G}, \mathcal{S}, \mathcal{V}$ based on any RSA/DSA-variants. The authorisation in personal data management is to specify access control (e.g., consent and usage policy); and data provenance tracking is to log data activities in an immutable and tamper-free ledger.

In the initial step (i.e., $Registration$ function), a DS grants consent to a DC for managing her personal data along with a shared key-pair (pk_{enc}, sk_{enc}). A new record is appended into the $3A\_ledger$ specifying a new key-pair for the personal dataset with default settings granting DS all permissions (e.g, CRUD operations) specified in the $policy$. The $policy$ can be considered as an access control list/rules for a dataset, updated when consent is granted or revoked. The *hash* and the *en_pointer* in the record are then updated once the DS upload her data to an RS by calling $DataUpload$ function. In our pseudo-codes, interactions with BC is through either $GetState$ or $PutState$ function provided by built-in APIs.

Algorithm 1 $GrantConsent$ Grants a Consent for a DP

Input : c-ID ci, signature t_{DS}, signature t_{DC}, public-key pk_{DP}, signature t_{DP}, permission op
Output: out

1 **Initialisation:** $rec \leftarrow null$, $out \leftarrow error$
2 $s1 \leftarrow \mathcal{V}(ci.pk_{DS}, t_{DS})$
3 $s2 \leftarrow \mathcal{V}(ci.pk_{DC}, t_{DC})$
4 $s3 \leftarrow \mathcal{V}(pk_{DP}, t_{DP})$
5 **if** $(s1 \wedge s2 \wedge s3)$ **then**
6 $policy \leftarrow$ GetState($3A\_ledger$).GetPolicy(ci)
7 PutState($3A\_ledger$).Update(ci, $policy$, $\{pk_{DP}, op\}$)
8 $rec \leftarrow$ JSON.Marshall($\{ci, pk_{DP}\}$, $\{scope[]$+=op, $access\_token$=rand(), $issue\_at$=Time.now(), $status$="approved"$\}$);
9 PutState(log_ledger).Append(rec);
10 $out \leftarrow success$
11 **Return** out

Fig. 3 depicts a sequence diagram of granting consent for a DP. The consent is granted if both DS and DP accept the request by providing their digital signatures t_DS and t_DC in step (2) and (3). Step (4) and (5) are carried out by the $GrantConsent$ function (Alg. 1). Authentication is achieved by using verification function $\mathcal{V}$ for all DS, DC, and DP (line 2-4). If the authentication is accepted (line 5), access control is then carried out by reflecting the permission into $policy$ in the $3A\_ledger$. As depicted in Alg. 1,

[11] http://couchdb.apache.org
[12] http://leveldb.org

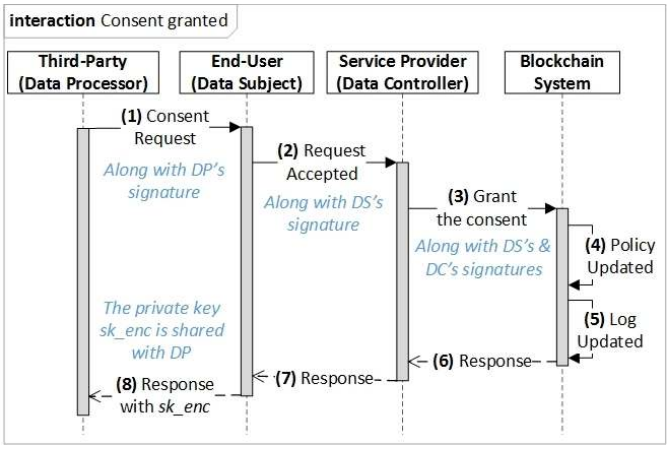

Fig. 3. Process of granting consent for a DP.

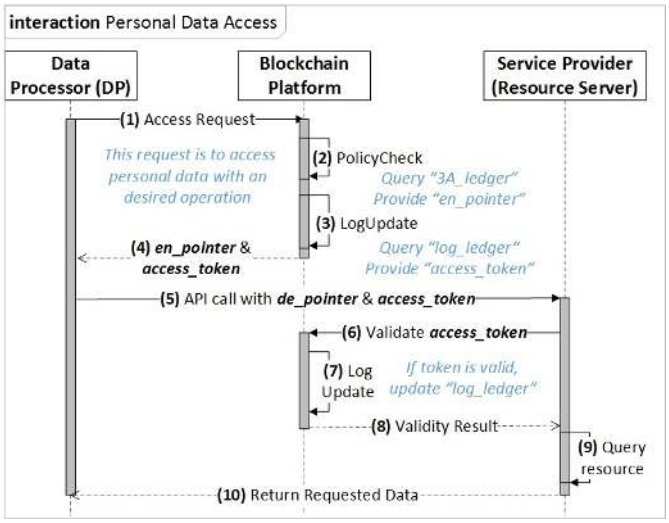

Fig. 4. Sequence diagram of accessing data stored in an RS by a DP.

the *GrantConsent* firstly grants permissions (i.e., requested operation *op*) by updating policy with *op* in the $3A\_ledger$ (line 6, 7). Secondly, the *GrantConsent* appends a new record into the log_ledger (line 9), which is used for validating and logging whenever the DP accesses the data. The $access\_token$ with other metadata is generated as *value* in the *key-value* format record (line 8). Technically, $access\_token$ is a string of random-looking characters referring to a collection of metadata in the log_ledger. A multi-signature technique is also used in the algorithm to ensure consent is granted by both DS and DC.

RevokeConsent function is to revoke a permission previously granted to a DP. As depicted in Alg. 2, it is only executed by either DS or DC. Similar to *GrantConsent* function, *RevokeConsent* appends an updated policy excluded the revoked permission *op* to the $3A\_ledger$ (line 4, 5) and updates the log_ledger accordingly (line 6,7).

Once consent is grant, the operation flow of accessing personal data is demonstrated in Fig. 4. Whenever DP desires to access personal data (step (1)), it invokes a corresponding SC with the *DataAccess* function (Alg. 3). As can be seen

Algorithm 2 *RevokeConsent* Revokes a Permission Previously Granted to a DP

Input : c-ID ci, signature t, public-key pk_{DP}, permission op
Output: out
1 **Initialisation:** $rec = null$, out = $error$
2 $s \leftarrow (\mathcal{V}(ci.pk_{DS}, t) \vee \mathcal{V}(ci.pk_{DC}, t))$
3 **if** s **then**
4 $policy \leftarrow$ GetState($3A\_ledger$).GetPolicy(ci)
5 PutState($3A\_ledger$).Update(ci, $policy$, {pk_{DP}, $-op$})
6 $rec \leftarrow$ GetState(log_ledger).GetRecord(ci, pk_{DP})
7 $rec \leftarrow$ PutState(log_ledger).Update(rec, {$scope$[]-=op, $access\_token$=rand(), $issue\_at$=Time.now()});
8 $out \leftarrow success$
9 **Return** out

in Fig. 4, after checking eligibility of the call (i.e., step (2) and (3) executed by line 2, 3 in Alg. 3), the SC returns two outputs $en\_pointer$ and $access\_token$ to the DP (step (4)), executed by line 6-9 in Alg. 3. The DP then uses the shared private key sk_{enc} (already obtained from step (8) in Fig. 3) for decrypting the $en\_pointer$. The decrypted ciphertext (i.e., $de\_pointer$) is the *datapointer* for the desired dataset. Both $de\_pointer$ and $access\_token$ are used as parameters for an API call to process the data (step (5)).

Algorithm 3 *DataAccess* Returns $en\_pointer$ and $access\_token$ for an Eligible Request

Input : c-ID ci, public-key pk_{DP}, signature t_DP, permission op
Output: out
1 **Initialisation:** $rec \leftarrow null$, $out \leftarrow rejected$
2 $s \leftarrow (\mathcal{V}(pk_{DP}, t_{DP})$
3 **if** s **then**
4 $policy \leftarrow$ GetState($3A\_ledger$).GetPolicy(ci)
5 **if** $(policy \subset (pk_{DP}, op))$ **then**
6 $en\_pointer \leftarrow$ GetState($3A\_ledger$).GetPointer(ci);
7 $access\_token \leftarrow$ GetState(log_ledger).GetToken(ci, pk_{DP});
8 $out \leftarrow (en\_pointer, access\_token)$
9 **Return** out

A function called $TokenValidation$ is dedicated to double-checking the validity of the $access\_token$ and updates the log_ledger. In Alg. 4, line 4 is to obtain metadata associated with the $access\_token$ from the log_ledger; if the request is from DS or DC then there is no need to validate the $access\_token$; only log_ledger is updated (line 5-7). Otherwise, the validation is then conducted by inspecting the metadata (line 9-12) before updating the log_ledger (line 13).

Algorithm 4 *TokenValidation* Double-Checks the Validity of an *access_token* and Update the *log_ledger*

Input : Token *access_token*, public-key *pk*, signature *t* permission *op*
Output: *out*
Initialisation: $rec \leftarrow null$, $out \leftarrow rejected$
$s \leftarrow (\mathcal{V}(pk, t)$
if *s* **then**
  $rec \leftarrow$ GetState(log_ledger).Query($access\_token$)
  if $((rec.owner = pk) \vee (rec.controller = pk))$ **then**
    $rec \leftarrow$ PutState(log_ledger).Update(rec, {$expires\_in$-=Time.now(), $issue\_at$=Time.now()});
    $out \leftarrow accepted$
  else
    if $(\ (rec.processor = pk) \wedge (rec.scope \subset op) \wedge (rec.expires\_in > 0) \wedge (rec.operation = op) \wedge (rec.status = approved) \wedge \ldots)$ **then**
      $rec \leftarrow$ PutState(log_ledger).Update(rec, {$expires\_in$-=Time.now(), $issue\_at$=Time.now()});
      $out \leftarrow accepted$
Return *out*

The *TokenValidation* is performed to ensure that only API calls with valid an *access_token* leads to an execution of the call (step (9)). Step (7) safeguards that all valid API calls are autonomously logged in the *log_ledger*. It is worth to mention that the honest-but-curious RS assumption plays a key role in the success of our platform because the RS must follow the authorisation process (i.e., double-check API calls from DPs with the BC system) before executing the calls.

V. Platform Deployment in Permission Blockchain

In this section, we implement a platform following the proposed design concept for managing personal profiles for an SNS. The choice of using a permissioned BC framework in the demonstration does not imply that a public one is less appropriate for implementing the proposed design concept. Instead, HLF is chosen due to its business-oriented architecture offering better adaptation to the use-case; also, thanks to its readily existing software components for a rapid development cycle of our platform. Detailed technical solutions and implementation of the platform are presented. Source-code of the demonstration can be obtained from Github.[13]

A. HLF Platform Setup

HLF is the most popular permissioned BC framework used by big enterprises such as IBM and Microsoft. As being permissioned, a node involved in an HLF network is associated with an identity and permissions provided by a CA and an MSP, respectively. Nodes in HLF take up one of three roles: *Client*, *Peer* and *Ordering Service Nodes* (OSNs). In our demonstration and for the performance evaluation, we have deployed different HLF network settings include 3 OSNs running in *Kafka* cluster mode for providing the ordering service, from 4 to 32 peers, and a varied number of clients from 10 to 1000. All peer nodes endorse both SCs (i.e., chaincodes in HLF terminology), namely $3A\_cc$ and log_cc. That means these two SCs are locally installed, instantiated and executed in all 5 peers to interact with the two ledgers $3A\_ledger$ and log_ledger, respectively. These two ledgers are exactly following the data models described in Section IV.D. As the two distributed ledgers are being used and HLF allows only one ledger per channel,[14] two HLF channels are created, namely $3A\_channel$ and $log\_channel$. All Peers and OSNs belong to both channels; the $3A\_cc$ and the log_cc SCs are operated in the $3A\_channel$ and the $log\_channel$, respectively. As a result, all the peer nodes separately endorse the two SCs corresponding to different local ledgers. The two local ledgers are stored in Linux filesystem whereas the world-state database is duplicated in CouchDB.

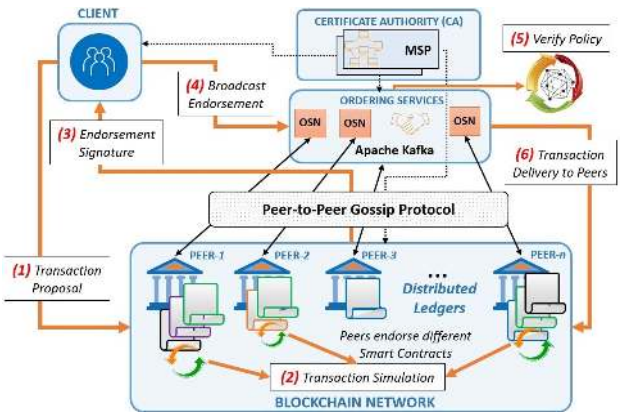

Fig. 5. High-level system architecture and transaction flow of the HLF framework.

All clients are populated using the Fabric Client SDK (for NodeJS) for interacting with the HLF network. As illustrated in Fig. 5, a client constructs a transaction proposal to invoke either $3A\_cc$ or log_cc SCs (step-1) and sends to all endorsing peers (i.e., endorsers). These peers verify the proposal and locally execute the $3A\_cc$ or log_cc to produce an endorsement signature (i.e., transaction results with the peer's signature) (step-2) and pass back to the client (step-3). Once receiving endorsement signatures, the client assembles the endorsements into the transaction and broadcast it to the OSNs, running *Kafka* mode (step-4). The OSNs validate and commit the transaction (step-5), then broadcast a message to all peers to update their local ledgers (step-6). In case the transaction is not successful, and the ledgers are not updated but the proposal is still logged for audit.

[13] https://github.com/nguyentb/Personal-data-management

[14] Channel is a terminology in HLF technically referring to a private blockchain overlays which offers data isolation and transaction confidentiality.

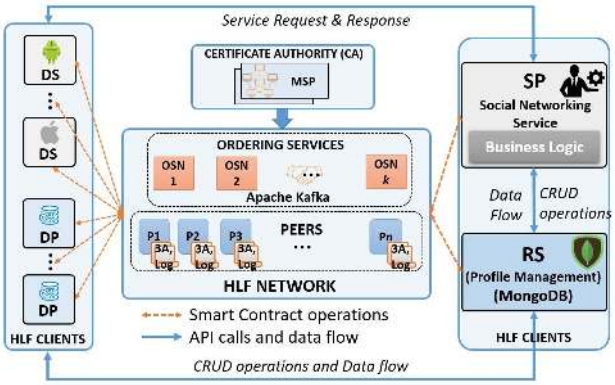

Fig. 6. System architecture of a GDPR-compliant social networking service with the RS for personal profiles using HLF.

B. Personal Profile Management Use-Case

We consider a use-case that a social networking SP processing profile data stored in a separate RS. This RS follows the honest-but-curious model anticipating the BC as an HLF client and honestly executing required protocols (i.e., interacting with the BC network for token validation). To comply with the GDPR, the SP participates in the proposed BC-based platform (Fig. 6). To demonstrate the use-case, we build the RS as a profile management web-service based on REST architecture[15] for parties to process profile data through calling corresponding RESTful APIs. Profile information is stored in JSON-like documents using MongoDB,[16] a document-oriented database system. The profile data model follows the Friend-Of-a-Friend (FOAF) ontology for describing person which is normally used in social networks.[17] Processing a profile includes $\{create, read, update, delete\}$ CRUD operations by making a request to a corresponding API provided by the RS.

A request to a RESTful API contains 6 parameters: (1)$API - Endpoint$, (2)$REST - Endpoint$, (3)$Method$, (4)$Header$, (5)$Params$, (6)$Payload$ in which the first fours are required. A RESTful request is as follows:

```
POST localhost:8080/ProfileManagement
-H 'Content-Type:application/json'
pubkey=pk&
signature=t&
token=access_token
&operation=read
```

where $Method$ is $POST$, $REST - Endpoint$ is $localhost$:8080, $API-Endpoint$ is $/ProfileManagement$, Header is $Content - Type$:$application/json$ following by $Params$ including the public-key pk with the signature t, the $access\_token$, and the requested $Read$ operation.

C. Identity Management and Pseudo-Anonymity

Any entity in HLF including clients, peers, orderers, CAs and MSPs needs to be identified by digital identities (e.g., $X.509$ standard) before interacting with the HLF network. In our HLF-based system, a built-in CA called *Fabric CA* is used to generate $X.509$ digital certificates, adopting the traditional Public Key Infrastructure (PKI) hierarchical model. An $X.509$ digital certificate contains a public key (along with a corresponding private key) and associated information of an entity (e.g., organisation, host-name, and domain. This certificate is then either signed by the Fabric CA or self-signed. The Fabric CA server in our system is initialised using Docker which hosts an HTTP server on the default port 7054 that offers REST APIs. All entities have to enrol and register with the CA server via the REST APIs using either the *Fabric CA client* or the *Fabric SDK* before participating in the blockchain system. Once an entity is enrolled and registered, an enrolment certificate ($eCert$), a network transaction certificate ($tCert$), a CA certificate, and a corresponding private key are stored in PEM files in the subdirectories of the entity's directory.

In the HLF settings, ECDSA, an updated version of the DSA scheme leveraging elliptic-curve cryptography, is used with 256-bit key-size, which guarantees that any public-private key pairs generated by the generator $\mathcal{G}$ is *practically* unique across the HLF network. Moreover, the hiding property of the ECDSA also ensures that there is no practical mechanism to recover a private key from the corresponding public key [46]. As a result, HLF entities, whose identifiers are $X.509$ digital certificates, preserve the pseudo-anonymity property. However, as HLF is a permissioned blockchain, all of HLF entities are under control of a certificate authority CA (in our system is the built-in Fabric CA); this means the pseudo-anonymity property depends on the security and trustworthiness of the Fabric CA.

To administer entities evolving in variety of HLF tasks, MSP is used for specifying participants, roles, and access privileges in a HLF network and channel. An MSP provides a configuration identifying trusted root and intermediate CAs; these CAs then define members of a trust domain by either *(i)* listing identities of the members or *(ii)* identifying authorised CAs that issue valid identities for members. The latter is used in the demonstration. Technically, an entity's identity is associated with its MSP and implemented using the HLF client identity chaincode library cid[18] as shown in Listing 3:

D. Smart Contracts Implementation

There are two chaincodes implemented in the HLF network: *(i)* the $3A\_cc$ for authentication, authorisation and access control, operating with the $3A\_ledger$; and *(ii)* the log_cc for access validation and logging, operating with the log_ledger. Theoretically, a contract can be written in any programming language; and in the demonstration, Go language is used. The two chaincodes inherit the built-in $shim$ package,[19] which provides a variety of APIs to interact with distributed ledgers such as accessing state variables, transaction context and call other chaincodes.

Regarding the distributed ledgers, $en\_pointer$ is the ciphertext of an identifier of a data object (i.e., $profile.ID$) using

[15]https://en.wikipedia.org/wiki/Representational_state_transfer
[16]https://www.mongodb.com/
[17]http://xmlns.com/foaf/spec/
[18]https://github.com/hyperledger/fabric/blob/release-1.1/core/chaincode/lib/cid/README.md
[19]https://godoc.org/github.com/hyperledger/fabric/core/chaincode/shim

```
function ClientID(stub shim.ccAPI) ci *clientID {
    hlfId = ci.New(stub);
    mspID = hlf.GetMSPID();
    cert = hlf.GetX509Certificate();
    return &clientID{mspID, cert};
}
```

Listing 3. Identity of a HLF client constituted from $mspID$ and $X.509$ certificates utilising the *cid* library.

```
"policy" {
    "Create": {pk_DS, pk_DC, ..},
    "Read": {pk_DS, pk_DC, pk_DP1, pk_DP2, ..},
    "Update": {pk_DS, pk_DC, pk_DP3 ...},
    "Delete": {pk_DS, pk_DC, pk_DP3, pk_DP4, ..}
}
```

Listing 4. Data usage policy defined as an access control list under JSON format.

the encryption function $\mathcal{E}$ with the encryption key pk_{enc}:

$$en\_pointer = \mathcal{E}(pk_{enc}, profile.ID) \tag{7}$$

A party who permitted access a profile has a shared private key sk_enc to decrypt $en\_pointer$ in order to obtain the $profile.ID$, which is then passed as a parameter for a RESTful API to access the desired profile information:

$$profile.ID = \mathbb{D}(sk_{enc}, en\_pointer) \tag{8}$$

The $policy$ in the $3A\_ledger$ is simply defined as an access control list (ACL) as shown in Listing 4. The ACL is implemented as a struct in the $3A_cc$ specifying four *access rights* for participants called $Create$, $Read$, $Update$ and $Delete$ (representing four CRUD operations). Associated with each *access right* is a list of granted parties including DS, DC and DPs (under their public keys such as pk_DS, pk_DC, and pk_DP).

Based on the identity scheme and detailed information for the two ledgers, core functions in personal data management such as $GrantConsent$, $RevokeConsent$, $TokenValidation$ and $DataAccess$ are then implemented exactly following the algorithms described in Section III.D.

VI. Analysis and Discussion

This section provides analysis and discussion on the platform deployed in Section V, including GDPR-compliance applicability, threat models and system performance.

A. Trust Assumption

Besides the *honest-but-curious* characteristic of the RS, a must assumption is that a large portion of peer nodes in the HLF network are honest. Generally, HLF v1.x offers multiple ordering techniques including a variety of BFT-based solutions such as pBFT. Such BFT-variant protocols are able to conditionally tolerate $\lfloor \frac{N-1}{5} \rfloor$ (e.g., in Ripple [47]) to $\lfloor \frac{N-1}{2} \rfloor$ (e.g., in crash-fault tolerance) simultaneously faulty nodes. Such BFT-variant protocols guarantee consistency despite any number of node failures and network partition, with at most $\lfloor \frac{N-1}{3} \rfloor$ faulty nodes [48]. Unfortunately, these protocols are under development for the HLF framework, and only Apache Kafka is provided as a reference implementation, which supports some levels of fault-tolerant (e.g., crash-faulty) but not Byzantine failure.

The cryptographic primitives (i.e., cryptographic hash function $SHA256$, the public-key cryptography RSA and the digital signature schemes ECDSA) are practically secure. This means adversaries are not able to: *(i)* reverse/break the cryptographic hash function, *(ii)* reverse a public key to obtain a private key, and *(iii)* forge a digital signature of another party without knowing the corresponding private key. As our system is built on top of the permissioned blockchain HLF, the Fabric CA with built-in PKI, which are responsible for the distribution of management of $X.509$ digital certificates, are assumed to be secure and honest. This means in general adversaries are not able to mislead the Fabric CA in large-scale (e.g., more than $\lfloor \frac{N-1}{3} \rfloor$ adversaries granted in the HLF network) in order to subvert the HLF system (e.g., 51% attack or Sybil attack). However, some internal adversaries might be granted to participate in the network, resulting in non-GDPR-compliance. Regarding key management, we assume that private keys obtained by the key generator $\mathcal{G}$ are effectively protected from adversaries by leveraging existing solutions from enterprise systems. However, this is the weak assumption, meaning that an adversary could somehow obtain a private key and impersonate an honest party to access data which also leads to non-GDPR-compliance. These threats of non-GDPR-compliance will be considered under Section VI.C.

B. GDPR-Compliance

From an applicability perspective, the proposed platform provides SPs (e.g., the SNS) mechanisms to fully comply with the GDPR. This is due to the following reasons:

1) Full Control Back to Data Owners: As following the design concept, the platform provides DSs:

- "Right of access" and "right of rectification": This is because DS is eligible to do all CRUD operations to her personal data as specified in the default policy when ledgers are initialised, and no one can change these rights.
- "Right of restricted processing" and "right of data portability": This is because DSs have full permissions to manage data usage policy (e.g., to grant or revoke consent anytime/anywhere by invoking the $GrantConsent$ and $RevokeConsent$ functions in the $3A\_cc$).
- "Right to be informed": This is because the platform always requires DS's signature for data collection or for granting consent.
- "Right to be forgotten": As personal data is stored off-chain, an RS is able to erase the data as requested from DS. However, a question is posed when leveraging BC for personal data management: "whether a BC platform complies with the GDPR as distributed ledgers are immutable; meaning that the ledgers, theoretically, will never be erased?". Therefore, if a piece of personal information is recorded in a ledger, the platform will violate the "right of forgotten". In the design concept, sensitive information is encrypted before writing into a

ledger (e.g., $data\_pointer$). The "right of forgotten" is then ensured by throwing decryption keys. Whether this remedy fully satisfies the GDPR is still an open question [44], [49].

2) Security, Transparency and Accountability: By following the design concept, the platform ensures that:

- Security of the identity, authentication and authorisation mechanisms, which depends on the security of the cryptographic primitives, is assumed to be secure.
- Operations (e.g., grant or revoke a consent, update usage policy, verify *access token*, and CRUD) are authenticated, authorised and autonomously executed only by invoking corresponding SCs deployed in the HLF network. This ensures system procedures are executed in a transparent and not compromised by any individuals.
- Information about management operations and CRUD activities on personal data, including who/what/when/ why/ and how are immutably recorded in the log_ledger.

Consequently, the proposed platform forces SPs, who participate in the system, to be responsible for complying with the GDPR; otherwise any unauthorised or malicious transactions initiated by a corresponding SP can be always figured out. Furthermore, the investigation for GDPR-compliance is empowered as all activities logged in the ledgers can be traced back. The signalling of a non-compliant activity could trigger official investigation and auditing of an SP by a supervisory authority. The decisions could be made based on whether a malicious activity recorded in the log_ledger exists that respects the associated data usage policies in the $3A\_ledger$. In this regard, the two distributed ledgers can be considered as legal grounds for the GDPR compliance. As a result, the platform is able to demonstrate the GDPR compliance. Therefore, the proposed BC-based platform provides efficient measures to meet the requirements of data accountability. For those reasons, a social networking SP, which utilises the platform for its personal data management tasks, fully complies with the GDPR.

C. Threat Models

The advanced capability of the BC framework plays a key role in providing a secure and trustworthy platform for complying with the GDPR. However, certain aspects of the contemporary BC and SC technologies present limitations imposing threats resulting in non-compliance with the GDPR.

1) Security Threats: Given the aforementioned assumptions, the decentralised nature of the BC ensures that an adversary cannot corrupt the BC network to unauthorisedly change the ledgers as that would imply the majority of the network's resources are compromised. Also, the adversary cannot impersonate an authorised party as its digital signature cannot be forged. Security threats are, thus, from two sources: *(i)* an internal adversary acting in a Byzantine way, who has been granted to access personal data; and *(ii)* an honest party whom both private key and decryption key sk_{enc} are disclosed to an external adversary; thus, the adversary could pose itself as the party. In such scenarios, the $TokenValidation$ function is of paramount importance since it plays as a role of a gatekeeper to reassure that any $access\_token$ expires after the amount of time and needs to be refreshed (i.e., re-authenticated and re-authorised). As a result, the $TokenValidation$ mitigates the risk of a long-lived $access\_token$ leaking, similar to the use of both $access\_token$ and $refresh\_token$ used in the standard OAuth2 specification.[20]

Admittedly, it is inevitable that an adversary is able to access the data in the time-frame window of the $access\_token$ (defined by the $expires\_in$ parameter in the log_ledger). During this period, it is unachievable to prevent the adversary from accessing data unless the security breach is detected. Once being detected, DS is able to revoke the consent by updating the ledgers to remove all permissions related to the adversary. The remedy is straightforward in case of the first scenario - the party is malicious. However, it turns to a complex situation when an honest party leaks its private key to the adversary. This party is never able to get granted again as its identity is compromised, which is unreasonable. A key management with an account recovery scheme could be an applicable solution to deal with this situation although it is expected to be much complicated to integrate the recovery scheme with a BC system [50]. Another security threat comes from poor quality code in SCs which exposes vulnerabilities to be exploited. For example, an attacker stole $3.6M$ Ether (worth \$50M at that time) in DAO[21] attack exploiting a concurrency bug in DAO's SCs. As a BC framework supporting Turing-complete SCs, software bugs are painful to avoid. Thus, SCs must be written in high-quality standards and follows strict security specifications [30], [51].

2) Privacy Threats: The openness of distributed ledgers, which allows parties to inspect, violates the idea of privacy. Even in a permissioned BC in which transactions take place between authenticated parties, some privacy threats remain as any participants could be malicious. In the proposed design concept, measures to tackle privacy leakage are to both: *(i)* provide pseudo-anonymity for parties using public key cryptography as identities; and *(ii)* encrypt sensitive information exposed on the ledgers.

The first measure provides pseudo-anonymity, thus, there is a possibility to link between public addresses with physical identification of the users by using a variety of de-anonymisation techniques [52]. Literally, the risk of revealing real-world identity by an adversary can be significantly reduced in a permissioned BC compared to a public one thanks to an additional permission access control layer [27], [53]. As a trade-off, anonymity is sacrificed as it requires more identity materials for stringent privacy requirements.

The second measure is to encrypt $data\_pointer$ (i.e., $profile.ID$ in the demonstration), which is used as a parameter in API calls for accessing a personal dataset. The encryption ensures that the information is only visible to designated parties, reducing the risk of leaking the information to adversaries. Some other information recorded onto the ledgers such as data usage policy (i.e., $policy$) and activities log must be in plain text as the information is referred by

[20] https://tools.ietf.org/html/rfc6749

[21] https://ethereum.org/dao

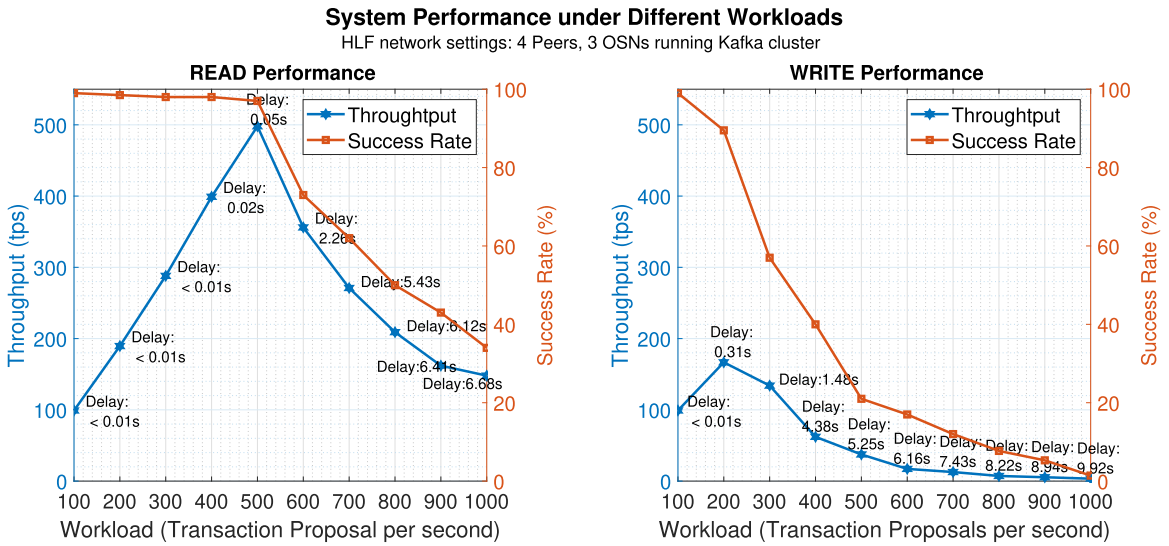

Fig. 7. Performance of $READ$ and $WRITE$ from/to distributed ledgers in the HLF-based system under different workloads.

peers for some business logic when executing the chaincodes. Even though this information is not directly related to identifiable individuals, this might be a source of privacy leakage as it might be used in de-anonymisation techniques. At this moment, no particular privacy threat has been pointed out due to exposing such information. Nevertheless, further investigation might need to be carried out for this potential threat. Homomorphic Encryption could be used for information encryption supporting query on cipher-text [54]. Flexible encryption schemes like attribute-based encryption (ABE) [55] might also be used as a remedy to encrypt such information and only a designated group of peers can be decrypted. These schemes are only suitable for permissioned BC as they rely on a trusted key generator - which could be integrated in a CA.

D. Performance Evaluation

As the proposed platform is expected to serve a large number of clients accessing data simultaneously, performance, and scalability of the platform is necessarily evaluated. The core technology leaders in BC such as Bitcoin, Ethereum Enterprise Alliance and Hyperledger Foundation have demonstrated promising technology advancements of both performance and scalability. However, at the moment, public BCs can only achieve limited throughput (e.g., Bitcoin gets 7 transactions per second (tps) with $Blocktime$ is around 10 minutes whereas Ethereum reaches around 15 tps with 15-second $Blocktime$ [22]). In permissioned BCs, additional permission control ensures that a majority of nodes are trusted; as well as all identities of the participants in the network are known. This allows the use of BFT-variant consensus in the BC platforms, theoretically resulting in higher throughput. For instance, FabricCoin deployed on top of the HLF framework can achieve about $3,500$ tps at a second latency [53]. However, scalability incurs as a critical issue for a permissioned BC framework, especially frameworks with the pBFT-variant consensus mechanisms.

1) Hyperledger Caliper Performance Benchmark: For our performance evaluation, we use a new evaluation tool developed by the Hyperledger Foundation called Caliper,[23] which is a performance benchmark framework for various BC frameworks including HLF, Hyperledger Sawtooth and Ethereum. Caliper is equipped with adaptors implementing interfaces for interacting with HLF systems version 1.x using either HLF native SDK or a RESTful API. To integrate with our existing HLF profile management system, we have programmed our adaptors using Fabric Client SDK (NodeJS version) to interact with the BC network and to invoke the two chaincodes $3A\_cc$ and log_cc. On top of the adaptation layer is a benchmark layer implementing predefined use-cases in the form of $YAML$ configuration files. We have written various use-cases for the performance benchmark following these configurations:

- $READ$ the ledgers (e.g., invoke $policy\_check$ function) and $WRITE$ the the ledgers (e.g., invoke the $GrantConsent$ and $RevokeConsent$ functions to update the ACL policy).
- Different HLF network settings in which the number of peer nodes are varied from 4 to 32.
- Different workloads to the system by generating a number of transaction proposals per second to the system. In each network setting, the workload is from 100 tps to 1000 tps.

2) Results and Analysis: There are four metrics in the Caliper benchmarking results, namely *(1)* Success Rate, *(2)* Throughput, *(3)* Latency, and *(4)* Resource Consumption. These metrics are counted from the time a transaction submitted by a client until it is processed and is written on a distributed ledger. Fig. 7 interprets our system performance under different number of workloads, from 100tps to 1000tps. The HLF network setting includes 4 peer nodes and 3 OSNs running Kafka cluster for crash-fault tolerance consensus.

[22] https://bitinfocharts.com/

[23] https://hyperledger.github.io/caliper/

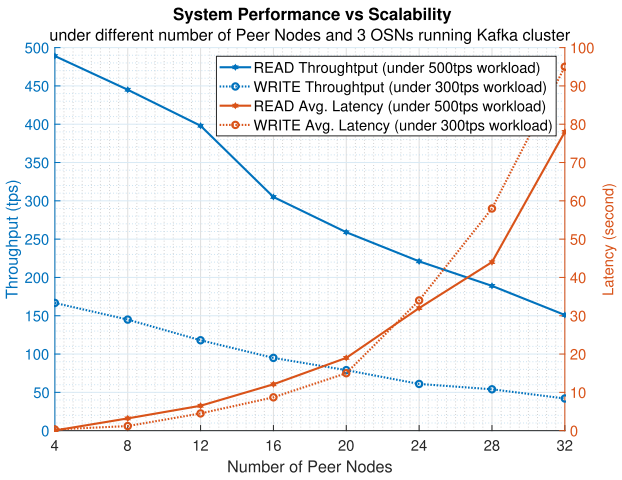

Fig. 8. System performance vs scalability under different number of peer nodes.

In this benchmark, 1000 clients are popularised that generates transaction proposals (including both $READ$ and $WRITE$ a distributed ledger) to our system. As can be seen in the figure, throughput of $READ$ transactions can reach highest to 492tps at 500-tps workload whereas $WRITE$ transactions only reach to 167 tps at 300-tps workload with highest success rate (more than 95%) and with less than 1-second latency. Compared to $READ$ transactions, $WRITE$ transactions require more processes from OSNs to chronologically order the transactions, create a new block, and broadcast it to all peers in the network to update a distributed ledger; that is why $WRITE$ transactions get lower throughput, lower success rate, and higher latency. After these peaks, the throughputs and the success rates of both $READ$ and $WRITE$ transactions dramatically decrease. For instance, at 1000-tps workload, the throughputs and success rates drop to about 34.5 tps and 30%; and 3.4 tps and 1.4% for $READ$ and $WRITE$ transactions, respectively. The average latency is significantly risen from less than 1 second to 6.68 seconds ($READ$ transaction) and 9.92 seconds ($WRITE$ transactions) as higher workload is generated.

Generally, the reason that the system cannot handle high workload due to local processing bottleneck as transactions are queued at endorsing peer's buffer and OSNs' buffers (for $WRITE$ transactions) to be processed. The HLF procedure requires that a transaction has to obtain enough proposal responses from endorsing peers, thus, if an endorsing peer processes the transaction lower than others, the transaction is delayed accordingly. Particularly for $WRITE$ transactions which require more processes for ordering service as such, all $WRITE$ transactions need to be buffered and processed at the 3 OSNs running Kafka cluster. As we observed, OSNs are always busy that the docker container can consume to 88% CPU-load on average.

Fig. 8 illustrates the performance vs. scalability of our proposed system when the number of peer nodes increasing while the ordering service remains the same with 3 OSNs running Kafka cluster. In this performance benchmark test, $READ$ and $WRITE$ transactions are set under 500 tps and 300 tps workload, respectively. More peer nodes mean more overhead messages exchanged between nodes, and the wait for endorsement messages before broadcasting a transaction response to the OSNs to create a new block and to update a distributed ledger. That is why the throughput decreases and the latency increases for both $READ$ and $WRITE$ transactions. As depicted in Fig. 8, the proposed HLF-based profile management system fails to support high performance and scalability since the throughput significantly decreases and the latency dramatically increases when the BC network scales up (e.g., at 32 peer nodes, throughputs are 151 tps and 42 tps and latencies are 78s and 95s for $READ$ and $WRITE$ transactions, respectively). Fortunately, HLF allows us to partition a BC network in which only a subset of peer nodes are permitted to endorse a particular chain-code. This will reduce the number of messages exchanged across the network as well as reduce the waiting time for endorsement messages from endorsing peers. As a trade-off, decentralisation is partly sacrificed and the system is more sensitive to 51% and selfish mining attacks [56].

VII. Conclusion and the Road Ahead

In this article, a design concept for a GDPR-compliant BC-based personal data management platform is proposed. Following the guidelines from the design concept including system architecture, ledger data models, and SC functionalities, a BC-based platform is implemented on top of the HLF framework. The platform interplays among an honest RS, a social networking SP, DPs, and DSs ensuring that all processing activities over profile data stored in the RS are compliant with the GDPR. The feasibility and effectiveness of the design concept are, therefore, successfully demonstrated.

For future work, we will deploy the design concept in a public BC (e.g., Ethereum) with an RS using distributed storage (e.g., IPFS, BigchainDB or Storj). In this regard, the RS is not trustworthy as some storage nodes might be malicious. Thus, more mechanisms need to be implemented to resolve the lack of a trusted centralised RS. As a reward, the system is truly decentralised. Another work is to develop a fine-grain expressive data usage policy using a policy language instead of a simple ACL as in the demonstration. A policy generator deployed in SCs that autonomously acquires data usage policy depending on specific contexts is also a promising research direction. Additionally, pricing and incentive models for the cost of data storage and BC operations should be carried out to finalise a complete system.

As the processing of personal data refers to CRUD operations – which is under the mindset of data storage, an ambitious research direction is to provide computational capability on a BC network [32]. This means an SP directly runs computation on the network and obtain results using secure Multi-Party Computation (MPC).[24] This approach is much securer as the SP does not directly observe raw data. We believe our work acts as a catalyst to open a variety of research directions regarding the use of BC and SCs in decentralised authorisation and access control, which plays

[24]https://en.wikipedia.org/wiki/Secure_multi-party_computation

a crucial role in digital assets management, particularly in personal data regulations.

REFERENCES

[1] H. Shafagh, L. Burkhalter, A. Hithnawi, and S. Duquennoy, "Towards blockchain-based auditable storage and sharing of IoT data," in *Proc. Cloud Comput. Secur. Workshop*, 2017, pp. 45–50.

[2] R. Li, T. Song, B. Mei, H. Li, X. Cheng, and L. Sun, "Blockchain for large-scale Internet of Things data storage and protection," *IEEE Trans. Services Comput.*, vol. 12, no. 5, pp. 762–771, Sep./Oct. 2019.

[3] S. Wang, Y. Zhang, and Y. Zhang, "A blockchain-based framework for data sharing with fine-grained access control in decentralized storage systems," *IEEE Access*, vol. 6, pp. 38437–38450, 2018.

[4] J. Benet, "IPFS—Content addressed, versioned, P2P file system," Jul. 2014, *arXiv:1407.3561*. [Online]. Available: https://arxiv.org/abs/1407.3561

[5] G. Zyskind, O. Nathan, and A. Pentland, "Decentralizing privacy: Using blockchain to protect personal data," in *Proc. IEEE Secur. Privacy Workshops*, May 2015, pp. 180–184.

[6] L. A. Linn and M. B. Koo, "Blockchain for health data and its potential use in health it and health care related research," in *Proc. ONC/NIST Use Blockchain Healthcare Res. Workshop*. Gaithersburg, MD, USA: ONC/NIST, 2016, pp. 1–10.

[7] A. Azaria, A. Ekblaw, T. Vieira, and A. Lippman, "MedRec: Using blockchain for medical data access and permission management," in *Proc. 2nd Int. Conf. Open Big Data (OBD)*, Aug. 2016, pp. 25–30.

[8] M. J. M. Chowdhury, A. Colman, M. A. Kabir, J. Han, and P. Sarda, "Blockchain as a notarization service for data sharing with personal data store," in *Proc. 17th IEEE Int. Conf. Trust, Secur. Privacy Comput. Commun. (TrustCom)*, Aug. 2018, pp. 1330–1335.

[9] R. Neisse, G. Steri, and I. Nai-Fovino, "A blockchain-based approach for data accountability and provenance tracking," in *Proc. 12th Int. Conf. Availability, Rel. Secur.*, 2017, p. 14.

[10] B. Faber, G. Michelet, N. Weidmann, R. R. Mukkamala, and R. Vatrapu, "BPDIMS: A blockchain-based personal data and identity management system," in *Proc. 52nd Hawaii Int. Conf. Syst. Sci.*, 2019, pp. 1–10.

[11] C. Wirth and M. Kolain, "Privacy by blockchain design: A blockchain-enabled GDPR-compliant approach for handling personal data," in *Proc. 1st ERCIM Blockchain Workshop, Eur. Soc. Socially Embedded Technol. (EUSSET)*, 2018, pp. 1–7.

[12] M. Walport *et al.*, *Distributed Ledger Technology: Beyond Blockchain*, vol. 1. London, U.K.: Government Office for Science, 2016.

[13] P. Voigt and A. Von dem Bussche, "The eu general data protection regulation (GDPR)," in *A Practical Guide*, 1st ed. Cham, Switzerland: Springer, 2017.

[14] I. P. Team, *EU General Data Protection Regulation (GDPR): An Implementation and Compliance Guide*. Cambridge, U.K.: IT Governance, 2017.

[15] S. Nakamoto, *Bitcoin: A Peer-to-Peer Electronic Cash System*. Bitcoin, 2008. [Online]. Available: https://bitcoin.org/en/bitcoin-paper

[16] M. Crosby, P. Pattanayak, S. Verma, and V. Kalyanaraman, "Blockchain technology: Beyond bitcoin," *Appl. Innov.*, vol. 2, nos. 6–10, p. 71, 2016.

[17] F. Tschorsch and B. Scheuermann, "Bitcoin and beyond: A technical survey on decentralized digital currencies," *IEEE Commun. Surveys Tuts.*, vol. 18, no. 3, pp. 2084–2123, 3rd Quart., 2016.

[18] N. B. Truong, T.-W. Um, B. Zhou, and G. M. Lee, "Strengthening the blockchain-based Internet of value with trust," in *Proc. IEEE Int. Conf. Commun. (ICC)*, May 2018, pp. 1–7.

[19] V. Gramoli, "From blockchain consensus back to byzantine consensus," *Future Gener. Comput. Syst.*, to be published.

[20] W. Wang *et al.*, "A survey on consensus mechanisms and mining strategy management in blockchain networks," *IEEE Access*, vol. 7, pp. 22328–22370, 2019.

[21] A. Gervais, G. O. Karame, K. Wüst, V. Glykantzis, H. Ritzdorf, and S. Capkun, "On the security and performance of proof of work blockchains," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2016, pp. 3–16.

[22] A. Kiayias, A. Russell, B. David, and R. Oliynykov, "Ouroboros: A provably secure proof-of-stake blockchain protocol," in *Proc. Int. Cryptol. Conf.* Santa Barbara, CA, USA: Springer, 2017, pp. 357–388.

[23] I. Bentov, C. Lee, A. Mizrahi, and M. Rosenfeld, "Proof of activity: Extending bitcoin's proof of work via proof of stake," *IACR Cryptol. ePrint Arch.*, vol. 2014, p. 452, 2014.

[24] A. Miller and J. J. LaViola, Jr. (2014). *Anonymous Byzantine Consensus From Moderately-Hard Puzzles: A Model for Bitcoin*. [Online]. Available: http://nakamotoinstitute.org/research/anonymous-byzantine-consensus

[25] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich, "Algorand: Scaling byzantine agreements for cryptocurrencies," in *Proc. 26th Symp. Operating Syst. Princ.*, 2017, pp. 51–68.

[26] V. Buterin. (Apr. 2014). *White Paper: A Next-Generation Smart Contract and Decentralized Application Platform*. [Online]. Available: https://www.ethereum.org/pdfs/EthereumWhitePaper.pdf

[27] A. Kosba, A. Miller, E. Shi, Z. Wen, and C. Papamanthou, "Hawk: The blockchain model of cryptography and privacy-preserving smart contracts," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2016, pp. 839–858.

[28] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum Project Yellow Paper*, vol. 151, pp. 1–32, Apr. 2014.

[29] C. Cachin, "Architecture of the hyperledger blockchain fabric," in *Proc. Workshop Distributed Cryptocurrencies Consensus Ledgers*, vol. 310, 2016, pp. 1–4.

[30] H. A. WC. (2018) *Hyperledger Architecture-Volume II-Smart Contracts*. [Online]. Available: https://www.hyperledger.org/wp-content/uploads/2018/04/Hyperledger_Arch_WG_Paper_2_SmartContracts.pdf

[31] K. Markus and G. Chung, *Blockchain in Logistics*. Germany: DHL Trend Research, 2018.

[32] F. Tian, "An agri-food supply chain traceability system for China based on RFID & blockchain technology," in *Proc. 13th Int. Conf. Service Syst. Service Manage. (ICSSSM)*, Jun. 2016, pp. 1–6.

[33] N. Hackius and M. Petersen, "Blockchain in logistics and supply chain: Trick or treat?" in *Proc. Hamburg Int. Conf. Logistics (HICL)*, 2017, pp. 3–18.

[34] X. Liang, S. Shetty, D. Tosh, C. Kamhoua, K. Kwiat, and L. Njilla, "ProvChain: A blockchain-based data provenance architecture in cloud environment with enhanced privacy and availability," in *Proc. 17th IEEE/ACM Int. Symp. Cluster, Cloud Grid Comput.*, May 2017, pp. 468–477.

[35] M. Ali, J. Nelson, R. Shea, and M. J. Freedman, "Blockstack: A global naming and storage system secured by blockchains," in *Proc. Annu. Tech. Conf. (USENIX/ATC)*, 2016, pp. 181–194.

[36] H. A. Kalodner, M. Carlsten, P. Ellenbogen, J. Bonneau, and A. Narayanan, "An empirical study of namecoin and lessons for decentralized namespace design," in *Proc. WEIS*, 2015, pp. 1–21.

[37] T. McConaghy *et al.*, "BigchainDB: A scalable blockchain database," BigChainDB, Berlin, Germany, White Paper, 2016. [Online]. Available: https://www.bigchaindb.com/whitepaper/bigchaindb-whitepaper.pdf

[38] J.-H. Lee, "BIDaaS: Blockchain based ID as a service," *IEEE Access*, vol. 6, pp. 2274–2278, 2017.

[39] Z. Chen, S. Chen, H. Xu, and B. Hu, "A security authentication scheme of 5G ultra-dense network based on block chain," *IEEE Access*, vol. 6, pp. 55372–55379, 2018.

[40] O. Novo, "Blockchain meets IoT: An architecture for scalable access management in IoT," *IEEE Internet Things J.*, vol. 5, no. 2, pp. 1184–1195, Apr. 2018.

[41] D. Hardt, "The oauth 2.0 authorization framework," IETF, Fremont, CA, USA, Tech. Rep. rfc6749, 2012.

[42] T. Lodderstedt, M. McGloin, and P. Hunt, "Oauth 2.0 threat model and security considerations," IETF, Fremont, CA, USA, Tech. Rep. rfc6819, 2013.

[43] R. Neisse, G. Steri, I. N. Fovino, and G. Baldini, "SecKit: A model-based security toolkit for the Internet of Things," *Comput. Secur.*, vol. 54, pp. 60–76, Oct. 2015.

[44] M. Berberich and M. Steiner, "Blockchain technology and the GDPR-how to reconcile privacy and distributed ledgers," *Eur. Data Protection Law Rev.*, vol. 2, p. 422, Mar. 2016.

[45] S. Wilkinson, T. Boshevski, J. Brandoff, and V. Buterin, "Storj a peer-to-peer cloud storage network," Storj Lab. Inc., Atlanta, GA, USA, Tech. Rep. Ver3, Oct. 2018. [Online]. Available: https://storj.io/storjv3.pdf

[46] D. Johnson, A. Menezes, and S. Vanstone, "The elliptic curve digital signature algorithm (ECDSA)," *Int. J. Inf. Secur.*, vol. 1, no. 1, pp. 36–63, Aug. 2001.

[47] D. Schwartz, N. Youngs, and A. Britto, "The ripple protocol consensus algorithm," *Ripple Labs Inc White Paper*, vol. 5, p. 8, Feb. 2018.

[48] S. Liu, P. Viotti, C. Cachin, and V. Quéma, and M. Vukolić, "{XFT}: Practical fault tolerance beyond crashes," in *Proc. 12th USENIX)Symp. Operating Syst. Design Implement. (OSDI)*, 2016, pp. 485–500.

[49] C. R. Meijer. (2018). *Blockchain Versus GDPR and Who Should Adjust Most*. [Online]. Available: https://www.finextra.com/blogposting/16102/blockchain-versus-gdpr-and-who-should-adjust-most
[50] H. Zhao, P. Bai, Y. Peng, and R. Xu, "Efficient key management scheme for health blockchain," *CAAI Trans. Intell. Technol.*, vol. 3, no. 2, pp. 114–118, Jun. 2018.
[51] L. Luu, D.-H. Chu, H. Olickel, P. Saxena, and A. Hobor, "Making smart contracts smarter," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2016, pp. 254–269.
[52] S. Meiklejohn *et al.*, "A fistful of bitcoins: Characterizing payments among men with no names," in *Proc. Conf. Internet Meas. Conf.*, 2013, pp. 127–140.
[53] E. Androulaki *et al.*, "Hyperledger fabric: A distributed operating system for permissioned blockchains," in *Proc. 13th EuroSys Conf.*, 2018, p. 30.
[54] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *Proc. STOC*, vol. 9. 2009, pp. 169–178.
[55] V. Goyal, O. Pandey, A. Sahai, and B. Waters, "Attribute-based encryption for fine-grained access control of encrypted data," in *Proc. 13th ACM Conf. Comput. Commun. Secur.*, 2006, pp. 89–98.
[56] I. C. Lin and T. C. Liao, "A survey of blockchain security issues and challenges," *Int. J. Netw. Secur.*, vol. 19, no. 5, pp. 653–659, 2017.

Nguyen Binh Truong received the B.Sc., M.Sc., and Ph.D. degrees from Liverpool John Moores University, U.K., Pohang University of Science and Technology, South Korea, and Hanoi University of Science and Technology, Vietnam, in 2008, 2013, and 2018, respectively. He was a Software Engineer with DASAN Networks, from 2012 to 2015, a leading company on networking products and services in South Korea. He is currently a Research Associate with the Department of Computing, Data Science Institute, Imperial College London, U.K. His research interests include security, privacy, and trust for IoT, blockchain, personal data management, fog, edge, and cloud computing.

Kai Sun received the B.Eng. degrees in computer science from the Harbin Institute of Technology and the University of Birmingham in 2009 and the M.Sc. degree and the Ph.D. degree in computing from Imperial College London in 2010 and 2014, respectively. From 2014 to 2017, she was a Research Associate with the Data Science Institute, Imperial College London. She is currently the Lab Manager of the HNA Centre of Future Data Ecosystem. Her research interests include translational research management, network analysis, and decentralised systems.

Gyu Myoung Lee received the B.S. degree from Hongik University and the M.S. and Ph.D. degrees from the Korea Advanced Institute of Science and Technology (KAIST), South Korea, in 1999, 2000, and 2007, respectively. He is currently a Reader with the Department of Computer Science, Liverpool John Moores University, U.K. He is also an Adjunct Professor with KAIST. His research interests include future networks, IoT, and multimedia services. He has actively contributed to standardization in ITU-T as a Rapporteur, oneM2M, and IETF. He is the Chair of the ITU-T Focus Group on data processing and management to support IoT and smart cities and communities.

Yike Guo received the B.Sc. degree in computing science from Tsinghua University, China, in 1985, and the Ph.D. degree in computational logic from Imperial College London in 1993. He is currently a Professor of computing science with the Department of Computing, Imperial College London, and the Founding Director of the Data Science Institute, Imperial College London. His research interests are in the areas of data mining for large-scale scientific applications, including distributed data mining methods, machine learning, and informatics systems. He is a member of Academia Europaeaa and a fellow of the Royal Academy of Engineering.