ESTABLISHING VERIFIABLE TRUST IN COLLABORATIVE HEALTH RESEARCH

ESTABLISHING VERIFIABLE TRUST IN COLLABORATIVE HEALTH RESEARCH

BY
ANDREW SUTTON, B.A.Sc.

A THESIS
SUBMITTED TO THE DEPARTMENT OF COMPUTING AND SOFTWARE
AND THE SCHOOL OF GRADUATE STUDIES
OF MCMASTER UNIVERSITY
IN PARTIAL FULFILMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF SCIENCE

Master of Science (2018) McMaster University
(Computing and Software) Hamilton, Ontario, Canada

TITLE:	Establishing Verifiable Trust in Collaborative Health Research
AUTHOR:	Andrew Sutton B.A.Sc. (Computer Science), McMaster University, Hamilton, Canada
SUPERVISOR:	Dr. Reza Samavi
NUMBER OF PAGES:	xiii, 68

Lay Abstract

Collaborative health research environments involve the sharing of private health data between a number of participants, including researchers at different institutions. The inclusion of AI systems as participants in this environment allows predictive analytics to be applied on the research data to provide better diagnoses. In such environments where private health data is shared among diverse participants, the maintenance of trust between participants and the auditing of data transformations across the environment are important for protecting the privacy of data contributors. Preserving the integrity of these transformations is paramount for supporting transparent auditing processes. In this thesis, we propose an architecture for establishing verifiable trust and transparency among participants in collaborative health research environments, present a model for creating tamper-proof privacy audit logs that support the privacy management of data contributors, and analyze methods for verifying the integrity of all logged data activities in the research environment.

Abstract

Collaborative health research environments usually involve sharing private health data between a number of participants, including researchers at different institutions. Inclusion of AI systems as participants in this environment allows predictive analytics to be applied on the research data and the provision of better diagnoses. However, the growing number of researchers and AI systems working together raises the problem of protecting the privacy of data contributors and managing the trust among participants, which affects the overall collaboration effort. In this thesis, we propose an architecture that utilizes blockchain technology for enabling verifiable trust in collaborative health research environments so that participants who do not necessarily trust each other can effectively collaborate to achieve a research goal. Provenance management of research data and privacy auditing are key components of the architecture that allow participants' actions and their compliance with privacy policies to be checked across the research pipeline. The architecture supports distributed trust between participants through a Linked Data-based blockchain model that allows tamper-proof audit logs to be created to preserve log integrity and participant non-repudiation. To maintain the integrity of the audit logs, we investigate the state-of-the-art methods of generating cryptographic hashes for RDF datasets. We demonstrate an efficient method of computing integrity proofs that construct a sorted Merkle tree for growing RDF datasets based on timestamps (as a key) that are extractable from the dataset. Evaluations of our methods through experimental realizations and analyses of their resiliency to common security threats are provided.

For my Mom and Dad
In memory of Gramma and Poppie

Acknowledgements

I would like to thank my supervisor, Dr. Reza Samavi, for his great support, assistance, and guidance that helped me successfully complete this thesis. Thank you for the enjoyable and memorable experiences throughout my graduate studies. I would also like to thank Dr. Thomas E. Doyle and Dr. David Koff for their helpful guidance and comments.

Contents

List of Figures

List of Tables

Definitions and Abbreviations

Definitions

Blockchain A distributed ledger that records immutable transactions between parties through a consensus protocol. A copy of the ledger is distributed and maintained among nodes on the network. Groups of transactions are batched together to form a block, and blocks of transactions are linked together to form a chain of blocks. Blocks are composed of a header that contains the block metadata, including the hash (see *Cryptographic Hash Function*) of the previous block (i.e., forming a chain of blocks), and a body containing a group of transaction records. A consensus protocol is used to verify transactions, generate blocks, and publish blocks to the blockchain.

Cryptographic Hash Function
A function that takes an arbitrary length input string (pre-image) and transforms it to a fixed length output string (hash value or *integrity proof*). A *cryptographic* hash function must support three properties: pre-image resistance (infeasible to modify data without changing the resulting hash value); second pre-image resistance (infeasible to find any two inputs that hash to the same output); one-way (infeasible to invert the hash).

Data Sharing Agreement
Legally binding contractual agreement between parties that want to share data. The contract outlines *who* can access and use the data, for *what* purpose (including *when* and *where*), *how* the data is managed while it is being used, and how the data should be disposed of at the end of the defined agreement period [82].

Integrity Proof
The output string from a cryptographic hash function; used

to verify the integrity of a data item. See *Cryptographic Hash Function*.

Linked Data Refers to a set of best practices for publishing and connecting structured data on the Web [8]. Data is published following the Linked Data principles: (i) use Uniform Resource Identifiers (URI) to name things; (ii) use HTTP URIs, so people can look up those things; (iii) provide useful information (using RDF, SPARQL) when someone looks up a URI; (iv) include links to other URIs [27].

Quad Store A database designed to store Linked Data named graphs (i.e., quads) and allow SPARQL queries to retrieve graph data.

RDF A graph-based data model designed to be used in highly scalable web contexts.

RDF Graph A set of RDF statements (see *RDF Statement*).

RDF Statement
Defined as the triple: *subject predicate object*. In an RDF graph, the statement is depicted by an arc between two nodes, where the arc travels from the *subject* node to the *object* node and is labeled by the *predicate* [63]. A triple can be extended to include a name to form a quad (i.e., named graph), defined as the quad: *name subject predicate object*.

Abbreviations

AI Artificial Intelligence

API Application Programming Interface

CA Certificate Authority

DSA Data Sharing Agreement

ECDSA Elliptic Curve Digital Signature Algorithm

EMR Electronic Medical Record

FHIR Fast Healthcare Interoperability Resources

HIS Hospital Information System

L2TAP	Linked Data Log to Accountability, Transparency, and Privacy
PII	Personally Identifiable Information
PKI	Public Key Infrastructure
RDF	Resource Description Framework
REST	Representational State Transfer
RSA	Rivest, Shamir, Adleman
SHA	Secure Hash Algorithm
SPARQL	SPARQL Protocol and RDF Query Language
STRIDE	Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, Elevation of Privilege
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
VM	Virtual Machine
W3C	World Wide Web Consortium

Chapter 1

Introduction

The era of immense data analytics is transforming healthcare with the ability to interpret large and diverse datasets to provide better diagnoses, manage complex diseases, improve efficiency of care, and generate aggregated data for Artificial Intelligence (AI) systems [74]. For instance, Alphabet, the parent company of Google, has launched a new initiative called Cityblock Health that will provide low income households with medical care by analyzing multiple datasets to determine where care is needed [74]. Furthermore, IBM Watson's cognitive computing platform uses multiple large datasets to generate predictive analytics for improved diagnoses. In both examples, individuals (or patients) are actively and continuously involved in contributing data to be fed to the AI systems and the research goal is achieved through an intense (sometimes real time) collaboration between humans (as the data contributors), diverse researchers from multiple disciplines (e.g., medicine, computer science, statistics) and AI systems. The active involvement of humans (or patients) in this type of emerging health research is different to their roles in the classical clinical trial type research where a limited number of patients are involved and the patient's connection to the research ends when the data collection step is over. For data analytics health research, a feedback loop among patients, researchers and AI systems needs to be maintained in order to improve the predictive model. However, a successful realization of such a *human-in-the-loop* paradigm for health research requires establishing verifiable trust among participants (including humans and AI systems). The following motivating scenario reveals the challenges in maintaining trust among participants.

The author of this thesis is part of a multi-disciplinary research team that is currently developing an active classifier to predict patients' cancer risk from accumulated medical imaging radiation exposure. The classifier is active since the machine learning model will be constantly refined as individuals contribute their data points on medical imaging and diagnosis. Patients at the point of care will benefit from the updated risk assessment model while they have a

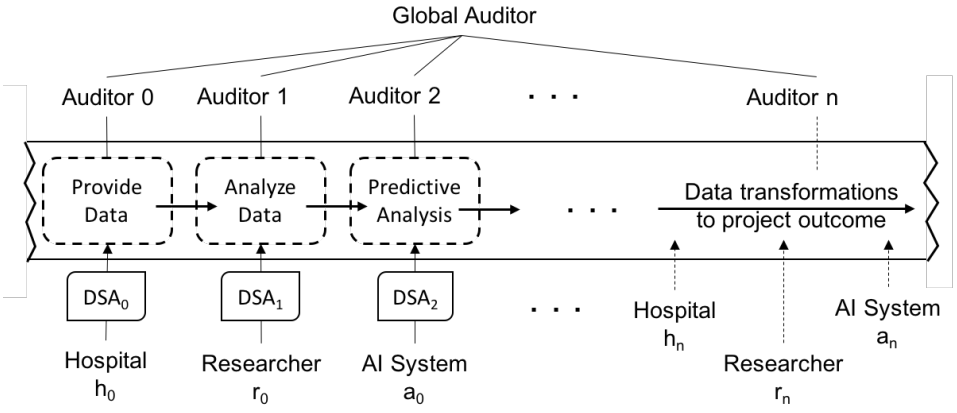


Figure 1.1: Collaborative Research Pipeline

chance to contribute their data for the consequent model refinement iteration. Such a complex collaboration on private data requires multiple Data Sharing Agreements (DSA) to be devised between diverse groups of researchers and hospitals as each participant may adhere to different privacy policies and jurisdictions. In addition, patients are in the loop of research since the AI system needs to be updated in near real time to produce accurate and relevant cancer risk assessment for each patient [45, 57].

Figure 1.1 conceptualizes the collaborative research pipeline for such a scenario, where hospitals $h_0, ..., h_n$ provide data for participants and multiple researchers $r_0, ..., r_n$ and AI systems $a_0, ..., a_n$ provide data analytics on the datasets. Hospitals continuously provide data as it is generated, and $DSA_0, ..., DSA_n$ govern access to the datasets in the research pipeline. Auditing the data lifecycle (from collection to use, disclosure and transformation) is essential in this scenario to ensure accountability. Typically, local and global auditors oversee the process [60], and the trust between different auditors and researchers is a presumption, however in reality there are conflicts of interests that may undermine the presumed trust [21]. In order to prevent conflicts of interests, transparency and accountability are required. However, transparency and accountability cannot be achieved without trustworthy auditing. Consequently, for trusted auditing, we need to ensure log integrity and participant non-repudiation. We focus on addressing the challenges of establishing verifiable trust among all participants in the research environment, preserving the privacy of all data contributors, and maintaining the integrity of all logged data activities for effective and trustworthy auditing.

1.1 Thesis Scope & Objectives

The objective of this thesis is to address the privacy and trust challenges that arise in collaborative health research environments, where private data is shared among multiple participants. When multiple researchers collaborate, the mismanagement of trust between actors affects the overall collaboration effort and hinders the ability to provide better diagnoses. We aim to achieve three objectives: (i) establishing *verifiable trust and transparency* in collaborative health research environments; (ii) supporting the *privacy management* of data contributors; and (iii) maintaining the *integrity of all data interactions* in the research environment.

In order to create a trusted research environment, we leverage blockchain technology. By using a permissioned blockchain (a blockchain that is controlled by authenticated participants as opposed to a public blockchain used for crypto-currencies) we support a collaborative environment that all data activities are transparent and can be verified by all participants. To protect data subjects' privacy and manage the data size, we define a concept of *data sharing transactions* where data is made accessible through data pointers that rely on institutional-based access control. Data pointers are stored on the blockchain and privacy policies related to each data item are captured. Transactions stored on the blockchain provide a tamper-proof trail of data while integrity verification can be performed and participants cannot repudiate their actions. We leverage theories and technologies stemming from blockchain technology [4, 66, 56, 16, 26], Linked Data graph signatures [34, 32, 30, 31], and Linked Data graph digest computation [30, 63] to create non-repudiable privacy audit logs and utilize the distributed and immutable properties of blockchain technology to make the audit logs tamper-proof. Furthermore, we exploit features in the datasets to create integrity proofs (cryptographic hashes) with desirable properties for auditing purposes.

Current approaches on protecting individuals' privacy and maintaining trust in collaborative research are based on data anonymization [35, 51] and provenance management, i.e., tracking the collection, use, disclosure, and transformation of data items throughout the research pipeline [85, 23]. These approaches are necessary, but insufficient when trust is not guaranteed between collaborators, especially when AI systems are also considered as participants in the process of data transformation. Importantly, the autonomous decision making process of AI systems needs to be captured and audited to demonstrate the trustworthiness of the predictive algorithms. Provenance and data anonymization methods can be supplemented by digitizing trust to mitigate the risk of collusion between auditors and researchers or inadvertent breaches of data usage by AI systems. Furthermore, approaches to generate integrity verification (cryptographic hash) for datasets are either based on incremental

methods or constructing different variations of a Merkle tree [46]. The properties of each method need to be investigated so that the most effective method can be used in the context of privacy auditing.

1.2 Thesis Contributions

The four main contributions of this thesis are that we:

1. propose a blockchain-based architecture to digitize trust in collaborative health research environments, where actors who do not necessarily trust each other can effectively collaborate to achieve a research goal;

2. present a Linked Data-based [27] model for creating tamper-proof privacy audit logs and provide a mechanism for log integrity and authenticity verification that auditors can execute in conjunction with performing compliance checking queries;

3. investigate the state-of-the-art methods of generating cryptographic hashes that can be used as integrity proofs for RDF datasets;

4. provide a more efficient method of computing a cryptographic hash for the special case of growing RDF datasets where the dataset statements carry some notion of ordering (e.g., statements are timestamped).

Abridged versions of the contributions described in this thesis have been published or have been submitted in [69, 70, 71, 72]. The author of this thesis is the lead author for each publication and received guidance from his supervisor and the other authors at McMaster University. Specifically, contribution 1 has been submitted to [72], contribution 2 has been published in [69] (nominated for best research paper and best student research paper awards) and submitted to [70], and contributions 3 and 4 have been published in [71].

1.3 Structure

The structure of the thesis is as follows. Chapter 2 presents an investigation of the related work. Chapter 3 describes the proposed blockchain-based architecture for collaborative health research. We provide an enhanced privacy audit log framework utilizing blockchain technology in Chapter 4. In Chapter 5, we discuss common metrics for integrity verification algorithms and provide a tree-based integrity proof algorithm that utilizes underlying data semantics. We conclude in Chapter 6 with some future directions for this research.

Chapter 2

Related Work

In this chapter, we present an investigation of the related work in three areas: (i) blockchain technology and its application in a healthcare domain (Section 2.1); (ii) methods for supporting non-repudiation and tamper-proof auditing (Section 2.2); and (iii) the current approaches to integrity preservation in auditing and digital signatures (Section 2.3).

2.1 Blockchain Applications in Healthcare

There have been numerous proposals for the use of blockchain in a healthcare environment [40, 43, 55]. McFarlane *et al.* [43] discuss a patient-centered peer-to-peer EMR supported by blockchain technology that provides an infrastructure for an interoperable health information exchange network and implements mechanisms for storing private information on an Ethereum-based blockchain. Use of blockchain for data sharing in clinical research has been investigated by Sotos *et al.* [65]. Sotos' solution treats data sharing as a series of transactions. Data sharing agreements are digitally signed and stored in the blockchain to make them public and unalterable.

Preliminary investigation of using blockchain for data sharing and collaboration has been performed in [37]. However, this approach focuses on investigating the issues related to mobile healthcare applications rather than the specific privacy issues related to collaborative health research. Zyskind *et al.* [86] present an automated decentralized access control manager that utilizes blockchain technology. Similarly, Zyskind's system has different types of transactions for user access control management and data storage and retrieval. Xu *et al.* [83] propose blockchain based access control that allows access policies and access histories to be stored on the ledger.

Additionally, Google's DeepMind Health Verifiable Data Audit project is developing a real-time auditing platform for health research data [68, 24, 18]. This approach uses an append-only tree-based ledger that records data usage

events. Similar to a blockchain, the ledger allows the integrity verification of data entries, however, the tree-based structure lacks some important properties that are inherent in blockchain technology, such as consensus and smart contracts, that contribute to preserving trust among participants. While this body of work considers the integration of blockchain into the healthcare domain, the approaches are limited to providing access control over patient data, rather than providing a mechanism to allow all participants in the environment to collaborate in a trusted and privacy-aware way. Chapter 3 intends to fill the gap in this research.

2.2 Tamper-proof Auditing

There are a number of approaches that provide a mechanism for verifying the integrity of an audit log [1, 67]. Butin *et al.* [11] address the issues of log design for accountability, such as determining what the log should include so auditors can perform meaningful a posteriori compliance analysis. Tong *et al.* [77] propose a method of providing role-based access control auditability in audit logs to prevent the misuse of data. Use of blockchain technology in the auditing of financial transactions have been investigated in [4] after the repercussions of the Enron Scandal in 2001 [36], where auditor fraud was the source of public distrust [66]. Anderson [4] proposes a method of verifying the integrity of files using a blockchain. Cucurull *et al.* [16] present a method for enhancing the security of logs by utilizing the Bitcoin blockchain.

These solutions focus on addressing the integrity preservation and verification of privacy audit logs and miss the non-repudiation aspect. There is a need for a practical solution for supporting both the non-repudiation *and* integrity of the logs. Chapter 4 intends to fill this gap by providing a model to create tamper-proof logs in a highly scalable Linked Data environment.

2.3 Integrity Preservation Techniques

As mentioned previously, Google's DeepMind Health Verifiable Data Audit project is developing an auditing platform for health research data that uses an append-only log to record events in a Merkle tree structure [25, 18]. This approach can be extended to capture the time of any data interaction, which can be utilized in the log tree structure such as in the method we propose in Chapter 5. Lindqvist presents a protocol for producing verifiable privacy-preserving membership proofs of Merkle trees [39]. In the context of audit logs, Lindqvist's protocol allows auditors to query the logs for audit proofs of log entries while preserving the privacy of unrelated entries. Since our timestamp tree approach, described in Chapter 5, is an extension of a sorted Merkle tree,

where the leaves are ordered, there is the potential of leaking information about adjacent leaves when querying the tree [39, 53]. Supplementing our approach with Lindqvist's protocol has potential for preventing undesirable data leakage.

Kleedorfer *et al.* [34] propose a Linked Data-based messaging system where conversations can be verified through digital signatures. This approach preserves the integrity of conversations through the chaining of message signatures where all messages and signatures are defined as RDF graphs. The hash of the graphs used for the digital signatures are calculated using incremental cryptography. However, this approach does not exploit the fact that the message graphs contain timestamp information for each message, which our approach in Chapter 5 can use to semantically preserve the order of the messages. Kasten *et al.* provide a framework for signing graph data where a number of graph hashing methods for the signatures are discussed [30, 32]. Similarly, this approach does not leverage available data in the graph datasets to construct a hash value to achieve sorted Merkle tree-like properties, which is desirable for preserving the data order. The research reported in Chapter 5 intends to supplement these methods.

Chapter 3

Digitized Trust in Human-in-the-Loop Health Research

In this chapter, we present our proposed blockchain-based architecture for enabling verifiable trust in collaborative health research environments. Our goal is to design an efficient architecture that protects the privacy of data that is dispersed among different organizations. We aim to support the human-in-the-loop paradigm for health research by establishing trust between participants, including human researchers and AI systems. Trust is established by making all data transformations transparent and verifiable by all participants. Section 3.1 provides an overview of our blockchain enabled trust architecture for collaborative health research. The security properties of the architecture are discussed in Section 3.2. A threat model evaluation and experimental realization are performed in Section 3.3. We conclude in Section 3.4.

3.1 Architecture

A trusted system means that all parties accept the actions of the system to be correct, the system's outputs to be true, and that the system will complete its expected task [64]. The trustworthiness of a system depends on the level of perceived trust of each system component [3]. Typically, trust is a subjective measurement based on the perception of how different parties evaluate each other and the systems they interact with. Achieving a trustworthy system requires transforming the notion of trust into an objective measurement. This transformation often relies on the use of a centralized trusted third party, where all collaborating parties trust this external entity (e.g., certificate authorities in public key infrastructures [73], arbitrated protocols [49]).

Establishing trust through a centralized third party is often the source of collusion, which threatens the trust, the very central notion that collaborative parties intend to establish. The Enron Scandal [36] is a textbook case when the trust placed in the centralized auditors was a major factor in the fraud. Rather than a centralized approach, we can leverage a distributed system to support trust between collaborating parties and alleviate the disadvantages that tend to threaten trust in centralized systems.

With the emergence of blockchain technology, the issues involved with a centralized trust system are alleviated through the use of distributed trust where all interactions are distributed, immutable, transparent and consensually agreed upon. Adapting blockchain from the typical crypto-currency use case to collaborative health research requires overcoming multiple challenges. First, transactions in health research involve immense data sizes (e.g., an entire health record with all medical imaging history [54]). Second, specific privacy statements must be captured for each individual data item, which might be applicable for all transactions. Therefore, blockchain adaptation for health research transactions requires careful investigation of the relationship between data and what is stored on the blockchain and the privacy of data subjects.

In this section we identify the classes of participants and their requirements when collaborating in a research environment as well as the properties required for a system to be trustworthy (Section 3.1.1). We then describe an architecture that supports blockchain enabled self-governed trust (Section 3.1.2), where specific components are detailed in Sections 3.1.3, 3.1.4, and 3.1.5, respectively.

3.1.1 Desiderata

Actors involved in collaborative health research include the data contributors (e.g., patients), data custodians (e.g., hospitals and research institutes), AI systems (e.g., active machine learning system [79]), researchers, and auditors. Patients are the source of the contributed data (e.g., subject of a CT scan) while data custodians store this data in their information systems. In an environment that involves the interaction of private health data among multiple actors, the data contributors want to know *who* is accessing and sharing their data, as well as *when* and *what* through some consent mechanisms. Data custodians are responsible for securely storing private health data while researchers require access to this data to perform analysis. Furthermore, AI systems also need to access patient data to update their predictive models. Finally, auditors monitor the system actors to determine compliance to privacy policies stated in DSAs and consent directives. The details of each actor type and their specific requirements are summarized in Table 3.1.

For an architecture to be trustworthy, certain properties need to be present.

Table 3.1: System Actors and Requirements

Actor Type	Requirements
Data Contributors	1) Provide consent regarding their data 2) Know who is accessing and sharing their data 3) Know when their data is being accessed and shared
Data Custodians	1) Remain in control of the data 2) Securely store private data 3) Comply with data protection and DSA policies and obligations
Researchers	1) Access patient data 2) Access AI System outcomes 3) Comply with DSA policies and obligations 4) Provide feedback to the AI system
Auditors	1) Monitor actor compliance
AI Systems	1) Comply with DSA policies and obligations 2) Provide analytical outcomes for researchers

Security properties that support trustworthy systems are grouped into six domains: confidentiality, integrity, availability, authentication (access control), non-repudiation, and accountability (transparency) [3]. Confidentiality and integrity define mechanisms that prevent the unauthorized reading and writing of data, respectively. Mechanisms such as encryption, digital signatures, and cryptographic hash functions support these properties. Ensuring that resources are accessible when required by authorized users falls under the availability category. Our proposed architecture relies on external mechanisms to support the availability of the system and is therefore considered out of scope for this research. Authentication and access control provide processes for verifying the identity of an entity and maintaining the entity's privilege across the system, respectively. The use of digital signatures and cryptographic hash functions support non-repudiation by creating undeniable evidence that an action has occurred. Finally, accountability and transparency, an extension of non-repudiation, capture non-compliant users and support the maintenance of user privacy through the use of logging and auditing. An ideal trustworthy system aims to support all six domains and properties to establish digitized trust. Especially in a health research environment that deals with private patient data, our proposed architecture must support these trust properties to provide a trustworthy system.

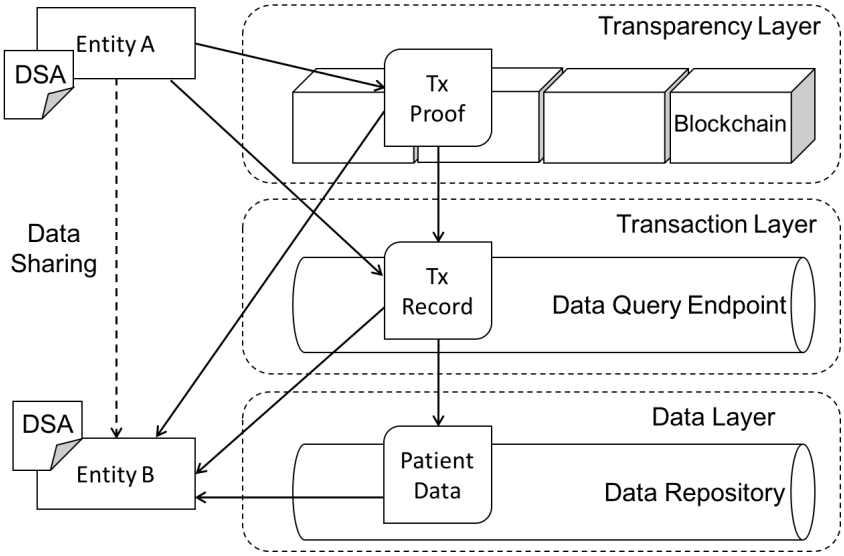


Figure 3.1: Architectural Layers

3.1.2 Architectural Overview

Given the requirements of each class of actors (Table 3.1) and the trustworthy system requirements, our architecture should support three main functionalities: provenance management of research data, privacy management of data subjects, and distributed and verifiable trust among participants. We present a layered architectural approach, as shown in Figure 3.1, with three layers to support data transactions, privacy and trust. At the bottom is the *data layer* responsible for generating data pointers that link to the medical records and can be shared among actors. Data pointers allow for the data custodians to remain in operational control of the data and provide their institutional-based access control mechanisms to protect the data. The middle layer is the *transaction layer* responsible for providing a mechanism for storing and querying data sharing transactions, including provenance and privacy information. At the top is the *transparency layer* responsible for distributed trust between all participants by allowing all data transactions to be transparent among all participants. A layered approach has multiple advantages. The functionality and the applied technology can be decoupled so that connections between components are clearly defined and a component does not rely on the internal logic and the technology used in other components. For example, a permissioned blockchain can be used as a plug-in for the transparency layer without changing any blockchain properties. The relationships between layers and how the layers work together are described below. The internal specifications of each layer are described in subsequent subsections.

The collaborative research pipeline (Figure 1.1) involves multiple actors

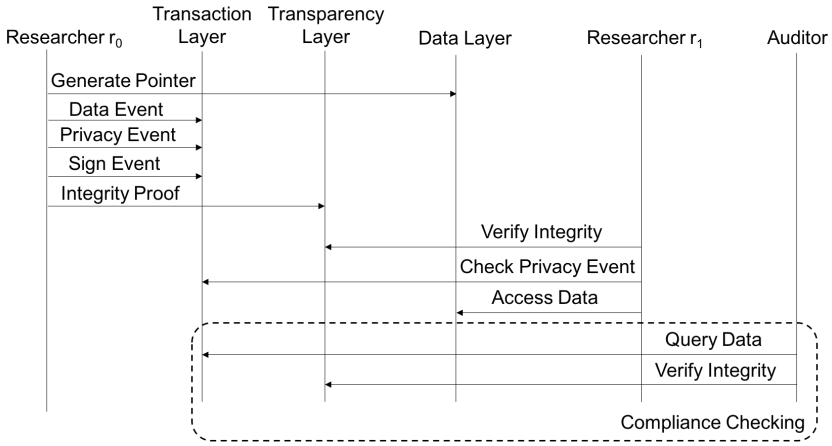


Figure 3.2: Data Sharing Transaction Sequence

interacting with each other and sharing private health data. The sequence diagram in Figure 3.2 depicts a dynamic view of the architecture in Figure 3.1 and demonstrates an instance of the research pipeline where data is being shared between two researchers (r_0 and r_1). DSAs are established between the researchers that outline the privacy policies that the researchers must abide by when using the data. When performing a data sharing transaction, the sending entity (r_0) first interacts with the data layer to generate a pointer to the data that they want to share. The data accessible by the pointer is stored in the data layer's data repository, which is hosted at a hospital or research institute. After researcher r_0 has generated the data pointer in the data layer, they construct a data sharing transaction, which is composed of a data sharing event, privacy event, and signing event. These events provide the transaction metadata (including the pointer generated in the data layer), privacy policies related to the data, and a digital signature of the transaction. The data sharing transaction is stored in a data query endpoint so that other actors can query the transactional data. An integrity proof (i.e., cryptographic hash) of the data sharing transaction is computed and written to the transparency layer so that all participants are aware of the transaction.

After the data sharing transaction has been completed by researcher r_0, researcher r_1 must first query the transparency layer to verify the integrity of the transaction. Integrity verification consists of recomputing the integrity proof of the data sharing transaction and comparing it to the integrity proof in the transparency layer. If both integrity proofs match, researcher r_1 can proceed to accessing the data through the data pointer specified in the data sharing transaction. Alternatively, if the integrity proof verification fails, researcher r_1 should not access the data and an auditor can perform further investigation. Researcher r_1 then queries the transaction layer to determine

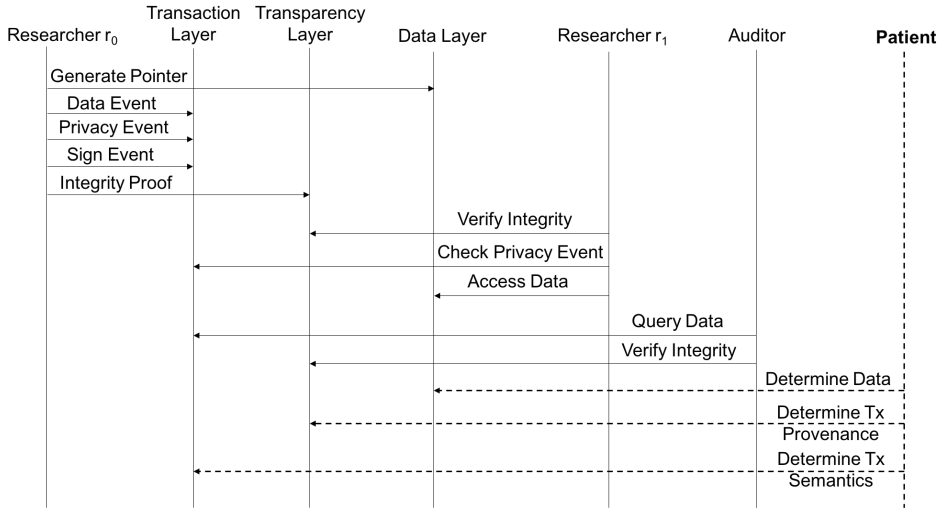


Figure 3.3: Active Involvement of Patients in the Research Process

the privacy policies they must abide by when using the data and then accesses the data through the data pointer at the data layer. Finally, auditors query the transaction and transparency layers to check the compliance of all participants with the policies in the governing DSAs.

A common use case scenario for this architecture involves patients, as data contributors, actively participating in the research process and auditing the use of their data. For example, suppose we have a research team rt_1 who has deployed an AI system (machine learning model) that provides a patient, p_i, cancer risk assessment based on p_i's accumulated exposure to low dose radiation from medical imaging (X-ray, CT scans, etc.). Over p_i's lifetime, their medical imaging data is stored in hospital information systems (data custodian) and as new imaging data is captured, the data is added to p_i's health record. In order to train the AI system to provide accurate and relevant cancer risk assessment, the researchers need to access the imaging data for all patients, including p_i. Additionally, the AI system must continuously consume new imaging data for each patient, as new data becomes available. Typically, in such a scenario, ethic review boards (ERB) review the research team's proposal for accessing private health data held by data custodians and grant approval after challenges relating to data privacy and security are addressed. Upon granting approval, DSAs are established to define how the researchers can share, access, and use the data, and the research project begins.

Traditionally, patients were not actively involved in the research process after contributing their data and there was a disconnect between patients and the researchers. However, with our architecture, patients can be actively involved in the research process where they can selectively determine what data is being shared and with whom the data is being shared with. Importantly,

they can also transparently see the provenance of their data across the research pipeline and determine how their data is being used. Figure 3.3 extends the sequence diagram of Figure 3.2 to include a symbolic user (i.e., anyone in the research environment, including the auditor). In this realization scenario, we consider the patient to be the symbolic user who wants to know what is happening to their data throughout the research process. Upon commencement of the research project, patient p_i can be active in the research process by enrolling in the transparency layer's blockchain network. When research team rt_1 shares patient p_i's data with the AI system or another research team rt_2, the data sharing transaction is proposed to the blockchain network participants. Patient p_i is enrolled in the network, so they can consent to the transaction if they agree with the usage of their data. If patient p_i does not consent to how their data is being used or who it is being shared with, they can reject the transaction and inform the research team to withdraw their data from the project. Confirmation from the patient p_i (confirmation can also include agreement from other actors such as data custodians) means that the data sharing transaction is recorded on the blockchain and the data records can be shared. As depicted in Figure 3.3, patient p_i can query each layer of the architecture to determine specifically what data records are being shared between researchers (data layer), determine data sharing transaction provenance to see who is accessing and sharing their data (transparency layer), and determine the semantics of the data sharing transaction, including privacy policies and obligations associated with their data (transaction layer).

3.1.3 Data Layer

The data layer acts as a data repository where the data pointers that are shared among actors reference the actual data records. We leverage the emerging Fast Healthcare Interoperability Resources (FHIR) standard to serve as our data pointers. A set of modular components called Resources are at the core of the FHIR framework [28]. Resources represent healthcare concepts, such as patients, providers, medications, and diagnostics. Each resource has a unique URL (uniform resource locator) and can be retrieved and manipulated through these URLs. In our architecture, authorized actors access the data using the FHIR URLs. By using FHIR we can not only support local hospital access control mechanisms but we also only maintain the pointers to data (or their hash values) in other layers instead of the actual data.

3.1.4 Transaction Layer

After the sending entity has generated the data pointer in the data layer, a data sharing transaction is generated by the transaction layer. This transaction is composed of a data sharing event, privacy event, and signing event as shown in Figure 3.2. We leverage a graph data model to represent the data sharing transaction events as it provides generalizability and flexibility [27], but our approach is data model agnostic and any other data models (e.g., relational data models) can also be used. We define Linked Data named graphs [13] for each type of event. The graphs are stored in an externally accessible data query endpoint, such as a quad store or SPARQL endpoint, so that they can be queried by other authorized actors in the system. Listing 3.1 represents the data sharing transaction and records the metadata of the transaction, including the sender, receiver, and timestamp (lines 4-7), the data pointer (FHIR URL) that is going to be shared (line 8), and the related privacy event graph (line 7). The transaction layer supports two types of transactions, depending on the purpose and initiator of the transaction (line 3). $T_{pointer}$ transactions specify that the transaction is sharing data pointers referencing patient data and are performed by the actors that store, generate, or manage the actual data (i.e., researchers, data custodians). The $T_{pointer}$ transactions can be queried by AI systems and by following the data pointer, the referenced data can be retrieved to update the AI system's predictive algorithm. $T_{outcome}$ transactions are performed by AI systems and specify that the transaction is sharing the results computed by their algorithms. The differentiation between $T_{pointer}$ and $T_{outcome}$ transactions allows actors to track transactions relating to data sharing and AI system predictive outcomes over the the course of the research pipeline, as well as provide feedback for the AI system improvement.

```
1 _:data-sharing-graph {
2 _:data-sharing-graph a https://example.com/DataSharingTransaction ;
3   _:hasTxType "pointer"^^xsd:string ;
4   _:hasTimestamp "1517263159399"^^xsd:string ;
5   _:hasSender "McMaster␣University␣Medical␣Center"^^xsd:string ;
6   _:hasReceiver "Juravinksi␣Hospital"^^xsd:string ;
7   _:hasPrivacyEvent _:privacy-graph-header ;
8   _:hasFHIRURL http://fhir.example.com/DiagnosticReport?id=system%7C00003. }
```

Listing 3.1: Data Sharing Event Graph

To support accountability and transparency, a privacy event captures promised and performed privacy acts, such as expressing privacy policies, requesting access, and defining data usage and obligations. For example, the Linked Data Log to Accountability, Transparency, and Privacy (L2TAP) framework [61] can be used to generate Linked Data privacy events. Listing 3.2 is an example of an L2TAP privacy event, which consists of a header that asserts provenance semantics (lines 4-8), and a body that asserts privacy semantics (lines 9-16).

This privacy event is an example of an access request (line 10) that contains properties for specifying the requested data item(s) (line 14), the purpose of access (line 15), and the data sender and requester (lines 12 and 11, respectively). L2TAP privacy events can also provide assertions of requested privacy privileges (omitted); more details can be found in [61, 69] and the extended triple set can be found in Section A.3 of the Appendix.

```
1 @prefix l2tap:<http://purl.org/l2tap#> .
2 @prefix scip:<http://purl.org/scip#> .
3 @prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
4 _:privacy-graph-header {
5   _:logevent a l2tap:PrivacyEvent ;
6     l2tap:eventParticipant _:researcher-2 ;
7     l2tap:receivingTimestamp "2018-01-29T12:00:00Z"^^xsd:dateTime ;
8   ... }
9 _:privacy-graph-body {
10  _:requests-req1 a scip:AccessRequest ;
11    scip:dataRequestor _:researcher-2 ;
12    scip:dataSender _:researcher-1 ;
13    scip:dataSubject _:patient-1234 ;
14    scip:requestedDataItem _:patient-1234-CTScan ;
15    scip:requestedPurpose _:purposes-treatment ;
16  ... }
```

Listing 3.2: Privacy Event Graph

A signature event graph (Listing 3.3) is used for integrity verification and provides participant non-repudiation so that data sharing actions cannot be denied. A data sender's digital signature of a data sharing and privacy event graph is captured in the signature event (line 6). The signer's public key used to verify the signature can be obtained through the signer's WebID [62] in line 5. The signed data sharing and privacy event graphs are referenced in line 7. Information describing how to verify the signature is also asserted in the graph, such as signing algorithms used (the full set of triples can be found in Section A.1 of the Appendix). The specific algorithm for computing digital signatures for graphs is described further in Chapter 4.

```
1 @prefix sig: <http://icp.it-risk.iwvi.uni-koblenz.de/ontologies/signature.owl#> .
2 _:sig-graph {
3   # graph signing methods omitted
4   _:sig-graph a sig:Signature ;
5       sig:hasVerificationCertificate <signer/WebID/URI> ;
6       sig:hasSignatureValue "BC3A14C44DEBF82A..."^^xsd:string ;
7       l2tap:hasSignedGraph _:privacy-graph-header, _:data-sharing-graph . }
```

Listing 3.3: Signature Event Graph

3.1.5 Transparency Layer

The underlying technology used at the transparency layer is a blockchain. Blockchain technology is a suitable candidate to operate at the transparency layer by providing mechanisms to record tamper-proof data transactions among multiple untrusted actors through its distributed consensus network [47, 6, 84, 55]. A blockchain is a decentralized database composed of a continuously increasing amount of records, or blocks, that represents an immutable digital ledger of transactions [56]. Distributed ledgers allow for a shared method of record keeping where each participant has a copy of the ledger, meaning that a majority of participants (i.e., nodes on the network) will have to be in collusion to modify the records in the blockchain. Each record, or block, in the blockchain is composed of a header containing a cryptographic hash of the previous block (forming a chain of blocks) and a payload of transactions. Blockchain is the technology behind the popular Bitcoin crypto-currency [50], where the blockchain provides a secure and consensus-driven record of monetary transactions between participants on the network. Similar to how Bitcoin leverages blockchain, our architecture leverages a blockchain to provide transparent and tamper-proof data sharing transactions. However, unlike the popular use of blockchain for crypto-currencies (e.g. Bitcoin [50]), which is public in nature, our blockchain network is private, or permissioned, since we are dealing with personal health information and the network participants are known (i.e., the network is composed of the actors in the collaborative research environment). Since the network is permissioned, we forgo the computationally expensive cryptographic consensus protocol used in public blockchain networks, and leverage participant signature-driven consensus protocols instead.

A data sharing transaction that is generated in the transaction layer is hashed using a cryptographic hash function to generate an integrity proof of the data sharing transaction (i.e., data sharing event, privacy event, signing event graphs). There are numerous methods for computing a digest of Linked Data graphs (e.g., [44, 12, 63, 22]), but we use the incremental cryptography approach in [63] since it provides an efficient runtime (Chapter 5 provides metrics for evaluating digest computation methods for Linked Data graphs). Incremental cryptography produces an integrity proof of Linked Data graphs by hashing each statement in the graphs and using a commutative operation (e.g., multiplication) modulo a large prime number to merge the statement hashes into an integrity proof. Formally, $integrityProof = \Pi_{i=0}^{n} h(s_i) mod(p)$ where n is the number of statements in the graphs, h is a cryptographic hash function (e.g., SHA-256), s_i is a graph statement, and p is a large prime number.

The data sharing transaction integrity proof is stored on the blockchain to have an immutable record of the transaction and for actors to be aware of the transaction. To store the integrity proof on the blockchain, we must define a

specific transaction for the network. A transaction in the transparency layer involves generating and storing a tuple $t = (integrityProof, sender, receiver, tx\ Type)$ on the blockchain. The tuple serves as a tamper-proof record of the data sharing transaction and is composed of an *integrity proof* of the data sharing transaction, the *sender* and *receiver* of the transaction (e.g., researcher, AI system), and the *transaction type* (i.e., $T_{pointer}$ or $T_{outcome}$).

The transparency layer supports *human-in-the-loop* functionality, even for patients as the data contributors in the collaborative health research environment. Inherent to blockchain technology, all participants in the network contribute and maintain the ledger of transactions. Therefore, all participants can audit data sharing transactions by verifying the integrity proofs on the blockchain and querying privacy events in the transaction layer to determine compliance and adherence to DSA policies and consent directives. We also leverage smart contracts (programs that run on a blockchain network and all participants can interact with) to encode and enforce DSA constraints that are part of contractual obligations.

3.2 Privacy, Security & Trust

In this section, we discuss how our architecture addresses important privacy and security properties in the health research domain. We then summarize the trustworthiness of the architecture in terms of our defined trust requirements.

Accountability and Transparency. Information accountability is an important aspect of privacy protection and has three main characteristics: validation (verifies a posteriori if tasks were performed as expected), attribution (finding non-compliant participants), and evidence (supporting information of non-compliant acts) [69, 14]. In order to support accountability, our architecture captures privacy events in the transaction layer that records deontic modalities such as privacy policies, purpose of data usage, obligations, and data access activities as described in [61]. Although the privacy events do not provide *enforcement* of the participants' performed acts and does not guarantee the accuracy of the reported actions, the privacy events provide a mechanism to express actions that can be effectively audited. The transparent flow and exchange of information in health research is an important factor in determining the compliance of participants in the research environment. Our architecture supports transparency through the use of blockchain technology to allow all participants to query *when*, *what*, by *whom* and *why* data is being shared in the entire data lifecycle. The transparency layer supports the concept of human-in-the-loop by allowing all participants to actively audit the sharing of private health data.

Confidentiality and Integrity. The confidentiality of private health data

is achieved through the use of encryption. Since our architecture relies on the sharing of data pointers, the actual data records do not leave their secure storage in data repositories at hospitals and research institutes. The data accessible through the pointers are encrypted at rest in the repositories. For example, the data repositories leverage a public key infrastructure (PKI) to encrypt the data with the data sharing recipient's public key, where only the recipient can decrypt the data using their private key. The integrity of the data sharing transactions is guaranteed by the integrity proofs that are stored on the blockchain in the transparency layer. Since the blockchain provides a tamper-proof storage mechanism for the integrity proofs, the integrity proofs in the blockchain can reliably be used for data sharing transaction integrity verification purposes.

Authentication and Access Control. To support authentication, the transparency layer utilizes a permissioned blockchain network that allows only authenticated participants (e.g. patients, researchers, hospitals) whose identity is verified through a PKI to transact and query the blockchain. A PKI allows our network to leverage a multi-signature consensus mechanism where n out of m (where $n \leq m$ and $n > 1$) signatures are required to validate transactions. Unlike proof-of-work consensus mechanisms [50] that are employed in public blockchain networks, a signature consensus only requires a majority of participant signatures to determine valid transactions and write data to the blockchain. Our architecture employs access control mechanisms to prevent unauthorized users from accessing private health data. Specifically, the access decision for participants who can perform data sharing transactions (e.g., write transactions to the transaction layer's endpoint or generate a data pointer in the data layer) is determined at the respective layers through mechanisms such as keys, certificates, tokens, passwords, and institutional access control mechanisms. Data sharing transaction types are limited to specific users, for example, $T_{outcome}$ transactions are only performed by AI systems and only authorized users, such as researchers, can view the results, whereas $T_{pointer}$ transactions can be performed by any user (e.g., researcher or AI system). Since each network participant can be identified through digital signatures, participant authentication and transactions can be verified.

Trustworthiness. To determine the trustworthiness of our proposed system, we must examine how each of the architectural components addresses the six trust requirements outlined in Section 3.1.1. Our proposed architecture is composed of three modular layers, transparency, transaction, and data, so we examine each of the layers in terms of each trust property.

The transparency layer's key feature is the use of a permissioned blockchain, which means this layer must support all six properties. In terms of confidentiality and integrity, the blockchain supports encrypted data over the network

Table 3.2: System Trustworthiness Summary

Architectural Component	Conf.	Integrity	Auth.	Non-rep.	Acc.	Avail.
Transparency Layer	✓	✓	✓	✓	✓	Out of Scope
Transaction Layer	✓	-	✓	✓	✓	Out of Scope
Data Layer	✓	-	✓	-	-	Out of Scope

and provides an immutable digital ledger of transactions. A permissioned blockchain network supports authentication and access control through network entity identification (i.e., digital signatures, certificates). All transactions on the network are digitally signed to achieve non-repudiation. A blockchain supports the transparency and accountability of participants on the network since all data interactions are stored on the blockchain and all transactions are verified through a consensus protocol. Furthermore, all data transformations and interactions across the research pipeline can be audited by the blockchain network participants.

The transaction layer operates as a data query endpoint, which provides confidentiality through data at rest encryption services. Only authenticated and authorized users can interact with the transaction layer to store data relating to data sharing events. Non-repudiation is achieved since all data sharing events stored in the transaction layer are digitally signed by the event generator. All data sharing events capture the provenance and privacy information relating to each individual data sharing event to capture the accountability of all actors participating in the sharing of data. Data sharing event integrity is not directly supported in this layer, rather the integrity proof of the transaction is preserved in the transparency layer.

The data layer offers data pointer generation services and provides the secure storage for the actual data records. By leveraging data pointers, we support the access control mechanisms enforced at local hospitals where the actual data records reside. Furthermore, the FHIR data pointer framework supports the authentication of users and provides role-based (RBAC) and attribute-based (ABAC) access control mechanisms [29]. The data pointer repositories also support confidentiality by providing encryption services to protect the data at rest. Similar to the transaction layer, the data layer does not directly achieve integrity, non-repudiation, and accountability, rather these properties are indirectly captured in the transparency and transaction layers.

Table 3.2 provides a summary of the system trustworthiness in terms of the six requirements for establishing trust. Although we achieve many of the requirements for establishing trust, there are some limitations with respect to

the trust properties. We discuss these limitations in the adversarial threat characterization of Section 3.3.1.

3.3 Evaluation

In this section, we first enumerate an adversary's capabilities when interacting with our architecture (Section 3.3.1). Then, we investigate our system's resiliency to common security threats using an industry standard threat model (Section 3.3.2). We then experimentally evaluate the realization of our architecture in Section 3.3.3.

3.3.1 Adversarial Threat Characterization

We classify the goals of an adversary as compromising the *confidentiality* or *integrity* of the private health data. An adversary compromises the confidentiality of the data by having unauthorized access to and reading plaintext data, whereas an integrity compromise refers to the malicious and unintended manipulation of the data. In terms of data confidentiality, our architecture relies solely on the sharing of data pointers rather than the actual private data records. Since we leverage FHIR as our data pointers, we rely on the institutional access control mechanisms to manage users' access rights when using the pointer to read the data. The data behind the pointer is encrypted at the data source. Although, the data pointer itself is not encrypted for auditing purposes, we can apply obfuscating techniques to the pointer so that no potentially private information is leaked through the URL. Since we utilize an immutable ledger in the form of a blockchain, all transactions have immutable integrity proofs stored in the blockchain. Attempts made by the adversary to perform retrospective modification of any transaction records will fail the integrity verification process in the transparency layer.

Based on the design of our proposed architecture, in a worst-case scenario, we assume that an adversary has three attack surfaces to potentially exploit: the transparency, transaction, and data layers. Since a blockchain serves as our transparency layer, the adversary would have to successfully become a participant in the blockchain network to potentially exploit attacks. However, we leverage a private (permissioned) blockchain network where participants' identities are known and participants are granted certificates for digital signatures through a certificate authority. Therefore, a more likely attack vector is from an insider adversary, where they have successfully enrolled in the network. In fact, in this case, the consensus mechanism leveraged by the network can prevent an adversarial participant in the network from exhibiting malicious behaviour since all transactions must be verified and accepted by

network participants. Majority-based consensus protocols give rise to a majority attack on the network, where 51% of the network must be adversarial to overcome the benign nodes, but these majority attacks are more common in public blockchain settings rather than private settings (due to the different consensus protocols used) [9].

At the transaction layer, although the confidentiality and authenticity of the transactions are supported, an internal attacker has the opportunity to generate and inject fake data sharing events (e.g., to possibly to hide non-compliant actions) into the system to subvert the verification process. Since collaborating participants perform data sharing transactions with each other, an internal attacker could create events that do not represent the true data transformation or activity that occurred (e.g., an adversary could create a fake and misleading privacy event). However, through retrospective auditing and the fact that all network participants are known, the adversary will be caught and identified by their digital signature (assuming the signing key was not stolen or the adversary is not masquerading as another entity). Furthermore, to be successful, an attacker would have to generate and sign a fake data sharing transaction, store the events in the transaction layer, calculate an integrity proof, and have the integrity proof transaction successfully verified and accepted by participants on the blockchain.

The data layer relies on the institutional-based protection mechanisms for preventing adversarial threats. Since the data layer stores the actual data records, it makes a prime target for an adversary to access private patient data. For this reason, we only interact and store pointers to this data in subsequent architectural layers so that hospitals and research institutes (i.e., data custodians) remain in operational control of their data and can apply their security policies to provide the safe and secure storage of data.

3.3.2 Threat Model Assessment

We assess our architecture using the Spoofing, Tampering, Repudiation, Information disclosure, Denial of Service, Elevation of privilege (STRIDE) threat classification methodology developed by Microsoft [48].

Spoofing. Since we leverage secure PKI, users cannot masquerade as another user (unless their private key has been exposed).

Tampering. Since we leverage a blockchain, the integrity proof stored in the blockchain cannot be altered. A data sharing transaction that is tampered with in the transaction layer can be detected by verifying the integrity of the transaction since there is an immutable integrity proof of the transaction stored in the blockchain.

Repudiation. All data sharing transactions are digitally signed by the initiator of the transaction, so users cannot deny the actions they have taken

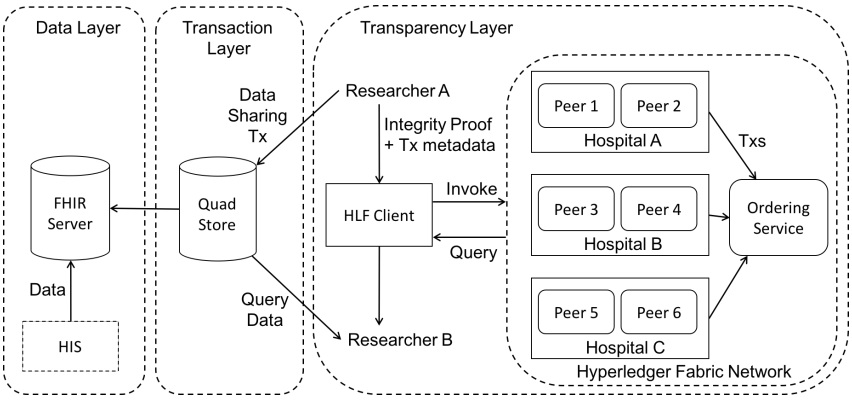


Figure 3.4: Architecture Realization

(assuming signing keys are not compromised).

Information disclosure. Since our architecture relies on sharing data pointers rather than actual data records, the likely-hood of a data record privacy breach is reduced. In the case of data leakage to unauthorized parties, the URL used to access the data can be terminated so that the data cannot be accessed through that URL. Access to the data at the URL is governed by secure access control mechanisms at the data source (password protection, one-time password generation, data is encrypted, etc.).

Denial of Service (DoS) and Elevation of privilege. These types of threats are out of the scope of our architecture and we rely on external mechanisms to address these threats.

3.3.3 Experimental Realization

We map our proposed architecture in Figure 3.1 to existing and emerging technologies to demonstrate the feasibility of such a system in a realistic collaborative health research environment. The technological realization is depicted in Figure 3.4. The data layer is mapped to a FHIR server from hospital information systems (HIS) that provide the data pointer services. A quad store with a SPARQL query endpoint (Virtuoso Universal Server [52]) is used for the transaction layer where the data transaction graphs are stored (and accessible through SPARQL queries). The transparency layer requires a permissioned blockchain network, so we utilized the Hyperledger Fabric blockchain platform [76].

We simulated a collaborative health research environment by running multiple virtual machines (VM), representing different actors, on a high performance cloud. Each VM is an Ubuntu 16.04.3 LTS instance with 1 VCPU (Intel Pentium III Xeon processor at 2.3 GHz) and 4 GB of memory. In particular, our Hyperledger Fabric blockchain network (v1.0.6) is composed of

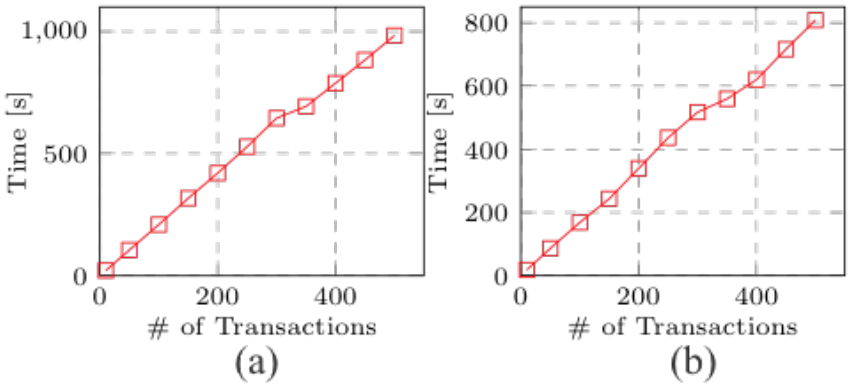


Figure 3.5: Elapsed Execution Times For a) Data Sharing Transaction Generation and b) Integrity Verification

three organizations (representing research institutes or hospitals) with each organization running two peers, as shown in Figure 3.4. The Fabric network also has a dedicated VM running as the network orderer and is attached to a Kafka-Zookeeper ordering service (for providing efficient transaction and block ordering). Hyperledger Fabric is capable of running smart contracts, or chaincode, and we created a DSA chaincode that enforces simple DSA constraints. Specifically, we encoded data retention periods found in DSAs and the chaincode rejects transactions that fall outside of some date range asserted by the DSA. Hyperledger Fabric stores data in the blockchain as key-value pairs, so in the case of our tuple defined in Section 3.1.5, we map the integrity proof as the key and the remaining tuple elements as the value (represented as a JSON object). To realize the human-in-the-loop concept, we used a blockchain visualization dashboard (Hyperledger Explorer [75]) to invoke transactions and perform transaction- and block-level queries.

We performed two experiments to measure the scalability of the architecture as the number of transactions increases in the research environment using the architecture realization in Figure 3.4. The first experiment is from the perspective of actors that want to generate data sharing transactions and involves generating a data sharing transaction event (composed of the graphs in Section 3.1.4), storing it in the transaction layer, computing an integrity proof of the event (using incremental cryptography) and writing the integrity proofs to the blockchain (integrity proof computation methods are further described in Chapter 5). The second experiment is from the perspective of those who want to audit the transactions (i.e., verify transaction integrity) and involves querying the transaction layer for a data sharing transaction event, recomputing the event's integrity proof, querying the blockchain for the integrity proof and verifying the integrity. The elapsed execution time of both experiments is plotted in Figure 3.5. Each reported elapsed time is the average of three

independent executions. It can be seen that the graphs validate the linear time growth of both perspectives.

3.4 Conclusion

In this chapter, we presented our proposed architecture to support verifiable trust in collaborative health research environments. We demonstrated how we can leverage existing technologies to support secure data sharing transactions in a data intensive research environment. In particular, we utilized the distributed and consensus-driven nature of permissioned blockchain networks to facilitate data sharing transactions and allow all participants in the environment to audit the transactions. By capturing the provenance of all data transformations in privacy events and storing records of transactions on a blockchain, our architecture supports the transparency of all data sharing transactions. The blockchain's distributed network allows all participants to be involved in the research process. Furthermore, the human-in-the-loop research paradigm is supported by making all data transformations transparent and auditable. In the next chapter, we expand on the privacy auditing aspect of the architecture to incorporate tamper-proof integrity verification mechanisms and non-repudiation.

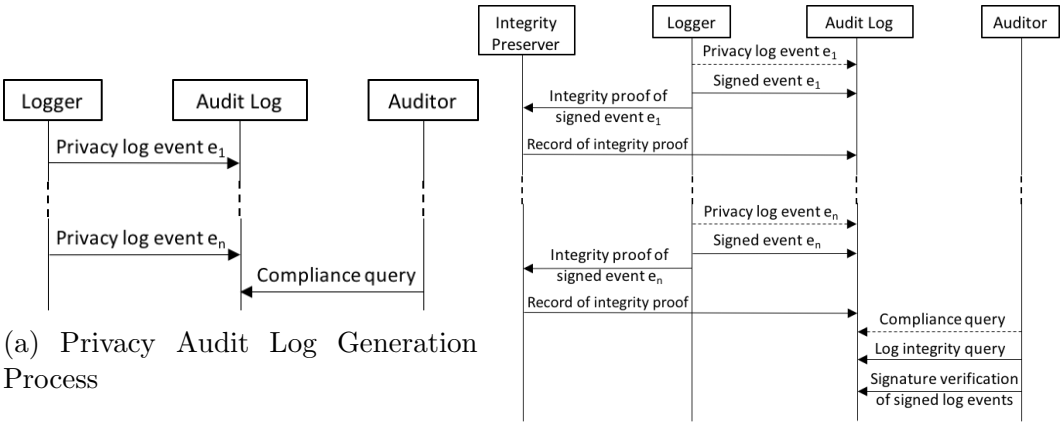
Chapter 4

Blockchain Enabled Privacy Audit Logs

As described in Chapter 3, to support accountability, we maintain the privacy of each data contributor through the use of privacy audit logs. However, collusion can occur between individuals in the organizations, such as researchers in the research teams, and the auditors to obfuscate or modify the integrity of the generated logs. The resulting degraded trust placed in the auditing process is a problem that needs to be solved in order to prove that organizations are responsible for privacy breaches resulting from non-compliant actions or to prove that they are compliant with the policies. In order to combat the potential modification of the logs due to collusion, a mechanism to provide tamper-proof audit logs is needed [4, 66]. This chapter describes a method that leverages blockchain technology to create tamper-proof audit logs that provides proof of log integrity and non-repudiation. Section 4.1 presents how privacy audit logs are generated and the design requirements for tamper-proof logs. Our solution to generate tamper-proof privacy audit logs is described in Section 4.2. Section 4.3 presents a SPARQL-based solution to perform log integrity verification. In Section 4.4, the results of an experiment to validate the scalability of our method is given. Concluding remarks are discussed in Section 4.5.

4.1 Properties of Tamper-proof Privacy Logs

Privacy auditing addresses three characteristics of information accountability: validation, attribution, and evidence [81, 14]. Validation verifies a posteriori if a participant has performed the tasks as expected, whereas attribution and evidence deal with finding the responsible participants for non-compliant actions and producing supporting evidence, respectively [81, 14]. To address these



(a) Privacy Audit Log Generation Process

(b) Tamper-proof Log Generation Process

Figure 4.1: Privacy Audit Log Generation Comparison

characteristics, privacy audit logs need to capture events with deontic modalities, such as capturing privacy policies, purpose of data usage, obligations of parties, and data access activities. A privacy audit log generation process is depicted in Figure 4.1a. The process is composed of a logger producing log events of promised and performed privacy acts and storing them in an audit log accessible to auditors. The logger generates multiple privacy log events (e_1 to e_n) over time (e.g., expressing privacy policies, requesting access and access activities). An auditor can then perform compliance queries against the audit log to determine if the performed acts are in compliance with the policies in the governing DSA (e.g., the scenario in Figure 1.1 and the sequence diagram in Figure 3.2) [60].

There are a number of proposals on logs for supporting privacy auditing [2, 11, 77]. In this research, we utilize the L2TAP privacy audit log because it provides an infrastructure to capture all relevant privacy events and provides SPARQL solutions for major privacy processes such as obligation derivation and compliance checking [60]. The L2TAP model follows the principles of Linked Data to publish the logs. By leveraging a Linked Data infrastructure and expressing the contents of the logs using dereferenceable URIs, the L2TAP audit log supports extensibility and flexibility in a web-scale environment [60]. In this research we extend the L2TAP ontology to support non-repudiation and log event integrity.

4.1.1 Tamper-proof Privacy Audit Log Desiderata

An event in a privacy audit log needs to be non-repudiable so that the performed act cannot be denied and the authenticity of the event can be provably

demonstrated. For example, in the scenario in Figure 1.1, if an auditor determines that the researchers have performed non-compliant actions, there is no provable method of holding the researchers accountable for their performed acts. Furthermore, after being logged, log events should not be altered by any participant, including the logger and auditor. If the researchers and auditors act in collusion to hide non-compliant acts in the log to avoid consequential actions, the resulting log does not represent the true events. Without a mechanism to provably demonstrate that the integrity of the log is intact, there will be a significant lack of confidence in the auditing process [4, 66]. The privacy audit log should enable the logger to digitally sign an event to support non-repudiation. The log should also offer a mechanism to preserve the integrity of log events (e.g., hashing or encryption). Verifying the signature of an event will prove the authenticity of the event logger. The ability to verify the integrity of the log events will result in a genuine audit of the participant's actions, since the performed actions (events) in the log are proven to be authentic.

Figure 4.1b depicts the additional steps required in the privacy audit log generation process to support event non-repudiation and integrity. The log is generated by the logger, but an additional entity, the integrity preserver, is required. After a log event is generated, the event must be signed by the logger to support provable accountability. Integrity proof digests (i.e. cryptographic hashes) of the log events should be generated and stored by the integrity preserver as the immutable record of the integrity proof. These records can then be retrieved to enhance the process of compliance checking with log integrity verification.

Besides the functionality described above, the tamper-proof privacy audit log should preserve the extensibility, flexibility and scalability of the underlying logging framework (i.e., L2TAP). We achieve flexibility through the Linked Data and SPARQL based solution for the log verification. The extensibility is addressed by a limited extension of the L2TAP ontology and using other external ontologies through the modular structure of L2TAP. As demonstrated in [60], the L2TAP privacy audit log is scalable. The additional verification processes introduced in this chapter to make the log tamper-proof should preserve the scalability.

4.2 Blockchain Enabled Privacy Audit Logs

In situations where a central authority has control over information resources, the trust placed in that authority to maintain correct and accurate information is reduced because there is no provable mechanism for external entities to verify the state of the resources. Blockchain technology solves the trust problem by maintaining records and transactions of information resources through a distributed network, rather than a central authority [33, 38]. The use of

blockchain technology to create an immutable record of transactions is analogous to the auditing problem we are trying to solve; the need for the immutable storage of information that is not governed by a central authority. In this section, we present how our blockchain enabled privacy audit log model works. We start with a brief background on the blockchain technology leveraged by our model, the Bitcoin blockchain, in Section 4.2.1. We describe the architecture of our model in Section 4.2.2. Sections 4.2.3 and 4.2.4 present the signature graph and block graph generation components of our model, respectively.

4.2.1 Bitcoin Blockchain

The Bitcoin system [50] is a crypto-currency scheme based on a decentralized and distributed consensus network. Transactions propagate through the Bitcoin peer-to-peer network in order to be verified and stored in a blockchain. Transactions are written to the blockchain through data structures that contain an input(s) and output(s). Monetary value is transferred between the transaction input and output, where the input defines where the value is coming from and the output defines the destination. The Bitcoin blockchain allows a small amount of data to be stored in a transaction output using a special transaction opcode that makes the transaction provably non-spendable [16]. Using the OP_RETURN opcode available through Bitcoin's transaction scripting language[1] allows up to 80 bytes of additional storage to a transaction output [5]. Changes to the state of the blockchain are achieved through a consensus mechanism called Proof of Work. Transactions are propagated through the Bitcoin network and specialized nodes, called miners, validate the transactions. These miners generate new blocks on the blockchain by solving a hard cryptographic problem and the other nodes on the network verify and mutually agree that the solution is correct. As more transactions and blocks are generated, the difficulty of the cryptographic problem rises, which makes the tampering of data written in the blocks very difficult. A blockchain explorer application programming interface (API) is required to query transaction information on the Bitcoin network. A blockchain explorer is a web application that acts as a Bitcoin search engine, allowing users to query the transactions and blocks on the blockchain [5]. We utilize this queryable special transaction to store an integrity proof of privacy audit logs on the Bitcoin blockchain.

4.2.2 Architectural Components

A blockchain is well suited to fill the role of the integrity preserver in the tamper-proof log generation process in Figure 4.1b. We use the capabilities

[1] `https://en.bitcoin.it/wiki/Script`

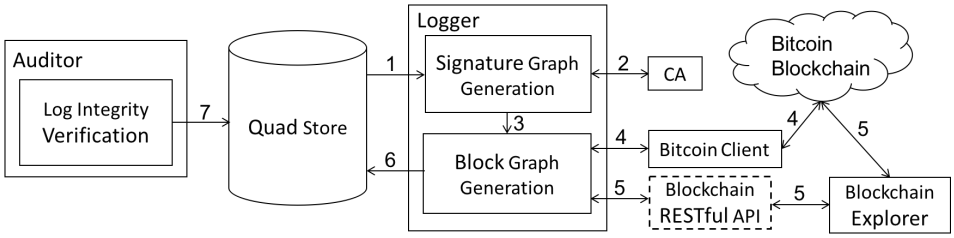


Figure 4.2: The Architecture of the Model

provided by the Bitcoin blockchain to store an immutable record of the log integrity proofs. The logger generates privacy log events and signs these events. After producing integrity proofs of the signed events, each of the proofs will be written to the Bitcoin blockchain through a series of transactions. The immutable record of the integrity proofs on the blockchain will be retrieved using a blockchain explorer. The components for signing log events and creating Bitcoin transactions are *signature graph generation* and *block graph generation* illustrated in Figure 4.2 and described below.

The signature graph generation component is responsible for capturing the missing non-repudiation property of the L2TAP audit log framework. An L2TAP audit log is composed of various privacy events such as data access requests and responses. The log events consist of a header that captures the provenance of an event and a body containing information about the event, such as what data is being accessed by whom. URIs are used to identify a set of statements in the header and body to form RDF named graphs stored in a quad store [13]. We generate a new named graph, called the *signature graph*, that contains assertions about the event's signature. The event that will be signed is pulled from the quad store and signed by the logger (flow 1). There needs to be a public key infrastructure (PKI) with certificate authorities (CA) in place where the logger has a generated key pair used for digital signatures (flow 2). The computed signature and signature graph will be passed to the block graph generation component to be part of the integrity proof digest computation (flow 3).

The block graph generation component conducts transactions on the Bitcoin network to write the integrity proof digest to the blockchain. The logger uses a Bitcoin client to create a transaction containing the integrity proof digest (flow 4). After the transaction is written to the blockchain, the transaction data is queried through a RESTful request [58] to a blockchain explorer API (flow 5). The queried data is parsed to an RDF named graph, called the *block graph*. The block graph contains the integrity proof digest and information identifying the block containing the transaction on the blockchain. After the block graph has been generated, it is stored in a quad store in order for an auditor to perform log event integrity and signature verification queries

(flows 6 and 7, respectively). Generating a block graph reduces the burden on the auditor when performing log integrity verification since all of the event integrity proofs are stored in a quad store. Without the block graphs, the auditor would have to search the *entire* Bitcoin blockchain for the integrity proof digests. Since the Bitcoin blockchain is a public ledger, there are many transactions unrelated to the auditor's search, which would make this method of searching inefficient. An alternative approach is to use a full Bitcoin client to download the entire blockchain, however in this case the required network bandwidth and local computing power are major limitations.

The signature graph and block graph generation components require two ontology modules to be added to the modular structure of the L2TAP ontology. The Signing Framework signature ontology [31] expresses all of the necessary algorithms and methods required for verifying a signature. The BLONDiE [78, 19] ontology semantically represents the Bitcoin blockchain. We also need to extend the L2TAP ontology to capture the signature graph and the signed log event. The new `hasSignedGraph` property in the L2TAP-participant module links the signature graph and signed event graph. An L2TAP log event body is dereferenced in the corresponding event header through the L2TAP `eventData` property. The signature graph just needs to reference the event header since there is an assertion between the body and header that the two graphs belong to the same event. The existing structure of L2TAP allows other components of the tamper-proof auditing to be asserted when a new log is initiated. For example, if a log uses Symantec[2] as the CA this can be included as a triple in the body of the log initialization event. The extended ontology is available on the tamper-proof audit log section of the L2TAP ontology website[3].

4.2.3 Signature Graph Generation

The process that the logger of an event has to take to compute a signature and generate a signature graph is formalized in Algorithm 1. The input parameters are the log event i header RDF graph, hg_i, body, bg_i, and the logger's private key, sk. Our algorithm follows the process of signing graph data in [32], which includes: canonicalization, serialization, hash, signature, and assembly [34, 32]. We can omit the canonicalization and serialization steps as we can assume our graphs are in canonicalized form and are serialized in the TriG syntax [7].

[2] `https://www.symantec.com`
[3] `http://l2tap.org`

Data: Event header graph: hg_i, event body graph: bg_i, private key: sk
Result: Signature graph: sg_i
1 $Triples \leftarrow (\text{extractTriples}(hg_i) \cup \text{extractTriples}(bg_i))$;
2 $Hash \leftarrow \prod_{j=1}^{|Triples|} h(t_j \in Triples) \text{mod}(p)$;
3 $Sig \leftarrow \text{Sign}(Hash, sk)$;
4 $sg_i \leftarrow \text{assembly}(Sig)$;
5 **return** sg_i

Algorithm 1: Signature Graph Generation Algorithm

The first step in Algorithm 1 is to compute the hash of the input event header, hg_i, and body, bg_i. We use incremental cryptography and the graph digest algorithm [63] to compute the digest of hg_i and bg_i. Since the ordering of triples in the RDF graph is undefined, the graph digest computation involves segmenting the input into pieces, using a hash function on each piece, and combining the results [63]. In line 1 we extract the triples from hg_i and bg_i into the set of triples, $Triples$, so that incremental cryptography can be performed on each triple. In line 2, a set hash over all of the triples in $Triples$ is computed using a cryptographically secure hash function (e.g., SHA-256) to produce a hash of each triple [63]. This triple hash is reduced using the modulo operation by a sufficiently large prime number, p (the level of security depends on the size of the prime number [63]). Each of the triple hashes are multiplied together, producing the $Hash$ value in line 2 as the resulting header and body graph digest. After constructing the graph digest, the logger generates a signature, Sig, by signing the digest using the Elliptic Curve Digital Signature Algorithm (ECDSA) in line 3. ECDSA uses smaller keys to achieve the same level of security as other algorithms (such as RSA), resulting in a faster signing algorithm. In the final step we generate the triples of the signature graph as a new named graph using the *assembly* function. The triples in this graph contain the signature value and algorithms for verifying the signature [32].

```
1 @prefix sig: <http://icp.it-risk.iwvi.uni-koblenz.de/ontologies/signature.owl#> .
2 @prefix l2tapp: <http://purl.org/l2tapp#> .
3 _:log-sig-1 {
4   # triples omitted describing graph signing methods
5   _:log-sig-1 a sig:Signature ;
6       sig:hasVerificationCertificate <signer/WebID/URI> ;
7       sig:hasSignatureValue "MEUCIQC44Qy2O8Mx..."^^xsd:string ;
8       l2tapp:hasSignedGraph _:log_h1 . }
```

Listing 4.1: Signature Graph

Listing 4.1 illustrates an example signature graph generated using Algorithm 1. Analogous to work presented in [34], we also use the Signing Framework signature ontology [31]. Lines 5-8 in this listing contain the signature

triples. Line 6 contains the WebID [62] where the signer's public key can be acquired [34]. The log event signature, Sig, is identified in line 7. Line 8 references the log event header that is signed. The signature graph also contains triples describing the algorithms required to verify the signature, which are included in the extended set of triples in Section A.1 of the Appendix.

4.2.4 Block Graph Generation

Algorithm 2 inputs an event's signature (sg_i), header (hg_i) and body (bg_i) graphs to compute and write an integrity proof digest to the Bitcoin blockchain and generate a block graph. Analogous to Algorithm 1, the triples are extracted from the input graphs into the set of triples, $Triples$ (line 1), so that incremental cryptography can be used to compute the integrity proof digest, H (line 2).

Data: Signature graph: sg_i, event header graph: hg_i, event body graph: bg_i
Result: Block graph: $BlockGraph_i$
1 $Triples \leftarrow$ (extractTriples(sg_i) $\cup$ extractTriples(hg_i) $\cup$ extractTriples(bg_i)) ;
2 $H \leftarrow \prod_{j=1}^{|Triples|} h(t_j \in Triples) mod(p)$;
3 Write H to Bitcoin blockchain (using Bitcoin client) ;
4 $md \leftarrow$ query block metadata (using blockchain API) ;
5 $BlockGraph_i \leftarrow$ assembly(md, H) ;
6 **return** $BlockGraph_i$

Algorithm 2: Block Graph Generation Algorithm

The next step is to create a Bitcoin transaction using a Bitcoin client[4] to write the integrity proof to the Bitcoin blockchain (line 3). An audit log requires one transaction per event. The Bitcoin client validates transactions by executing a script written in Bitcoin's transaction scripting language. The language provides the *scriptPubKey* output and the *scriptSig* input scripts[5] to validate transactions. A transaction in our model contains the OP_RETURN opcode in the *scriptPubKey* output ($scriptPubKey = OP\_RETURN + H$) and the logger's signature and public key in the *scriptSig* input ($scriptSig = signature + publicKey$). We store the integrity proof digest in the 80-byte data segment of the OP_RETURN transaction output. The transaction is propagated through the Bitcoin network for verification.

After the transaction containing the integrity proof digest has been stored on the Bitcoin blockchain, two queries are performed to retrieve the metadata

[4] `https://blockchain.info/wallet/#/`

[5] `https://en.bitcoin.it/wiki/Script#Provably_Unspendable.2FPrunable_Outputs`

of the transaction using the Bitcoin blockchain data API[6] provided by the *blockchain.info* blockchain explorer (line 4). The first query is an HTTP GET request to `https://blockchain.info/rawaddr/$bitcoin_address`, where *$bitcoin_address* is the logger's Bitcoin address used to create the transaction. JSON data is returned containing an array of transactions made from the specified Bitcoin address. The JSON data is parsed to find the block height of the block containing the integrity proof digest transaction. The block height is the number of blocks between the first block in the Bitcoin blockchain and the current block. The block height can be found in the transaction array using the transaction's *scriptSig* value. The second query is an HTTP GET request to `https://blockchain.info/block-height/$block_height?format=json`, where *$block_height* is the retrieved block height. This query returns the block metadata needed to assemble the block graph, such as the hash of the previous block and timestamp. This information is necessary to build a complete representation of the block and allow for the block graph data to be easily verified.

The final step in the algorithm is to use the *assembly* function to create a new named graph, called the block graph, that describes the metadata about the block containing the integrity proof digest transaction. Listing 4.2 illustrates an example of a block graph output by Algorithm 2, serialized in TriG. We use the BLONDiE [78] ontology to generate the triples in this listing. The object of each triple is populated with the values extracted from the *blockchain.info* queries. Lines 5-9 describe the integrity proof transaction. The *scriptSig* value is captured in Line 8 and the hash of the transaction in line 7. Line 9 holds the integrity proof digest of the event and signature graphs (in hexadecimal). This value is what an auditor will be querying when conducting log integrity verification. Additional triples that describe the block header and payload are omitted here. The full set of triples is included in the appendix (Section A.2).

```
1 @prefix blo: <http://www.semanticblockchain.com/Blondie.owl#> .
2 _:exlog-block-1 {
3  _:exlog-block-1 a blo:BitcoinBlock ;
4   # triples omitted describing block header and payload
5  blo:BitcoinTransaction blo:hasBitcoinTransactionInput blo:BitcoinTransactionInput ;
6     blo:hasBitcoinTransactionOutput blo:BitcoinTransactionOutput .
7  blo:BitcoinTransactionInput blo:hashBitcoinTransactionInput "1a2...3fc"^^xsd:string ;
8     blo:scriptSignBitcoinTransactionInput "4730440...41d6e6"^^xsd:string .
9  blo:BitcoinTransactionOutput blo:scriptPubkeyBitcoinTransactionOutput "6a2848...e46e65"^^
       xsd:string . }
```

Listing 4.2: Block Graph

[6] `https://blockchain.info/api/blockchain_api`

4.3 Log Integrity Verification

The goal of an auditor in a privacy auditing scenario is to check the compliance of participants' actions with respect to the privacy policies. The authors in [60] described a SPARQL-based solution for compliance checking; i.e. answering the question of, for a given access request and its associated access activities, have the data holders followed the access policies? This section describes our extended SPARQL-based solution to enhance the compliance checking queries described in [60] to include the integrity and authenticity verification of log events.

For a given L2TAP log, the process of verifying the log integrity and authenticity and compliance checking can be performed in a sequence; i.e., for all events in the log, first ensure the integrity and authenticity of all events and then execute the compliance queries for the interested access request. However, in practice this approach is not desirable as for a fast growing log, verifying the entire log for each audit query is very expensive (see our experiment in Section 5.3). Alternatively, we can devise an algorithm that verifies the integrity and authenticity of a small subset of the event graphs for a given access request. The L2TAP ontology provides compliance checking of a subset of events through SPARQL queries [59], which the following algorithm can leverage to reduce the runtime.

Data: Event header graph: hg_i, event body graph: bg_i, signature graph: sg_i
Result: Boolean verification value: v_i

$Triples \leftarrow (\text{extractTriples}(hg_i) \cup \text{extractTriples}(bg_i) \cup \text{extractTriples}(sg_i))$;
$H \leftarrow \prod_{j=1}^{|Triples|} h(t_j \in Triples) mod(p)$;
$URI \leftarrow$ Query block graphs for H (Listing 4.3) ;
if $URI \neq \emptyset$ **then**
  $v_i \leftarrow \text{verifySignature}(sg_i, hg_i, bg_i)$;
else
  $v_i \leftarrow$ false ;
end
return v_i

Algorithm 3: Verification Algorithm

Algorithm 3 formalizes the steps an auditor takes to verify the integrity of a log event and the event signature prior to checking compliance. The input parameters are the event header (hg_i), body (bg_i), and signature (sg_i) graphs, for an event i in the subset of events related to an access request. Assuming a cryptographically secure hash function is used to recompute the digest, any modification of the graphs will result in a different digest. If the search of the

block graphs is successful and the computed digest is found, then the log event must have remained unmodified [4]. Therefore, the first step in the algorithm is to recompute the integrity proof digest of the log event. We originally used incremental cryptography to calculate the integrity proof digest, so the same method must be used again for computing consistent digests. In lines 1 and 2, we first extract the triples of the input graphs and compute the integrity proof digest, H, similar to what we described in Section 4.2.4. The SPARQL query in Listing 4.3 is executed against the block graphs to find a matching digest in the `scriptPubkeyBitcoinTransactionOutput` relation (line 3). This query is parameterized with the integrity proof digest, H (`@integrityProofDigest`). If the query returns the URI of a block graph containing the integrity proof digest, we proceed to verify the signature in the signature graph (lines 4 and 5). Otherwise, if no matching value is found in the block graphs, we conclude that the integrity of the log event has been compromised.

```
1 PREFIX blo: <http://www.semanticblockchain.com/Blondie.owl#> .
2 SELECT ?g WHERE {
3   GRAPH ?g { ?s blo:scriptPubkeyBitcoinTransactionOutput @integrityProofDigest }}
```

Listing 4.3: SPARQL Query for Integrity Verification

The signature graph of the event can be found through the `hasSignedGraph` property. The algorithms used to verify the signature are extracted from the signature graph triples containing the hashing (i.e. SHA-256) and signing algorithms (i.e. ECDSA). The public key of the logger is retrieved by following the WebID URL in the `hasVerificationCertificate` property of the signature graph. If the signature verification process in line 5 fails, the algorithm returns `false`. In the case of no matching integrity proof digest or signature verification failure, the auditor will know which event has been modified and who the logger of the event is. However, the auditor will not know *what* the modification is, only that a modification *has occurred*. Therefore, proof of malicious interference would need further investigation.

Despite the process in this section supporting the confidentiality, authenticity and integrity of a privacy log, the approach is susceptible to an *internal* attack to subvert the verification process. However, to be successful, an attacker would have to generate and sign a *fake* log event, store the event in the quad store, calculate an integrity proof, store the proof on the blockchain, and finally generate a block graph pointing to the fake integrity proof block.

4.4 Experimental Evaluation

This section presents a scalability evaluation of our blockchain enabled privacy audit log model from the perspective of an auditor. In the experiment, we ran

our integrity checking algorithm on increasingly sized L2TAP privacy audit logs. Section 4.4.1 describes the synthetic audit log used in the experiment. In Section 4.4.2, we illustrate the details of the test environment. The results of the experiment are discussed in Section 4.4.3.

4.4.1 Dataset

To simulate the process of an auditor checking the integrity of an audit log, we generated synthetic L2TAP logs[7]. A basic log consists of eight events: log initialization, participants registration, privacy preferences and policies, access request, access response, obligation acceptance, performed obligation, and actual access. The actual contents of these events can be found in [60]. To create a larger audit log, we repeatedly generate the access request events. Our largest synthetic log, composed of 10,000 events, contains approximately 990,000 triples: 90 triples from the first three events of log initialization, participants registration, and privacy preferences and policies, and 989,703 triples generated from 9997 additional access request events where each access request leads to the generation of the five remaining events with a total of 99 triples.

The signature and block graph for each event needs to be generated for the auditor to perform the integrity verification procedure. A log containing 10,000 events would generate the same number of signature graphs, block graphs, and Bitcoin transactions. In this case, the total size of the dataset that the auditor would need to process is about 40,000 graphs. The initial state of the experiment is an audit log containing n events (composed of $2n$ header and body graphs) with n generated signature graphs and n generated block graphs. All of these graphs ($4n$) would be stored on a server in a quad store prior to measuring the scalability of the integrity verification solution. Figure 4.3 illustrates the log sizes used for the experiment, which range from a log containing 100 events to 10,000 events.

4.4.2 Test Environment

The experiment was run by executing the SPARQL queries on a Virtuoso [52] server and quad store deployed on a Red Hat Enterprise Linux Server release 7.3 (Maipo) with two CPUs (both 2 GHz Intel Xeon) and 8 GB of memory. The RDF graph processing and hash computations in Algorithm 3 were run on a MacBook Pro with a 2.9 GHz Intel Core i7 processor and 8 GB of memory. The Java method used to measure the elapsed execution time of the experiment is *System.nanoTime()*. The execution time is measuring the time difference between sending the queries to the quad store on the server

[7]Datasets are available in the figshare repository: `https://doi.org/10.6084/m9.figshare.5234770`

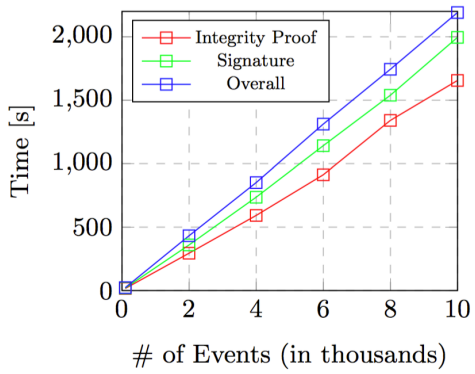
over HTTP and verifying the integrity proof digest and the signature. The recorded time does not take into account the time to generate the signature and block graphs (these were pre-computed before the experiment) or the time needed to write the data to the Bitcoin blockchain. To account for variability in the testing environment, each reported elapsed time is the average of five independent executions.

4.4.3 Experimental Results

In practice, an auditor would operate on a subset of events in the log based on the results from compliance queries for a given access request. We have opted to demonstrate a `worst-case` scenario by verifying the integrity and authenticity of an *entire* log rather than a subset of the events. This will also demonstrate the execution time of large subsets of events that are the size of the entire logs we conducted the experiment on.

The experiment consisted of retrieving all of the log events and their corresponding signature graphs from a quad store deployed on the server. Each set of graphs were input to Algorithm 3, which computes the integrity proof digest and executes the query in Listing 4.3 to determine if the integrity proof could be found in a block graph in the quad store. This procedure was executed on an audit log that contained a number of events ranging from 100 to 10,000, as shown in Figure 4.3. Figure 4.3 also illustrates the number of graphs that were input to the integrity proof digest computation in line 2 of Algorithm 3. A log consisting of 10,000 events requires 30,000 (10,000 header, 10,000 body, 10,000 signature graphs) graph digest calculations. A log of this size will generate 10,000 block graphs as well, which will need to be searched for the integrity proof.

The elapsed execution time is plotted in Figure 4.3. The graph illustrates the execution time of verifying the signature, computing and verifying the integrity of the events, and the overall process. The experiment validates the linear time growth for the entire integrity checking procedure. It can be seen that an increase of about 2000 events results in an increase of approximately 7 minutes to the integrity verification procedure. The reported results can be extrapolated to predict that a log containing an extreme case of one million events will take approximately 48 hours to perform an integrity check. This time is relatively small considering the vast amount of triples that would need to be processed from the event header and body, signature and block graphs ($>$ 100 million triples). The results of the experiment validate the scalability of our blockchain solution and demonstrates that the solution can perform efficiently at the task of verifying the integrity and signature of the audit log events.



# of Events	# of Graphs for Hashing	Execution Time (s)
98	294	23.19
1998	5994	432.15
3998	11994	851.28
5998	17994	1311.51
7998	23994	1743.78
9998	29994	2191.06

Figure 4.3: Elapsed Execution Times for Integrity and Signature Verification

4.5 Conclusion

In this chapter, we proposed an extended privacy audit log model that supports non-repudiation and integrity verification. We supplement an existing privacy audit log framework with capabilities to support non-repudiation through the use of Linked Data-based digital signatures. Loggers digitally sign all generated privacy events so that the actions they have performed cannot be denied and responsible participants for non-compliant actions can be identified. Integrity verification is supported through the use of integrity proofs that allow auditors to determine if privacy events in the log have been maliciously tampered with to hide non-compliant actions. We demonstrate how leveraging blockchain technology, in combination with event integrity proofs, can produce a tamper-proof log generation process. The algorithm we employed to generate our event integrity proofs is one of many integrity proof algorithms to choose from. In the proceeding chapter we investigate the methods for generating integrity proofs of datasets and provide a set of metrics that can be used to determine effective algorithms for different use cases. Furthermore, we propose an extension to an existing approach such that, in specific contexts such as privacy auditing, yields improved query runtime, supports random access, and preserves the order of data.

Chapter 5

Timestamp-based Integrity Proofs for Linked Data

In Chapters 3 and 4, we relied on existing methods of computing integrity proofs of RDF datasets. As for any data model, a desirable feature for RDF is having efficient methods to generate integrity proofs of RDF statements, particularly when the size of the dataset is incrementally growing. For example, the Linked Data based log designed for encoding privacy auditing, which we leveraged in Chapters 3 and 4, constantly grows and the integrity proof needs to be recomputed to guarantee that the log is tamper-proof and/or supports non-repudiation [69]. However, in certain contexts, such as privacy auditing, we can leverage the underlying data semantics to produce integrity proofs with specific properties.

Current approaches to generate integrity verification (cryptographic hash) for RDF datasets are either based on incremental methods where a commutative operation (e.g., concatenation or multiplication) is applied to the hash values of data items (e.g., individual RDF statements or RDF subgraphs) to produce a hash value for the whole dataset [44, 12, 63, 22] or based on constructing different variations of a Merkle tree [46], which is a rooted binary hash tree where nodes are labeled with cryptographic hashes and the root provides a hash for the whole dataset [46, 15, 17, 42]. The hash value of an RDF graph generated using each of the proposed methods carries different properties. For example, one method may generate the same hash value when the order of statements in a dataset changes and another might be order sensitive. The methods may also differ in their running time to generate the integrity proof of a graph or in supporting an efficient method for pinpointing a specific RDF statement in a graph that contributes to the computation of a different hash value for the entire graph.

In this chapter, we investigate the state-of-the-art methods of generating cryptographic hashes that can be used as integrity proofs for RDF datasets

and define a set of properties that can be used to compare and select the appropriate method of generating integrity proofs. We then propose a more efficient method of computing a cryptographic hash for the special case of growing RDF datasets where the dataset statements carry some notion of ordering (e.g., statements are timestamped). In Section 5.1, we present a comparative analysis of existing methods of generating integrity proofs and define common properties among these methods. In Section 5.2, the algorithm for generating a timestamp based integrity proof is described. We compare our proposed algorithm with nine other integrity proof methods and evaluate how our algorithm is resistant to security threats in Section 5.3. We conclude in Section 5.4.

5.1 Integrity Proof Properties

In this section, we perform a comparative analysis between nine methods of generating a cryptographic hash value for a set of data items that can be used as the integrity proof for the dataset. We investigate these methods in terms of six common properties as described below. For the remainder of the paper, we interchangeably refer to RDF statements (i.e., triples or quads) or RDF graphs as data items and a collection of statements or graphs as a dataset.

Preserving integrity proof independent to data order. This property determines if the method for computing a hash value of the entire dataset is independent from the order of the data items. It is often the case that the same set of data items are ordered in different sequences. For example, querying a database may provide the same set of data to different users, but the order of the data in the query result may be different for each user. In certain scenarios, such as when calculating digital signatures, it is undesirable for the same set of data to generate a unique hash value for each possible data item sequence order. Rather, we would like the hash value to remain *data order independent* so that the same dataset, regardless of the order of its data items, generates the same hash value (assuming that the data items themselves have not been modified).

Support for random access to a hash value. For this property, we are interested to investigate if the hash method supports random access (or provide a notion of indexing) to the hash value of a specific data item. When some data items are modified in a dataset (intentionally or maliciously) and in turn the computed hash value of the entire dataset captured the integrity violation, the ability to efficiently pinpoint hash values of which data items contributed to the inconsistency of the integrity proof is a desirable property. For example, in Linked Data-based auditing, where privacy events are represented by RDF graphs, we need to determine which graph was modified so that auditors can

focus their investigation on the modified statements. Alternatively, a lack of random access forces an auditor to examine the entire dataset to pinpoint altered subgraphs.

Pre-processing and running time. In order to achieve properties such as calculating data order independent hash values, some methods require data pre-processing, such as running the data through a sorting algorithm. The additional pre-processing effort may affect the overall efficiency of the hash calculation method. Since each hash calculation method requires different amounts of pre-processing to achieve desirable properties (such as data order independence), the runtime of each method is an important factor to consider. Importantly, determining the computational cost of achieving different hash calculation properties must be examined.

Proof of membership and non-membership. The last property we investigate is if the hash computation method allows proofs of membership and/or proofs of non-membership. A proof of membership means that there is way to determine if a data item is or is not at a given position in the set, without having to store or retrieve the entire dataset. Alternatively, a proof of non-membership means that we can produce evidence, such as a position or a path, that a given data item is not present in the dataset, without storing or retrieving the entire dataset.

In the following subsections, we evaluate the state-of-the-art methods of computing hash values that can be applied to RDF datasets in terms of the above properties. The results of our analysis is summarized in Table 5.1. Columns in this table are ordered according to the appearance of properties and rows are ordered according to the algorithms and methods described below.

5.1.1 Linked Data Graph Digests

We first discuss two similar methods for computing the digest of Linked Data (RDF) graphs proposed by Melnik [44] and Carroll [12]. A statement in an RDF graph is composed of a subject, predicate, object triple. For each statement in a graph, Melnik [44] computes a hash value of a statement's subject, predicate, and object, and concatenates the three digests to produce the statement's digest. Upon computing a digest for each statement, the set of statement digests are sorted, concatenated together, and hashed to produce the digest of the graph. Similarly, Carroll [12] uses a sort function on the statements, concatenates the sorted statements, and computes the digest. Since both methods use a sort function, the computed hash value is independent to the order of the data items, as the same hash value will be computed for different sequences of the same data. Using concatenation means that there is

Table 5.1: Comparative Analysis of Integrity Proof Methods for RDF Datasets

Algorithm	**Order**	**Indexing**	**Pre-proc.**	**Run. time**	**Mem.**	**Non-mem.**	**Ref.**
Melnik [44]	Y	N	Y	$O(nlog(n))$	N	N	Sect. 5.1.1
Carroll [12]	Y	N	Y	$O(nlog(n))$	N	N	Sect. 5.1.1
Sayers *et al.* [63]	Y	N	N	$O(n)$	N	N	Sect. 5.1.2
Fisteus *et al.* [22]	Y	N	N	$O(n)$	N	N	Sect. 5.1.2
Merkle Tree [46]	N	Y	N	$O(log(n))$	Y	N	Sect. 5.1.3
Sorted Merkle Tree	Y	Y	Y	$O(log(n))$	Y	Y	Sect. 5.1.4
Position Merkle Tree [42]	N	Y	Y	$O(log(n))$	Y	N	Sect. 5.1.5
History Merkle Tree [15]	N	Y	N	$O(log(n))$	Y	N	Sect. 5.1.6
Merklix Tree [17]	Y	Y	Y	$O(log(n))$	Y	Y	Sect. 5.1.7

no indexing ability for these methods since there is no key associated with each data item and the position of each data item in the resulting hash is dependent on the sort function. Both methods require the use of a sort function, such as merge sort, which results in a hash computation runtime of $O(nlog(n))$. Since all of the data are concatenated when computing the hash value, there is no way to provide a proof of membership or non-membership without supplying the whole dataset. These comparisons are summarized in the first two rows of Table 5.1.

5.1.2 Incremental Cryptography

Sayers *et al.* [63] and Fisteus *et al.* [22] utilize incremental cryptography to compute the digest of Linked Data graphs. Incremental cryptography is the process of incrementally applying a commutative operation to a set of data item hash values to produce a hash value for the whole dataset. Sayers and Fisteus use multiplication as the commutative operation [63, 22]. The hash value of a graph is computed by hashing the statements of the graph and multiplying each statement digest together modulo a large prime number [63, 30]. Both methods can compute the same hash value for a set of data independent to the data order because of the commutative operation. However, the consequence of using the commutative operation is that the position of a data

item in the resulting hash value will be lost, so these methods do not support proof of membership and non-membership. Since each of these methods utilize incremental cryptography, they do not need to perform pre-processing to the data (for example, sorting), which is an improvement over the previous two methods, and results in a runtime of $O(n)$. The third and fourth rows of Table 5.1 summarize these comparisons.

5.1.3 Merkle Tree

A Merkle tree [46] is used for the integrity verification of an ordered set of data items. Formally, a Merkle tree is a rooted binary hash tree defined as an undirected graph $MT = (V, E)$, where V is a finite set of vertices (or nodes) and E is a finite set of edges. The vertex set of MT contains three types of elements, *root*, *internal*, and *leaf*. An example Merkle Tree is shown in Figure 5.1. The *root* node has no parents and is defined as $r = h(r.leftChild||r.rightChild)$ and h is a cryptographic hash function (e.g., SHA-256). *Internal* nodes have a parent and are defined as $n_i = h(n_i.leftChild||n_i.rightChild)$. A *leaf* node has a parent and is defined as $l_i = h(data_i)$, where $data_i$ is a data item. Each node in the tree is labeled with a hash value. The label of the root h_r, is used to verify the integrity of the data items that a Merkle tree is generated for. The integrity of a data item can then be verified by reconstructing a portion of the tree up to the root and then comparing the original root hash with the recomputed hash. The root hash of a Merkle tree is dependent on the order of the data items in its leaf nodes. Each sequence of a set of data items results in a unique Merkle tree and in turn a different hash value for the root node. To make the Merkle tree order independent, we need to either sort the data items prior to constructing the tree (Section 5.1.4) or determine an item's position while constructing the tree (Section 5.1.7). The fifth row of Table 5.1 summarizes the Merkle tree properties.

5.1.4 Sorted Merkle Tree

If the set of data contains key-value pairs then a sorted Merkle tree can be constructed. Formally, if we have a dataset $D = \{(k, d)|k \in K, d \in D\}$, where each data item $d \in D$ has an associated key k, then we can construct a Merkle tree where the data items are sorted based on their keys. Contrary to a Merkle tree where the resulting root hash is dependent on the order of the data items, a sorted Merkle tree allows the root hash to be independent of the data order. Since each data item is associated with a key, no matter what the initial order of the data is, a sort function can be applied to the keys to order the data into a specific sequence and then construct a Merkle tree on this ordered sequence of data. However, the sort function requires additional

pre-processing to get the data into an ordered sequence. Similar to a Merkle tree, a sorted Merkle tree allows for proofs of membership without revealing or storing the entire dataset. However, unlike a Merkle tree, a sorted Merkle tree can provide proofs of non-membership by producing a path in the tree where the data item should be. Since the data items are sorted, the correct position of the data item is known. Row six in Table 5.1 summarizes the properties for a sorted Merkle tree.

5.1.5 Position-Aware Merkle Tree

Mao *et al.* [42] present a modified Merkle tree called a Position-aware Merkle tree (PMT) where each node in a Merkle tree can keep track of its relative position to its parent node. A node n_i in a PMT records its position in the tree and is defined as a 3-tuple $(n_i.p, n_i.r, n_i.v)$, where $n_i.p$ is n_i's relative position to its parent node, $n_i.r$ is the number of n_i's leaf nodes, and $n_i.v$ is the value of n_i [42]. A PMT allows the generation of an integrity authentication path for data verification by directly computing the root of the tree without querying the whole tree structure. Although this approach makes each node cognisant of its position in the overall tree structure, it does not utilize the underlying semantics of the data to position the data items in the tree. Similar to a Merkle tree, a PMT does not provide data order independent integrity proofs of datasets. Since each node is aware of its position in the tree, data items can be accessed through the positioning scheme. Proofs of membership can be achieved since a data item can be checked if it is or is not at a given position due to the node position data. Unlike a Merkle tree, a PMT requires some additional processing to record the 3-tuple position index for each node. Row seven of Table 5.1 summarizes the position-aware Merkle tree properties.

5.1.6 History-Based Merkle Tree

Crosby *et al.* propose a tree-based history data structure for tamper-evident logging called a History-Based Merkle tree [15]. The history-based tree is an append-only tree where loggers add log events to the tree incrementally and consistency proofs are generated to prove that each addition to the tree has not altered past additions. However, the addition of data items to the tree does not preserve the semantic order of the data. If log events are added to the tree out of sequence, then the resulting root hash won't be representative of the specific order of the events. Therefore, the root hash of the tree is dependent on the order of the data items. Similar to the Merkle tree, a history-based tree supports indexing and proofs of membership since paths in the tree for each data item can be produced. These comparisons are shown in the eighth row of Table 5.1.

5.1.7 Merklix Tree

A Merklix tree [17] (also known as a Merkle Patricia Tree [20]) is a binary tree with Merkle and radix tree properties. A radix tree is a search tree where keys are strings defined by the position of nodes in the tree and all children of a node share a common prefix of the key. Similar to a Merkle tree, a Merklix tree labels non-leaf nodes with the cryptographic hash of their children and leaf nodes with the cryptographic hash of a data item. Unlike a Merkle tree, a Merklix tree uses a radix tree structure to store elements in the tree by using a key, where each sub-tree shares a common prefix in their key [17]. A Merklix tree is also a rooted binary hash tree defined as an undirected graph $MXT = (V, E)$. An example Merklix Tree is shown in Figure 5.2. The *root* and *leaf* nodes are defined the same way as for the Merkle tree. However, the *internal* nodes of a Merklix tree are defined as the ordered pair $n_i = (hash, key)$, where $hash = h(n_i.leftChild||n_i.rightChild)$ and key is the common prefix shared among the hash values of n_i's children. Assuming that the hash values are in binary, node n_0 is the subtree of nodes whose hash values have a common prefix of 0 (e.g., $h_{l_0} = \mathbf{0}01101001...$ and $h_{l_1} = \mathbf{0}11001110...$). Similarly, node n_1 is the subtree of nodes whose hash values have a common prefix of 1 (e.g., $h_{l_2} = \mathbf{1}0110010...$ and $h_{l_3} = \mathbf{1}1100110...$). Note that the order of the data items in Figure 5.2 differs from the order shown in Figure 5.1, however, the computed hash value of the root for a Merklix tree will always be the same for each possible sequence of the nodes' data items (assuming the data items have not changed) since the data items will be sorted into the correct sequence based on their hash values. Another advantage of Merklix tree is its support for proof of non-membership. Since the position of a data item in the tree is known based on the key, we can determine if a data item was part of the dataset used to construct the tree by producing a path in the tree that would lead to that data item [17].

Although a Merklix tree provides integrity verification proofs of unordered data, additional computation is required to determine the key and an element's position. Furthermore, in certain situations where maintaining the order of the data is important, a Merklix tree does not guarantee order preservation. For example, in security logs, preserving the order of events is critical when conducting forensic analysis and maintaining log provenance. Since a Merklix tree employs a radix tree structure, the data items will be positioned based on their hash values rather than some underlying semantic of the dataset. The last row of Table 5.1 shows these properties.

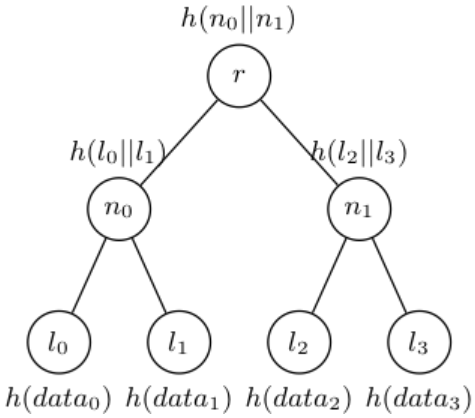


Figure 5.1: Merkle Tree

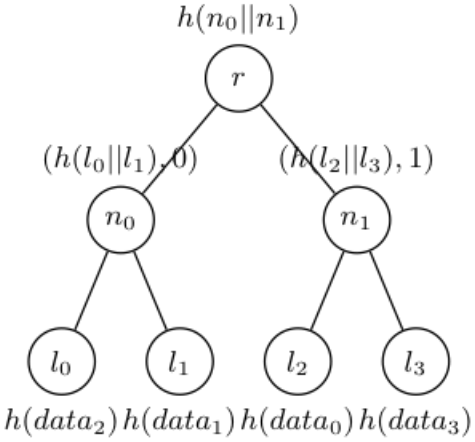


Figure 5.2: Merklix Tree

5.2 Timestamp Tree

In this section we propose an algorithm to construct a sorted Merkle tree for incrementally growing RDF datasets based on a key that is semantically extractable from the RDF dataset. Although not all RDF datasets carry a notion of a key in their dataset, the assumption of finding a key, in special cases, is reasonable since there are a number of semantic databases built on Linked Data principles in which capturing provenance assertions of an individual or collection of RDF statements is necessary. For example, in privacy auditing, a privacy log event is designed as a named graph [80] that the provenance assertions about the event include the necessary timestamp of its publication [61]. This timestamp can be a perfect candidate to be used as the key for the log event (an RDF named graph). We can leverage this existing feature of the dataset to create a sorted Merkle tree without additional pre-processing effort to generate a key or to sort the dataset based on the generated key.

A timestamp tree is an incrementally growing (i.e., append-only) sorted Merkle tree that uses pre-determined timestamp data as the key for data items in the tree. Formally, a timestamp tree is a rooted binary hash tree defined as an undirected graph $TT = (V, E)$. Similar to the trees discussed in Section 5.1, the vertex set of TT contains three types of elements, *root*, *internal*, and *leaf*, where each type is defined as an ordered pair $(hash, timestamp)$. For a *root* or *internal* node n_i, $hash = h(n_i.leftChild||n_i.rightChild)$ and $timestamp = n_i.rightChild.timestamp$. For a *leaf* node, $hash = h(data_i)$ and $timestamp = data_i.timestamp$.

We now illustrate the operations in the construction of TT through the sub-figures of Figure 5.3, where node updates are highlighted. The construction of a timestamp tree includes one `generate` operation followed by $n - 1$ `insert` operations (formally described in Algorithm 4), where n is the number of data

items at a given time.

Data: Previous timestamp tree: tt_{i-1}, RDF data: d_i
Result: Timestamp tree: tt_i
1 $TS_{d_i} \leftarrow$ extractTimestamp(d_i) ;
2 $position_i \leftarrow$ compareTimestamp(TS_{d_i}, tt_{i-1}) ;
3 $newLeaf_i \leftarrow$ insertData($h(d_i)$, TS_{d_i}, tt_{i-1}, $position_i$) ;
4 $tt_i \leftarrow$ recomputeParentHashes($newLeaf_i$, tt_{i-1}) ;
5 **return** tt_i

Algorithm 4: Timestamp Tree Insertion Algorithm

Initially, the `generate`(d_0) operation starts the construction of a timestamp tree as shown in Figure 5.3a, where d_0 is the first data item. The timestamp TS_{d_0} is extracted from d_0 and will be used to determine d_0's position in the tree. Since this is the first stage in the tree generation, there is no position to determine and d_0 is inserted as the root of the tree and is labeled with hash of d_0, $h(d_0)$, and TS_{d_0}. There are no other nodes in the tree so there are no parent node hashes to recompute.

At time T_1, a new data item d_1 is available and is inserted in the tree with the `insert`(tt_0, d_1) operation in Algorithm 4 (Figure 5.3b). Data item d_1's timestamp TS_{d_1} is extracted (line 1) and is compared with each node's timestamp in the tree at time T_0 (tt_0) to determine d_1's position (line 2). Since there is only one node in the tree, TS_{d_1} is compared with TS_{d_0}. Assuming that $TS_{d_1} > TS_{d_0}$, d_1 is inserted to the right of d_0 as a leaf l_1 labeled with the hash of d_0, $h(d_0)$, and TS_{d_0} (line 3). Since a new node was inserted in the tree, the root of the tree must be recomputed. In this case the root from Figure 5.3a becomes leaf l_0 and a new root r_1 labeled with the hash of it's children, $h(l_0||l_1)$, and the rightmost child's timestamp TS_{d_1} is computed (line 4). Data item d_2 is available at time T_2 and is inserted in the tree (Figure 5.3c). The extracted timestamp TS_{d_2} is compared with each node of the tree from T_1 starting at the root r_1. Assuming that $TS_{d_2} > TS_{d_1}$, we know that all children of r_1 contain timestamps before TS_{d_2} so a new root r_2 is created where the previous root r_1 becomes the left child of r_2 and d_2 becomes the right child.

Finally, at time T_3, data item d_3 is inserted in the tree (Figure 5.3d). Again, the comparison with the extracted timestamp TS_{d_3} is performed on r_2 from T_2. Assuming that $TS_{d_3} > TS_{d_2}$, d_3 will be inserted to the right of d_2. Since d_2 is a leaf l_2 at T_2, a new internal node n_1 is created with d_2 as it's left child and d_3 as it's right child. Node n_1 is labeled with the hash of it's children, $h(l_2||l_3)$, and the left child's timestamp TS_{l_3} and is the right child of the new root r_3. The root r_3 updates it's hash value to be $h(n_0||n_1)$ and timestamp to be the right child's timestamp, TS_{n_1}. The process of inserting new data items through timestamp extraction, comparison, insertion, and node re-computation continues as data becomes available.

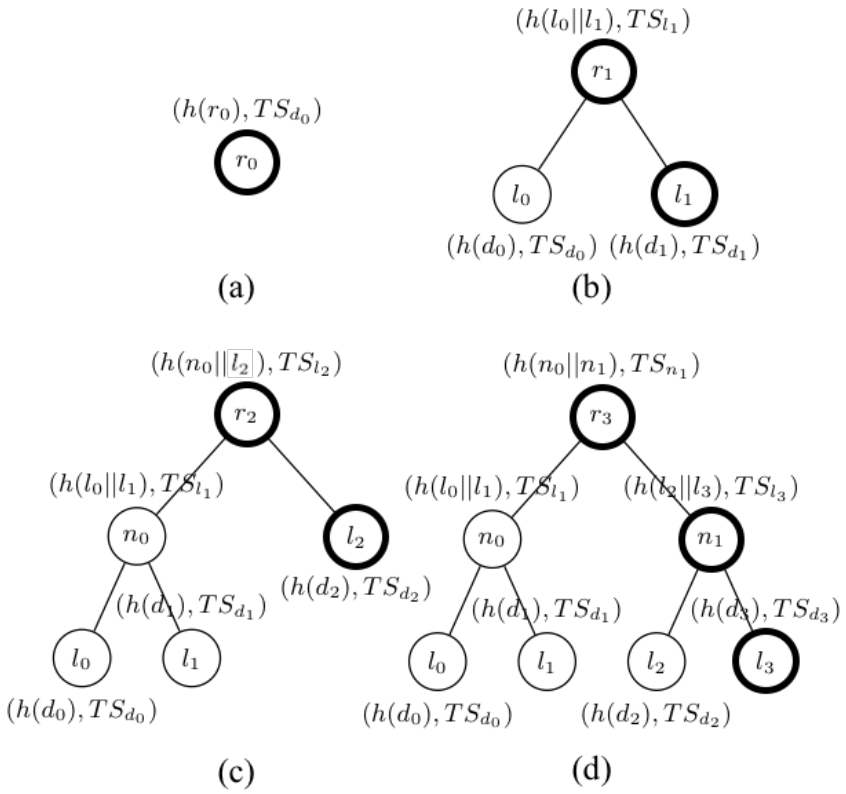


Figure 5.3: Timestamp Tree TT Generation

Throughout the provided tree generation example, it is assumed that $TS_{d_1} < TS_{d_2} < ... < TS_{d_n}$. This assumption is valid in situations where out of sequence data are considered invalid, such as in security and privacy logs. In such datasets, the new data item is always appended to the tree as the rightmost leaf node since the timestamps are sequentially ordered from left to right and the current rightmost leaf of the tree has the latest timestamp. In this case, computation of the hash value of the root node in each increment of a data item is limited to only recomputing the hash values of the rightmost path of the tree. For this reason, we store the right child's timestamp at each parent to provide a timestamp comparison upper bound so that we can immediately insert a new data item as the rightmost leaf node of the tree upon performing a timestamp comparison with the root. However, if in a rare case, insertion of out-of-sequence nodes is a requirement (for example, due to system latency where appending out of sequence data items is necessary), this algorithm supports this type of insertion with the additional cost of comparing timestamps and following a path to a leaf of the tree depending on where in the tree the out-of-sequence data item ends up.

5.3 Evaluation

In this section, we present an evaluation of the timestamp tree in comparison with other methods outlined in Table 5.1. We perform an experimental evaluation of the timestamp tree with the standard Merkle tree and the sorted Merkle tree in terms of generation and query runtimes. Finally, we discuss how the timestamp tree can withstand types of security-related threats.

5.3.1 Comparative Analysis

In terms of the six properties we have presented in Section 5.1, a timestamp tree achieves the ideal set of properties compared to the other methods. A timestamp tree produces an integrity proof independent of the data order since it inserts data items into the tree based on underlying data semantics (i.e., timestamp). Since the data items are ordered in a specific way, a timestamp tree supports random access to a hash value since each node in the tree is indexed with a timestamp key related to the data. Leveraging the semantics of the data to order the items means that there is no pre-processing cost associated with applying a sort function to the data or computing an external key for indexing. The running time for querying and inserting a hash value is only dependent on a subset of the tree, which results in a runtime of $O(log(n))$. This runtime is similar to the asymptotic insertion time for Merkle-based binary hash trees. Proofs of membership and non-membership can also be provided since paths to leaves in the tree can be determined based on the timestamp indexing.

We have performed experiments to determine if the additional steps of the timestamp extraction and comparison in the timestamp tree generation procedure affects the overall generation runtime. Additionally, we measure the execution time of querying the tree for auditing related tasks. We compare the timestamp tree with two similar methods, the Merkle tree and the sorted Merkle tree. The results of the experiment are detailed below.

Dataset

We leveraged the Linked Data Log to Transparency, Accountability, and Privacy (L2TAP) privacy audit log framework [61] to generate synthetic privacy audit log events to compute integrity proofs on. An example log event is shown in Listing 5.1. The events in an L2TAP log are composed of a header representing provenance semantics (lines 4-13), and a body describing privacy semantics (lines 14-23). To construct our dataset, we incrementally generate log events (Listing 5.1) over time so that each event has a different timestamp. The publication timestamp for each event is captured in the event header in

line 11 (this is extracted in the timestamp tree generation). The full listing is shown in Section A.3 of the Appendix.

```
1 @prefix l2tap:<http://purl.org/l2tap#> .
2 @prefix scip:<http://purl.org/scip#> .
3 @prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
4 _:privacy-graph-header {
5   _:logevent a l2tap:PrivacyEvent ;
6     l2tap:eventParticipant _:researcher-2 ;
7     l2tap:publicationTimestamp "2018-01-29T12:00:00Z"^^xsd:dateTime ;
8   ... }
9 _:privacy-graph-body {
10  _:requests-req1 a scip:AccessRequest ;
11    scip:dataRequestor _:researcher-2 ;
12    scip:dataSender _:researcher-1 ;
13    scip:dataSubject _:patient-1234 ;
14    scip:requestedDataItem _:patient-1234-CTScan ;
15    scip:requestedPurpose _:purposes-treatment ;
16  ... }
```

Listing 5.1: Privacy Event Graph

Test Environment

Prior to running the experiment, we generated increasing amounts of the privacy log event in Listing 5.1 with sequential timestamps. The event graph hash computation, tree generation, and hash query operations were run on a MacBook Pro with a 2.9 GHz Intel Core i7 processor and 16 GB of memory. The tree-based integrity proof methods were implemented in Java and the *System.nanoTime()* Java method was used to measure the elapsed execution time of the experiments. The execution time is measuring the time difference between constructing variations of Merkle trees and querying hash values in the trees. The recorded time does not take into account the time to generate the audit log events (these were pre-computed before the experiment). To account for variability in the testing environment, each reported elapsed time is the average of ten independent executions.

Experimental Results

In the context of privacy auditing, it is assumed that all events in a privacy log have an associated publication timestamp. We want to demonstrate how leveraging the underlying semantics of the data can improve the Merkle tree-based integrity proof methods. Specifically, we are interested in evaluating two aspects of the methods: *integrity proof generation* and *data hash query* runtimes. In this experimental evaluation, we compare our timestamp tree approach (Section 5.2) with the standard Merkle tree (Section 5.1.3) and the sorted Merkle tree (Section 5.1.4).

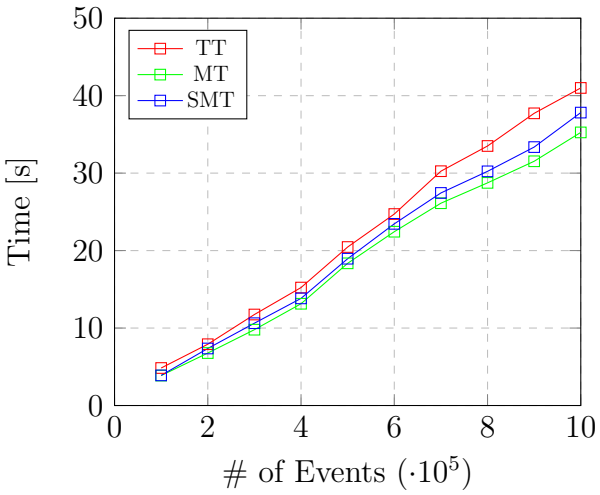


Figure 5.4: Tree Generation Runtime

Integrity proof generation. We opted to perform static tree generation rather than incremental tree generation. Static tree generation constructs the tree at once for n data items, whereas incremental generation constructs a new tree for each data item 1...n. Although we have shown the incremental generation approach in Algorithm 4, we can easily adapt this algorithm for statically generating the tree (i.e., rather than inserting a single data item each iteration, we run the algorithm over the whole dataset at once). In a privacy auditing context, incremental generation is leveraged for when the integrity proof method requires real-time updates, where new log events can be inserted to the tree as they become available. On the other hand, static generation is leveraged for retrospective auditing, where we have a set of generated log events and we want to compute an integrity proof for that dataset. We decided to evaluate the static tree generation since it better highlights the differences in overall runtimes between the three tree-based methods for datasets of increasing sizes.

Figure 5.4 depicts the results of the integrity proof generation runtime experiment comparing the timestamp tree (TT), Merkle tree (MT), and sorted Merkle tree (SMT) for log datasets of increasing sizes (100,000 to 1,000,000 log events). The standard Merkle tree constructs a binary hash tree from the data with no pre-processing steps involved in the construction. A sorted Merkle tree extends this process to include a sorting step prior to constructing the tree. This sorting process requires a key to be generated to order the data. A timestamp tree extracts the key used to sort the data from the data (i.e., timestamp) and additionally carries the timestamp ordering semantics throughout the entire tree. In this experiment, the data are ordered sequentially based on their timestamps prior to constructing the tree (as is typically

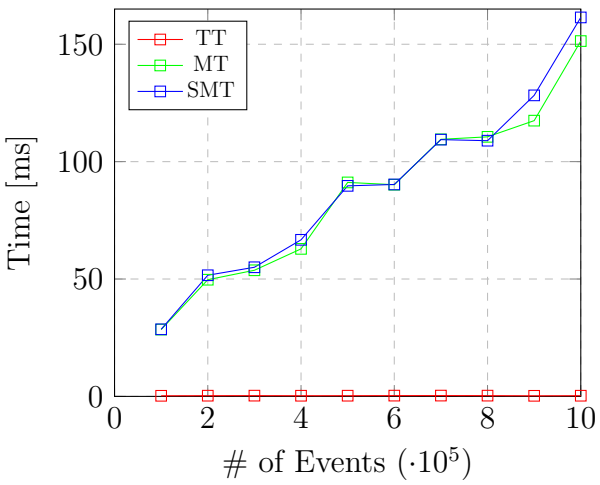


Figure 5.5: Data Hash Query Runtime

the case for audit log events). As shown in Figure 5.4, the Merkle tree has the lowest generation runtime since it simply computes the hash of each data item and constructs the tree. Since the log events are generated in order prior to constructing the tree, the sorting step for the sorted Merkle tree negligibly adds to the processing time. The timestamp tree has the additional step of extracting the timestamp from the event and storing that information in the nodes of the tree, which yields a slight increase in runtime over the other two methods. Based on the results, it can be seen that all three methods have similar generation runtimes and that the timestamp extraction and comparison for the timestamp tree negligibly adds to the runtime.

Data hash query. The ability to query the integrity proof of a privacy audit log for a specific event is beneficial for auditors when conducting an investigation. Specifically, an auditor may ask the question of "did this event produce a hash value that was part of the dataset integrity proof?". We evaluate the three methods on their ability and effectiveness at supporting such a query. After the tree is generated, we compute the hash of a data item and query the tree for that data item. A typical auditing query would want to check if the tree contains the hash of the latest event in the log. To evaluate the query execution runtime for each tree method, we perform a black-box query, where the auditor running the query is not aware of the underlying tree ordering or semantics (i.e., does not know the internal structure of the timestamp tree, sorted Merkle tree, etc.). Querying a timestamp tree is similar to a binary search through the tree (i.e., comparing timestamps at each node), whereas querying the Merkle tree methods resembles a depth first search (i.e., brute-force checking each path in the tree).

Figure 5.5 shows the results of the data hash query runtime experiment.

Although the sorted Merkle tree orders the data items prior to constructing the tree, under the assumption that the auditor is performing a black-box query, they cannot exploit the fact that the latest event is in the rightmost leaf position of the tree. Therefore, the query execution time is similar to the Merkle tree. Furthermore, the sorted Merkle tree does not carry the underlying data semantics throughout the tree (as opposed to the timestamp tree, which carries the timestamps in all tree nodes), so we cannot perform a binary tree search, rather we must rely on the depth first search method. Since the Merkle and sorted Merkle tree methods rely on depth first searching, the query must traverse $2 \cdot d - 1$ nodes, where d is the size of the dataset (i.e., the query must check all leaves to find the data hash). Conversely, the timestamp tree only requires traversing h nodes, where h is the height of the tree (since the query is performing binary search based on the timestamps). Since $h \ll 2 \cdot d - 1$ and h grows slowly, the timestamp tree query runtime is sub-linear compared with the Merkle and sorted Merkle trees. For example, the timestamp tree query for a dataset of size 30,000 takes about 0.32 ms, whereas the Merkle and sorted Merkle trees take around 54 ms. If we increase the dataset to 40,000 the timestamp tree remains at around 0.32 ms, whereas the Merkle trees increase to about 64 ms.

Experiment Conclusion

Although the timestamp tree does not improve over the Merkle tree generation performance, the additional timestamp semantics greatly reduces the query runtime. While the sorted Merkle tree produces identical integrity proofs independent to the order of data, similar to the timestamp tree, we cannot leverage this property when querying, especially in a black-box query scenario, since the tree does not carry the semantics of the data ordering throughout the tree. When comparing the three tree-based integrity proof methods against the generation and query runtime aspects, the timestamp tree is a more suitable structure for audit log based tasks.

5.3.2 Threat Model Evaluation

When the items in a dataset have a temporal aspect, as in the case of auditing, an adversary may change the order of log events to avoid the detection of malicious activity. For example, at time t_0 a log recorded the authorization for a process p_0. At time t_1, an adversary performs an action and is recorded in the log. At time t_2, a benign entity performs another action and is recorded in the log. Suppose a security incident occurs and upon investigation, is determined to have occurred at time t_1 (and unknowingly to the investigators, was ultimately caused by the adversary's actions at time t_1). By attempting to reorder the data items in the timestamp tree to hide malicious activity, the

adversary makes the appearance that the benign entity's actions caused the security incident. A timestamp tree should be able to detect and prevent the malicious attempt to incorrectly insert and change the order of the leaves.

In order to prevent an adversary from reordering the data items in a timestamp tree, we can utilize a chaining mechanism that takes advantage of the underlying data order semantics to detect and prevent this malicious behavior. When a new data item is being inserted into the tree, the data item leaf node's label is modified to additionally contain the hash of the previous root of the tree. Originally, a data item leaf in a timestamp tree is labeled with the pair $(h(d_i), TS_{d_i})$. The modified label contains the triple $(h(d_i), TS_{d_i}, h(r_{i-1}))$, where r_{i-1} is the previous root of the tree. By adding the hash of the previous tree root to the leaf node, an adversary could not successfully insert a data item in the incorrect location. Since the tree is incrementally growing over time, the adversary cannot retrospectively calculate the correct $h(r_{i-1})$ for the previous tree root. Of course for this method to work, all tree root hashes must be published, for example in a blockchain [69, 56], so that they can be used for integrity verifications.

The chaining method previously described can also be applied at the data item generation level. For example, in a privacy auditing scenario, where there are multiple participants contributing to the log, each participant can link together their generated log events. Specifically, if a participant is inserting event data d_i into the tree and they have previously generated data d_{i-1} and d_{i-2}, the leaf node for d_i would be labeled with the pair $(h(d_i||d_{i-1}||d_{i-2}), TS_{d_i})$. Therefore, an adversary could not alter a participant's event data in the tree since they do not possess all of a participant's log events.

5.4 Conclusion

In this chapter, we first defined a set of properties that can be used to compare and select the appropriate method of generating the integrity proof of an RDF graph depending on the requirements that the proof needs to satisfy. Second, we proposed an algorithm to generate an integrity proof specifically in the context of privacy auditing (or other similar logging contexts) where the RDF statements in the log are required to be timestamped (such as in Chapters 3 and 4). Our method is an extension of a sorted Merkle tree where the key is exploited from the semantics of the RDF dataset (e.g., the keys can be retrieved through SPARQL queries) rather than externally generated, as is typically the case for sorted Merkle trees. Although our algorithm is limited for special cases where a key is inherently present in the dataset, it provides the advantage of avoiding the additional cost of preprocessing for sorting or indexing the RDF dataset prior to generating an integrity proof. This is desirable particularly in privacy auditing cases where the RDF dataset is incrementally growing.

Chapter 6

Conclusion

In this chapter, we provide a summary of the thesis contributions and present future research directions for this work.

6.1 Contribution Summary

This thesis aimed to achieve three objectives: (i) establishing *verifiable trust and transparency* in collaborative health research environments; (ii) supporting the *privacy management* of data contributors; and (iii) maintaining the *integrity of all data interactions* in the research environment.

In Chapter 3, we proposed a blockchain-based layered architecture to facilitate the management of trust in collaborative health research environments. Our architecture, which is composed of three modular layers, supports the provenance management of research data, the privacy management of data subjects, and distributed and verifiable trust among all participants. The data layer is responsible for acting as a data repository that stores the actual data records. The transaction layer is where the data sharing transactions take place. Finally, the transparency layer is responsible for providing human-in-the-loop functionality that facilitates transparent data sharing transactions. A discussion on how the architecture addresses privacy and security properties is presented and how the architecture supports the requirements for a trustworthy system. We evaluated the architecture's resiliency to common security threats through a threat model assessment and described the capabilities of adversaries interacting with the system. Finally, we examined the feasibility and scalability of the architecture through a technological realization and experimental evaluation.

Chapter 4 presented a method for utilizing blockchain technology to provide tamper-proof privacy audit logs. The provided solution applies to Linked Data based privacy audit logs, in which lacked a mechanism to preserve log integrity. SPARQL queries and graph generation algorithms are presented

that a log generator can perform to write log events to a blockchain and auditors can perform to verify the integrity of log events. The model can be used by loggers to generate tamper-proof privacy audit logs whereas the integrity queries can be used by external auditors to check if the logs have been modified for nefarious purposes. We include an experimental evaluation that demonstrates the scalability of the audit log integrity verification procedure. Based on our experimental results, the solution scales linearly with increasingly sized privacy audit logs.

We defined a set of properties that can be used to analyze appropriate methods of generating integrity verification proofs for RDF datasets in Chapter 5. Current methods are based on incremental procedures that utilize a commutative operation or use constructions of Merkle trees to produce a hash value for a dataset. We proposed an extension of a sorted Merkle tree, called a timestamp tree, where keys are semantically extractable from an RDF dataset (e.g., timestamps) and exploited to produce an integrity proof that achieves characteristics such as data order independent integrity proofs, random access, and proofs of membership and non-membership. We include an evaluation of the timestamp tree generation algorithm compared to other integrity proof methods and an adversarial threat model evaluation. Based on the results of our comparative analysis, our method remains comparable to existing methods in terms of runtime growth, performs efficiently at data hash queries, and can resist malicious actors that attempt to insert incorrect data to the tree.

6.2 Future Work

There are a number of directions for future work. In this section, we summarize the future work in the three areas of the digitized trust architecture, blockchain-based privacy auditing, and timestamp integrity proofs.

Digitized Trust Architecture. Further research is required to examine how the architecture in Chapter 3 can effectively communicate with end users, different actors, and AI systems across the collaborative research environment. One direction could be application programming interfaces (API) developed for AI systems to interact with the architecture and existing Linked Data endpoints. Additionally, further investigation is required to extend the architectural layers with a presentation layer for all entities across the system to effectively access and utilize the data, as well as participate in the collaborative research pipeline.

Smart contracts are programs that run on the blockchain network that all participants can interact with. Currently, we leverage smart contracts on the blockchain to encode and enforce simple DSA constraints, such as retention periods. By running DSA smart contracts on the blockchain, we

can supplement the current method of ethic review boards (ERB) reviewing and granting approval of data sharing activities, which often takes months to achieve, by having the *system* itself capable of granting approval. Since we are using semantic technologies such as Linked Data to represent the data sharing transactions, the DSA smart contract can *automatically* reason over the data to decide if the data sharing transaction conforms with the constraints outlined in the DSA. For example, a privacy event in a data sharing transaction contains related clauses in a DSA, such as purpose of use. Prior to a data transaction record being written to the blockchain, the DSA smart contract can interact with the transaction layer to confirm that the performed privacy acts in the privacy event comply with the policies in the DSA and depending on the compliance, either reject the transaction or allow the transaction to proceed. Further research is required in encoding and enforcing complex DSA constraints and conditions on the blockchain.

Blockchain-based Privacy Auditing. The tamper-proof privacy auditing model in Chapter 4 has some limitations. First, we acknowledge Bitcoin's limitations in terms of cost, speed, and scalability [41]. We utilized Bitcoin since it provides an established storage mechanism suitable for integrity proofs and to demonstrate the feasibility of our solution applied to Linked Data. For an optimized implementation, other blockchain technologies, such as Ethereum [10] and Hyperledger Fabric [76], should be compared in terms of transaction fee, scalability, and smart contract and private ledger support. Second, a log containing thousands of events will require thousands of transactions and occupy a large space on the blockchain. Using Merkle trees [16, 46] can reduce the storage and transaction requirements by writing the root of the tree (composed of multiple integrity proofs) to the blockchain. However, this will increase the work for an auditor to verify the log integrity since more hash value computations are required to reconstruct the hash tree. Formalizing the trade-offs between hash trees and the verification effort is an interesting optimization problem to investigate.

Timestamp Integrity Proofs. In some application domains (e.g., in privacy audit logs) the nature of the chosen key used in the timestamp tree proposed in Chapter 5 may be considered private information. Further work can be done to apply privacy-preserving techniques such as fully homomorphic encryption or zero knowledge proofs, where the exact timestamp is not revealed but the data insertion comparisons can still be performed over the encrypted data. Furthermore, identical timestamps are stored in multiple tree nodes, resulting in redundant data in the tree. Node data optimization needs to be investigated to determine indexing schemes that allow the random access of data where a subset of the nodes carry a key rather than all nodes.

Appendix A

Full Graph Listings

A.1 Signature Graph

```
@prefix sig: <http://icp.it-risk.iwvi.uni-koblenz.de/ontologies/signature.owl#> .
@prefix l2tapp: <http://purl.org/l2tapp#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
_:sig-graph {
  _:gsm a sig:GraphSigningMethod ;
     sig:hasDigestMethod sig:dm-sha-256 ;
     sig:hasGraphDigestMethod sig:gdm-sayers-2004 ;
     sig:hasSignatureMethod sig:sm-ecdsa .
  _:sig-graph a sig:Signature ;
      sig:hasVerificationCertificate <signer/WebID/URI> ;
      sig:hasSignatureValue "MEUCIQC44Qy2O8Mx..."^^xsd:string ;
      l2tapp:hasSignedGraph _:log_header . }
```

Listing A.1: Signature Graph

A.2 Block Graph

```
@prefix block: <http://www.semanticblockchain.com/Blondie.owl#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
_:block-graph {
 _:block-graph a block:BitcoinBlock ;
  block:hasBitcoinBlockheaderBlock block:BitcoinBlockheader ;
  block:hasBitcoinPayloadBlock block:BitcoinPayload .
 block:BitcoinBlockheader block:hashPreviousBlockheader "000000...66b485"^^xsd:string ;
  block:nonceBlockheader 2614034692^^xsd:decimal ;
  block:timestampBlockheader 1337500093^^xsd:decimal .
 block:BitcoinPayload block:hasTransactionPayload block:BitcoinTransaction .
 block:BitcoinTransaction block:hasBitcoinTransactionInput block:BitcoinTransactionInput ;
  block:hasBitcoinTransactionOutput block:BitcoinTransactionOutput .
 block:BitcoinTransactionInput block:hashBitcoinTransactionInput "1a2...3fc"^^xsd:string ;
  block:scriptSignBitcoinTransactionInput "4730440...41d6e6"^^xsd:string .
 block:BitcoinTransactionOutput block:scriptPubkeyBitcoinTransactionOutput "6a28486F...
      Ae46e65"^^xsd:string . }
```

Listing A.2: Block Graph

A.3 Privacy Event Graph

```
@prefix l2tap:<http://purl.org/l2tap#> .
@prefix scip:<http://purl.org/scip#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
_:privacy-graph-header {
  _:logevent a l2tap:PrivacyEvent ;
    l2tap:memberOf _:log1 ;
    l2tap:eventParticipant _:researcher-2 ;
    l2tap:receivingTimestamp _e1-t1 ;
    l2tap:eventData _:privacy-graph-body .
  _:e1-t1 a tl:Instant ;
    tl:atDateTime "2018-01-29T12:00:00Z"^^xsd:dateTime ;
    tl:onTimeline _:tlphysical .
  _:e1-t2 a tl:Instant ;
    tl:atDateTime "2018-01-29T12:00:00Z"^^xsd:dateTime ;
    tl:onTimeline _:tlphysical .
  _:privacy-graph-body a rdfg:Graph . }
_:privacy-graph-body {
  _:requests-req1 a scip:AccessRequest ;
    scip:dataRequestor _:researcher-2 ;
    scip:dataSender _:researcher-1 ;
    scip:dataSubject _:patient-1234 ;
    scip:requestedDataItem _:patient-1234-CTScan ;
    scip:requestedPurpose _:purposes-treatment ;
    scip:requestedPrivilege _:Use .
  _:researcher-2 scip:requestorRole _:Principle-Investigator .
  _:researcher-1 scip:senderRole _:Researcher . }
```

Listing A.3: Privacy Event Graph

Bibliography

[1] Accorsi, R. (2009). Log data as digital evidence: What secure logging protocols have to offer? In *Computer Software and Applications Conference, 2009. COMPSAC'09. 33rd Annual IEEE International*, volume 2, pages 398–403. IEEE.

[2] Agrawal, R., Evfimievski, A., Kiernan, J., and Velu, R. (2007). Auditing disclosure by relevance ranking. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 79–90. ACM.

[3] Alexander, P., Kimmell, G., and Burke, D. (2007). Security as a system property: Modeling trust and security in rosetta.

[4] Anderson, N. (2016). Blockchain technology: A game-changer in accounting? *Deloitte*.

[5] Antonopoulos, A. M. (2014). *Mastering Bitcoin: unlocking digital cryptocurrencies*. O'Reilly Media, Inc.

[6] Azaria, A., Ekblaw, A., Vieira, T., and Lippman, A. (2016). Medrec: Using blockchain for medical data access and permission management. In *2nd International Conference on Open and Big Data (OBD'16)*, pages 25–30.

[7] Bizer, C. and Cyganiak, R. (2013). Trig: Rdf dataset language. W3C. `http://www.w3.org/TR/trig/`. Accessed May 2017.

[8] Bizer, C., Heath, T., and Berners-Lee, T. (2009). Linked data-the story so far. *International journal on semantic web and information systems*, **5**(3), 1–22.

[9] Bradbury, D. (2013). The problem with bitcoin. *Computer Fraud & Security*, **2013**(11), 5–8.

[10] Buterin, V. (2013). Ethereum white paper. GitHub Repository. `https://github.com/ethereum/wiki/wiki/White-Paper`. Accessed July 2017.

[11] Butin, D., Chicote, M., and Le Métayer, D. (2013). Log design for accountability. In *2013 IEEE Symposium on Security and Privacy Workshops*, pages 1–7.

[12] Carroll, J. J. (2003). Signing rdf graphs. In *International Semantic Web Conference*, pages 369–384. Springer.

[13] Carroll, J. J., Bizer, C., Hayes, P., and Stickler, P. (2005). Named graphs, provenance and trust. In *Proceedings of the 14th international conference on World Wide Web*, pages 613–622. ACM.

[14] Castelluccia, C., Druschel, P., Hübner, S., Pasic, A., Preneel, B., and Tschofenig, H. (2011). Privacy, accountability and trust-challenges and opportunities. ENISA. `http://www.enisa.europa.eu/activities/identity-and-trust/library/deliverables/pat-study/atdownload/fullReport`. Accessed May 2017.

[15] Crosby, S. A. and Wallach, D. S. (2009). Efficient data structures for tamper-evident logging. In *USENIX Security Symposium*, pages 317–334.

[16] Cucurull, J. and Puiggalí, J. (2016). Distributed immutabilization of secure logs. In *International Workshop on Security and Trust Management*, pages 122–137. Springer.

[17] Deadalnix's Den (2016). Introducing merklix tree as an unordered merkle tree on steroid. `https://www.deadalnix.me/2016/09/24/introducing-merklix-tree-as-an-unordered-merkle-tree-on-steroid/`. Accessed Jan. 2018.

[18] Eijdenberg, A., Laurie, B., and Cutter, A. (2015). Verifiable data structures. GitHub Repository. `https://github.com/google/trillian/blob/master/docs/VerifiableDataStructures.pdf`. Accessed March 2018.

[19] English, M., Auer, S., and Domingue, J. (2016). Block chain technologies & the semantic web: A framework for symbiotic development. In *Computer Science Conference for University of Bonn Students, J. Lehmann, H. Thakkar, L. Halilaj, and R. Asmat, Eds*, pages 47–61.

[20] Ethereum Wiki (2017). Merkle patricia trie specification. GitHub Repository. `https://github.com/ethereum/wiki/wiki/Patricia-Tree`. Accessed Jan. 2018.

[21] Field, M. J. and Lo, B. (2009). Conflict of interest in medical research, education, and practice. National Academies Press (US). `https://www.ncbi.nlm.nih.gov/books/NBK22942/`. Accessed March 2017.

[22] Fisteus, J. A., García, N. F., Fernández, L. S., and Kloos, C. D. (2010). Hashing and canonicalizing notation 3 graphs. *Journal of Computer and System Sciences*, **76**(7), 663–685.

[23] Freire, J., Koop, D., Santos, E., and Silva, C. T. (2008). Provenance for computational tasks: A survey. *Computing in Science & Engineering*, **10**(3).

[24] Google (2018). Trillian. GitHub Repository. `https://github.com/google/trillian`. Accessed Feb. 2018.

[25] Google DeepMind Health (2017). Trust, confidence and verifiable data audit. `https://deepmind.com/blog/trust-confidence-verifiable-data-audit/`. Accessed Jan. 2018.

[26] Haber, S. and Stornetta, W. S. (1990). How to time-stamp a digital document. In *Conference on the Theory and Application of Cryptography*, pages 437–455. Springer.

[27] Heath, T. and Bizer, C. (2011). Linked data: Evolving the web into a global data space. *Synthesis lectures on the semantic web: theory and technology*, **1**(1), 1–136.

[28] HL7 International (2017). Fhir. `https://www.hl7.org/fhir/index.html`. Accessed June 2017.

[29] HL7 International (2017). FHIR Security. `https://www.hl7.org/fhir/security.html`. Accessed April 2018.

[30] Kasten, A. (2016a). *Secure semantic web data management: confidentiality, integrity, and compliant availability in open and distributed networks.* Ph.D. thesis, University of Koblenz and Landau, Germany.

[31] Kasten, A. (2016b). A software framework for iterative signing of graph data. GitHub Repository. `https://github.com/akasten/signingframework`. Accessed Jan. 2017.

[32] Kasten, A., Scherp, A., and Schauß, P. (2014). A framework for iterative signing of graph data on the web. In *European Semantic Web Conference*, pages 146–160. Springer.

[33] Kehoe, L., Dalton, D., Leonwicz, C., and Jankovich, T. (2015). Blockchain disrupting the financial services industry? *Deloitte*.

[34] Kleedorfer, F., Panchenko, Y., Busch, C. M., and Huemer, C. (2016). Verifiability and traceability in a linked data based messaging system. In *Proceedings of the 12th International Conference on Semantic Systems*, pages 97–100. ACM.

[35] Kushida, C. A., Nichols, D. A., Jadrnicek, R., Miller, R., Walsh, J. K., and Griffin, K. (2012). Strategies for de-identification and anonymization of electronic health record data for use in multicenter research studies. *Medical care*, pages S82–S101.

[36] Li, Y. (2010). The case analysis of the scandal of enron. *International Journal of business and management*, **5**(10), 37.

[37] Liang, X., Zhao, J., Shetty, S., Liu, J., and Li, D. (2017). Integrating blockchain for data sharing and collaboration in mobile healthcare applications. In *Personal, Indoor, and Mobile Radio Communications (PIMRC), 2017 IEEE 28th Annual International Symposium on*, pages 1–5. IEEE.

[38] Libert, B., Beck, M., and Wind, J. (2016). How blockchain technology will disrupt financial services firms. *Knowledge@ Wharton*, pages 2–7.

[39] Lindqvist, A. (2017). *Privacy Preserving Audit Proofs*. Master's thesis, KTH Royal Institute of Technology, Sweden.

[40] Linn, L. A. and Koo, M. B. (2017). Blockchain for health data and its potential use in health it and health care related research. `https://www.healthit.gov/sites/default/files/11-74-ablockchainforhealthcare.pdf`. Accessed Sept. 2017.

[41] Manu, S. (2017). Building better blockchains. In *Linked Data in Distributed Ledgers Workshop Keynote*. WWW2017.

[42] Mao, J., Zhang, Y., Li, P., Li, T., Wu, Q., and Liu, J. (2017). A position-aware merkle tree for dynamic cloud data integrity verification. *Soft Computing*, **21**(8), 2151–2164.

[43] McFarlane, C., Beer, M., Brown, J., and Prendergast, N. (2017). Patientory: A healthcare peer-to-peer emr storage network v1. 0.

[44] Melnik, S. (2001). Rdf api draft: Cryptographic digests of rdf models and statements.

[45] Mercuri, M., Rehani, M. M., and Einstein, A. J. (2012). Tracking patient radiation exposure: Challenges to integrating nuclear medicine with other modalities. *Journal of Nuclear Cardiology*, **19**(5), 895–900.

[46] Merkle, R. C. (1980). Protocols for public key cryptosystems. In *Security and Privacy, 1980 IEEE Symposium on*, pages 122–122.

[47] Mettler, M. (2016). Blockchain technology in healthcare: The revolution starts here. In *IEEE 18th International Conference on e-Health Networking, Applications and Services (Healthcom'16)*, pages 1–3.

[48] Microsoft (2005). The stride threat model. `https://msdn.microsoft.com/en-us/library/ee823878(v=cs.20).aspx`. Accessed Jan. 2018.

[49] MIT Kerberos (2009). Kerberos: The network authentication protocol. `https://web.mit.edu/kerberos/`. Accessed April 2018.

[50] Nakamoto, S. (2008). Bitcoin: A peer-to-peer electronic cash system.

[51] Nelson, G. S. (2015). Practical implications of sharing data: a primer on data privacy, anonymization, and de-identification. In *SAS Global Forum Proceedings*.

[52] OpenLink Software (2017). Virtuoso universal server. `https://virtuoso.openlinksw.com/`. Accessed Jan. 2017.

[53] Ostrovsky, R., Rackoff, C., and Smith, A. (2004). Efficient consistency proofs for generalized queries on a committed database. In *International Colloquium on Automata, Languages, and Programming*, pages 1041–1053. Springer.

[54] Patil, H. K. and Seshadri, R. (2014). Big data security and privacy issues in healthcare. In *Big Data (BigData Congress), 2014 IEEE International Congress on*, pages 762–765. IEEE.

[55] Peterson, K., Deeduvanu, R., Kanjamala, P., and Boles, K. (2016). A blockchain-based approach to health information exchange networks.

[56] Pilkington, M. (2015). Blockchain technology: principles and applications. *Research Handbook on Digital Transformations*.

[57] Rehani, M. M. (2013). Challenges in radiation protection of patients for the 21st century. *American Journal of Roentgenology*, **200**(4), 762–764.

[58] Rodriguez, A. (2008). Restful web services: The basics. *IBM developerWorks*, page 33.

[59] Samavi, R. and Consens, M. P. (2012). L2tap+scip: An audit-based privacy framework leveraging linked data. In *8th International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom'12)*, pages 719–726.

[60] Samavi, R. and Consens, M. P. (2014). Publishing l2tap logs to facilitate transparency and accountability. In *The* 23^{rd} *International World Wide Web Conference (WWW'14) Workshop on Linked Data on the Web*.

[61] Samavi, R. and Consens, M. P. (2018). Publishing privacy logs to facilitate transparency and accountability. In *The Journal of Web Semantics*.

[62] Sambra, A., Story, H., and Berners-Lee, T. (2014). Webid 1.0: Web identity and discovery. W3C. `https://www.w3.org/2005/Incubator/webid/spec/identity/`.

[63] Sayers, C. and Karp, A. H. (2004). Computing the digest of an rdf graph. *Mobile and Media Systems Laboratory, HP Laboratories, Palo Alto, USA, Tech. Rep. HPL-2003-235*, **1**.

[64] Schneier, B. (2007). *Applied cryptography: protocols, algorithms, and source code in C*. john wiley & sons.

[65] Sotos, J. and Houlding, D. (2017). Blockchains for data sharing in clinical research: Trust in a trustless world.

[66] Spoke, M. (2015). How blockchain tech will change auditing for good.

[67] Stathopoulos, V., Kotzanikolaou, P., and Magkos, E. (2008). Secure log management for privacy assurance in electronic communications. *computers & security*, **27**(7-8), 298–308.

[68] Suleyman, M. and Laurie, B. (2017). Trust, confidence and verifiable data audit.

[69] Sutton, A. and Samavi, R. (2017). Blockchain enabled privacy audit logs. In *Proceedings of the 16th International Semantic Web Conference*, pages 645–660. Springer.

[70] Sutton, A. and Samavi, R. (2018a). Tamper-proof privacy auditing for artificial intelligence systems. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, pages 1–4.

[71] Sutton, A. and Samavi, R. (2018b). Timestamp-based integrity proofs for linked data. In *Proceedings of Semantic Big Data Workshop (SBD'18)*, pages 1–6. ACM.

[72] Sutton, A., Samavi, R., Doyle, T. E., and Koff, D. (2018). Digitized trust in human-in-the-loop health research. In *Privacy, Security, and Trust (PST'18)*, pages 1–10.

[73] Symantec (2018). `https://www.symantec.com/`. Accessed April 2018.

[74] The Economist (2018). Apple and amazon's moves in health signal a coming transformation. `https://www.economist.com/news/business/21736193-worlds-biggest-tech-firms-see-opportunity-health-care-which-could-mean-empowered`. Accessed March 2018.

[75] The Linux Foundation (2018). Hyperledger explorer. `https://www.hyperledger.org/projects/explorer`. Accessed Feb. 2018.

[76] The Linux Foundation Projects (2017). Hyperledger fabric. `https://www.hyperledger.org/projects/fabric`. Accessed June 2017.

[77] Tong, Y., Sun, J., Chow, S. S., and Li, P. (2014). Cloud-assisted mobile-access of health data with privacy and auditability. *IEEE Journal of biomedical and health Informatics*, **18**(2), 419–429.

[78] Ugarte, H. E. R. (2016). Blondie - blockchain ontology with dynamic extensibility. GitHub Repository. `https://github.com/hedugaro/Blondie`.

[79] Veeramachaneni, K., Arnaldo, I., Cuesta-Infante, A., Korrapati, V., sBassias, C., and Li, K. (2016). Ai^2: Training a big data machine to defend. In *IEEE International Conference on Intelligent Data and Security (IDS'16)*, pages 49–54.

[80] W3C (2004). RDFG: Named Graph Vocabulary. `http://www.w3.org/2004/03/trix/rdfg-1/`. Accessed June 2015.

[81] Weitzner, D. J., Abelson, H., Berners-Lee, T., Feigenbaum, J., Hendler, J., and Sussman, G. J. (2008). Information accountability. *Communications of the ACM*, **51**(6), 82–87.

[82] Wilson, M., Crompton, S., Matthews, B., and Orlov, A. (2011). Enforcing scientific data sharing agreements. In *E-Science (e-Science), 2011 IEEE 7th International Conference on*, pages 271–278. IEEE.

[83] Xu, L., Chen, L., Shah, N., Gao, Z., Lu, Y., and Shi, W. (2017). Dl-bac: Distributed ledger based access control for web applications. In *Proceedings of the 26th International Conference on World Wide Web Companion*, pages 1445–1450. International World Wide Web Conferences Steering Committee.

[84] Yue, X., Wang, H., Jin, D., Li, M., and Jiang, W. (2016). Healthcare data gateways: Found healthcare intelligence on blockchain with novel privacy risk control. *Journal of Medical Systems*, **40**(10), 218.

[85] Zafar, F., Khan, A., Suhail, S., Ahmed, I., Hameed, K., Khan, H. M., Jabeen, F., and Anjum, A. (2017). Trustworthy data: A survey, taxonomy and future trends of secure provenance schemes. *Journal of Network and Computer Applications*, **94**, 50–68.

[86] Zyskind, G., Nathan, O., and Pentland, A. (2015). Decentralizing privacy: Using blockchain to protect personal data. In *2015 IEEE Security and Privacy Workshops*, pages 180–184.