Blockchain Enabled Privacy Audit Logs

Andrew Sutton(✉) and Reza Samavi

Department of Computing and Software, McMaster University,
1280 Main St. West, Hamilton, ON L8S 4K1, Canada
{suttonad,samavir}@mcmaster.ca

Abstract. Privacy audit logs are used to capture the actions of participants in a data sharing environment in order for auditors to check compliance with privacy policies. However, collusion may occur between the auditors and participants to obfuscate actions that should be recorded in the audit logs. In this paper, we propose a Linked Data based method of utilizing blockchain technology to create tamper-proof audit logs that provide proof of log manipulation and non-repudiation. We also provide experimental validation of the scalability of our solution using an existing Linked Data privacy audit log model.

Keywords: Blockchain · Privacy audit log · RDF signatures · Bitcoin · Tamper-proof · Linked Data · Semantic Web · DSA · Privacy

1 Introduction

Protecting the privacy of individuals who contribute their data to a collaborative service or research environment is becoming more challenging. This becomes apparent as an individual's personal information is passed between different organizations that might operate under different jurisdictions and governing bodies. Data sharing agreements (DSA) are legally binding documents established between organizations that detail the policies and conditions related to the sharing of personal data [1]. The scenario in Fig. 1 demonstrates a collaborative research environment where the research teams must comply with the DSAs and are monitored for their compliance through the use of privacy audit logs. Auditors are responsible for checking compliance with the DSA by examining the privacy logs generated by the research teams [2].

In the scenario in Fig. 1, there is a problem in the trust placed in an auditor and the audit log itself. If the auditor works for the organization that they are auditing then the quality of the audit depends on influencing factors between the organization and the auditor [3]. Collusion can occur between individuals in the organization, such as researchers in the research teams, and the auditor 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.

C. d'Amato et al. (Eds.): ISWC 2017, Part I, LNCS 10587, pp. 645–660, 2017.
DOI: 10.1007/978-3-319-68288-4_38

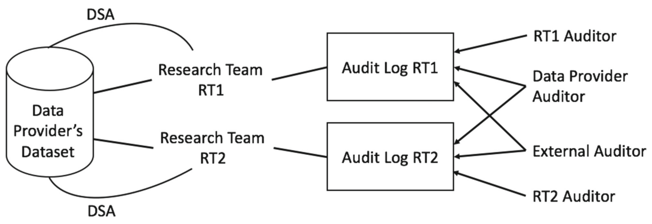


Fig. 1. Example privacy auditing scenario in a data sharing environment [5]

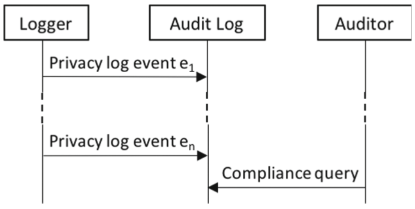
In order to combat the potential modification of the logs due to collusion, a mechanism to provide tamper-proof audit logs is needed [3,4].

In this paper, we propose a Linked Data-based [6] 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. The Linked Data-based L2TAP (Linked Data Log to Transparency, Accountability, and Privacy) audit log framework [5] is used as the underlying log framework. We leverage theories and technologies stemming from blockchain technology [3,4,7–9], Linked Data graph signatures [10–13], and Linked Data graph digest computation [12,14] to create non-repudiable log events and utilize the distributed and immutable properties of blockchain technology to make the audit logs tamper-proof. We experimentally verify that the log integrity verification process scales linearly.

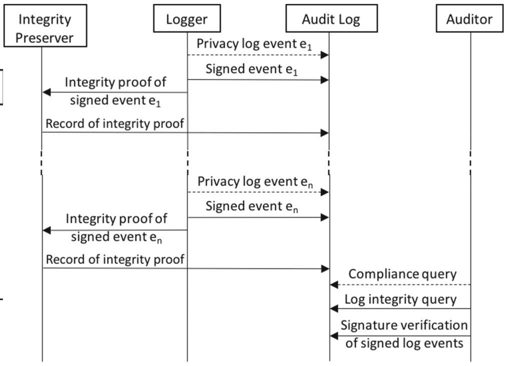
The structure of the paper and the contributions of our work are as follows. Section 2 presents how privacy audit logs are generated and the design requirements of our model. Our solution to generate tamper-proof privacy audit logs is described in Sect. 3. Section 4 presents a SPARQL-based solution to perform log integrity verification. In Sect. 5, the results of an experiment to validate the scalability of our method is given. Section 6 provides an investigation of the related work. Concluding remarks are discussed in Sect. 7.

2 Characteristics of Tamper-Proof Privacy Logs

Privacy auditing addresses three characteristics of information accountability: validation, attribution, and evidence [15,16]. 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 [15,16]. To address these 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 Fig. 2a. 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



(a) Privacy audit log generation process


Integrity Preserver
Logger
Audit Log
Auditor
Privacy log event e_1
Signed event e_1
Integrity proof of signed event e_1
Record of integrity proof
Privacy log event e_n
Signed event e_n
Integrity proof of signed event e_n
Record of integrity proof
Compliance query
Log integrity query
Signature verification of signed log events

(b) Tamper-proof log generation process

Fig. 2. Privacy audit log generation comparison

can then perform compliance queries against the audit log to determine if the performed acts are in compliance with the polices in the governing DSA (e.g., the scenario in Fig. 1) [5].

There are a number of proposals on logs for supporting privacy auditing [18–20]. 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 [5]. 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 [5]. In this research we extend the L2TAP ontology to support non-repudiation and log event integrity.

2.1 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 Fig. 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 [3,4]. 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 2b 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 [5], the L2TAP privacy audit log is scalable. The additional verification processes introduced in this paper to make the log tamper-proof should preserve the scalability.

3 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 [21,22]. 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 Sect. 3.1. We describe the architecture of our model in Sect. 3.2. Sections 3.3 and 3.4 present the signature graph and block graph generation components of our model, respectively.

3.1 Bitcoin Blockchain

The Bitcoin system [23] is a cryptocurrency 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. A blockchain is a decentralized database comprised of a continuously increasing amount of records, or blocks, that represents an immutable digital ledger of transactions [7].

Distributed ledgers allow for a shared method of record keeping where each participant has a copy of the ledger, meaning that each node on the network will have to be in collusion to modify the records in the blockchain. Each block in the blockchain is composed of a header containing a hash of the previous block in the chain (forming a chain of blocks) and a payload of transactions.

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 [8]. 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 [24]. 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 [24]. We utilize this queryable special transaction to store an integrity proof of privacy audit logs on the Bitcoin blockchain.

3.2 Architectural Components

A blockchain is well suited to fill the role of the integrity preserver in the tamper-proof log generation process in Fig. 2b. We use the capabilities 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 Fig. 3 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

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

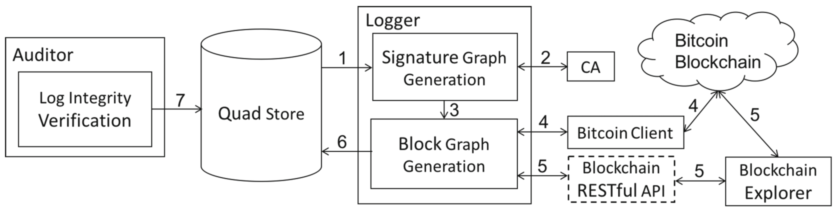


Fig. 3. The architecture of the model

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 [25]. 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 [26] 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 [13] expresses all of the necessary algorithms and methods required for verifying a signature. The BLONDiE [27,28] 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].

3.3 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 [11], which includes: canonicalization, serialization, hash, signature, and assembly [10,11]. 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 [29].

```
**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 [14] 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 [14]. 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 [14]. 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 [14]). 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

[2] https://www.symantec.com.

[3] http://l2tap.org.

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 [11].

Listing 1.1 illustrates an example signature graph generated using Algorithm 1. Analogous to work presented in [10], we also use the Signing Framework signature ontology [13]. Lines 5–8 in this listing contain the signature triples. Line 6 contains the WebID [30] where the signer's public key can be acquired [10]. 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 (omitted here).

```
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 1.1. Signature graph

3.4 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

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

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

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 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 1.2 illustrates an example of a block graph output by Algorithm 2, serialized in TriG. We use the BLONDiE [27] 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 to save space.

```
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 1.2. Block graph

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

4 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 [5] 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 [5] 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 Sect. 5). 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 [2], 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: $v_i$
1 $Triples \leftarrow (\text{extractTriples}(hg_i) \cup \text{extractTriples}(bg_i) \cup \text{extractTriples}(sg_i))$ ;
2 $H \leftarrow \prod_{j=1}^{|Triples|} h(t_j \in Triples) mod(p)$ ;
3 $URI \leftarrow$ Query block graphs for $H$ (Listing 1.3) ;
4 if $URI \neq \emptyset$ then
5     $v_i \leftarrow \text{verifySignature}(sg_i, hg_i, bg_i)$ ;
6 else
7     $v_i \leftarrow$ false ;
8 end
9 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 [3]. 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 Sect. 3.4. The SPARQL query in Listing 1.3 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 1.3. 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.

5 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 5.1 describes the synthetic audit log used in the experiment. In Sect. 5.2, we illustrate the details of the test environment. The results of the experiment are discussed in Sect. 5.3.

5.1 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

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

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 [5]. To create a larger audit log, we repeatedly generate the access request events. Our largest synthetic log, composed of 9998 events, contains a total of 989,589 triples: 84 triples from the first three events of log initialization, participants registration, and privacy preferences and policies, and 989,505 triples generated from 9995 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 9998 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 39,992 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 illustrates the log sizes used for the experiment, which range from a log containing 98 events to 9998 events.

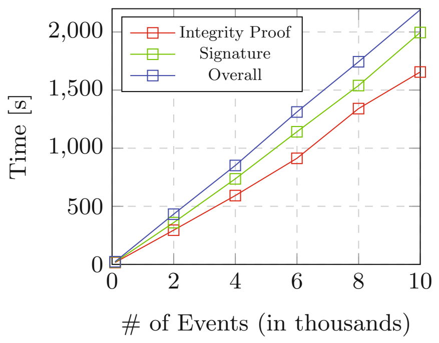
5.2 Test Environment

The experiment was run by executing the SPARQL queries on a Virtuoso [31] 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 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.

5.3 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 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

Fig. 4. Elapsed execution times for integrity and signature verification

of graphs were input to Algorithm 3, which computes the integrity proof digest and executes the query in Listing 1.3 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 98 to 9998, as shown in Fig. 4. Figure 4 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 9998 events requires 29,994 (9998 header, 9998 body, 9998 signature graphs) graph digest calculations. A log of this size will generate 9998 block graphs as well, which will need to be searched for the integrity proof.

The elapsed execution time is plotted in Fig. 4. 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 min 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.

6 Related Work

There are a number of proposals that provide a mechanism for verifying the integrity of an audit log [17,32]. Butin et al. [19] 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. [20] propose a method of providing role-based access control auditability in audit logs to prevent the misuse of data. These solutions only address the integrity of privacy audit logs and miss the non-repudiation aspect. There is a need for a practical solution for supporting the non-repudiation *and* integrity of the logs.

Kleedorfer et al. [10] propose a Linked Data based messaging system that verifies conversations using digital signatures. The RDF graph messages are signed and a signature graph is produced, which can be iteratively signed as the messages pass between recipients. Kasten et al. [11] provide a framework for computing RDF graph signatures. This framework supports signing graph data at different levels of granularity, multiple graph signatures, and iterative graph signatures [11]. Kasten [12] discusses how the confidentiality, integrity, and availability of Semantic Web data can be achieved through approaches of Semantic Web encryption and signatures.

Use of blockchain technology in the auditing of financial transactions have been investigated [3] after the repercussions of the Enron Scandal in 2001, where auditor fraud was the source of public distrust [4]. Anderson [3] proposes a method of verifying the integrity of files using a blockchain. Similar to our approach, Cucurull et al. [8] present a method for enhancing the security of logs by utilizing the Bitcoin blockchain. Our approach differs by providing a model to create tamper-proof logs in a highly scalable Linked Data environment.

7 Conclusions

In this paper we 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. The paper includes 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.

There are a number of directions for future work. First, we acknowledge Bitcoin's limitations in terms of cost, speed, and scalability [33]. 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 [34], 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 [8,35] 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.

Acknowledgments. Financial support from NSERC and SOSCIP are greatly acknowledged.

References

1. Swarup, V., Seligman, L., Rosenthal, A.: A data sharing agreement framework. In: Bagchi, A., Atluri, V. (eds.) ICISS 2006. LNCS, vol. 4332, pp. 22–36. Springer, Heidelberg (2006). doi:10.1007/11961635_2
2. Samavi, R., Consens, M.P.: L2TAP+SCIP: an audit-based privacy framework leveraging Linked Data. In: 8th International Conference on Collaborative Computing: Networking, Applications and Worksharing, CollaborateCom 2012, pp. 719–726. IEEE (2012)
3. Anderson, N.: Blockchain Technology: A Game-Changer in Accounting? Deloitte (2016)
4. Spoke, M.: How blockchain tech will change auditing for good. CoinDesk (2015). http://www.coindesk.com/blockchains-and-the-future-of-audit/. Accessed Feb 2017
5. Samavi, R., Consens, M.P.: Publishing L2TAP logs to facilitate transparency and accountability. In: Proceedings of the Workshop on Linked Data on the Web Co-located with the 23rd International World Wide Web Conference (WWW), Seoul, Korea, vol. 1184. CEUR-WS (2014)
6. Heath, T., Bizer, C.: Linked Data: Evolving the Web into a Global Data Space. Synthesis Lectures on the Semantic Web: Theory and Technology, 1st edn., vol. 1, pp. 1–136. Morgan & Claypool, San Rafael (2011)
7. Pilkington, M.: Blockchain Technology: Principles and Applications. Research Handbook on Digital Transformations. Edward Elgar, Northampton (2016)
8. Cucurull, J., Puiggalí, J.: Distributed immutabilization of secure logs. In: Barthe, G., Markatos, E., Samarati, P. (eds.) STM 2016. LNCS, vol. 9871, pp. 122–137. Springer, Cham (2016). doi:10.1007/978-3-319-46598-2_9
9. Haber, S., Stornetta, W.S.: How to time-stamp a digital document. In: Menezes, A.J., Vanstone, S.A. (eds.) CRYPTO 1990. LNCS, vol. 537, pp. 437–455. Springer, Heidelberg (1991). doi:10.1007/3-540-38424-3_32
10. Kleedorfer, F., Panchenko, Y., Busch, C.M., Huemer, C.: Verifiability and traceability in a linked data based messaging system. In: Proceedings of the 12th International Conference on Semantic Systems, SEMANTiCS 2016, pp. 97–100. ACM, Leipzig (2016)
11. Kasten, A., Scherp, A., Schauß, P.: A framework for iterative signing of graph data on the web. In: Presutti, V., d'Amato, C., Gandon, F., d'Aquin, M., Staab, S., Tordai, A. (eds.) ESWC 2014. LNCS, vol. 8465, pp. 146–160. Springer, Cham (2014). doi:10.1007/978-3-319-07443-6_11
12. Kasten, A.: Secure semantic web data management: confidentiality, integrity, and compliant availability in open and distributed networks. Doctoral Dissertation, Universität Koblenz-Landau, Germany (2016)
13. Kasten, A.: A software framework for iterative signing of graph data. GitHub repository (2016). https://github.com/akasten/signingframework. Accessed Jan 2017
14. Sayers, C., Karp, A.H.: Computing the digest of an RDF graph. Technical report, Mobile and Media Systems Laboratory, HP Laboratories, HPL-2003-235, Palo Alto, USA (2004)
15. Weitzner, D.J., Abelson, H., Berners-Lee, T., Feigenbaum, J., Hendler, J., Sussman, G.J.: Information accountability. Commun. ACM **51**, 82–87 (2008)

16. Castelluccia, C., Druschel, P., Hübner, S., Pasic, A., Preneel, B., Tschofenig, H.: Privacy, Accountability and Trust - Challenges and Opportunities. Technical report, ENISA (2011)
17. Accorsi, R.: Log data as digital evidence: what secure logging protocols have to offer?. In: Proceeding of 33rd Annual IEEE International Computer Software and Applications Conference, vol. 2, pp. 398–403. IEEE (2009)
18. Agrawal, R., Evfimievski, A., Kiernan, J., Velu, R.: Auditing disclosure by relevance ranking. In: Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data, pp. 79–90 (2007)
19. Butin, D., Chicote, M., Le Metayer, D.: Log design for accountability. In: Security and Privacy Workshops (SPW), pp. 1–7. IEEE (2013)
20. Tong, Y., Sun, J., Chow, S.S.M., Li, P.: Cloud-assisted mobile-access of health data with privacy and auditability. IEEE J. Biomed. Health Inform. **18**, 419–429 (2014). IEEE
21. Kehoe, L., Dalton, D., Leonowicz, C., Jankovich, T.: Blockchain Disrupting the Financial Services Industry? Deloitte (2015)
22. Libert, B., Beck, M., Wind, J.: How blockchain technology will disrupt financial services firms. Knowledge@Wharton, Wharton University of Pennsylvania (2016). http://knowledge.wharton.upenn.edu/article/blockchain-technology-will-disrupt-financial-services-firms/. Accessed May 2017
23. Nakamoto, S.: Bitcoin: A Peer-to-Peer Electronic Cash System (2008)
24. Antonopoulos, A.M.: Mastering Bitcoin: Unlocking Digital Cryptocurrencies. O'Reilly Media Inc., Sebastopol (2014)
25. Carroll, J.J., Bizer, C., Hayes, P., Stickler, P.: Named graphs. In: Web Semantics: Science, Services and Agents on the World Wide Web, vol. 3, pp. 247–267 (2005)
26. A. Rodriguez, "Restful web services: The basics", IBM developerWorks, 2008
27. English, M., Auer, S., Domingue, J.: Block chain technologies & the semantic web: a framework for symbiotic development. In: Computer Science Conference for University of Bonn Students, Germany, pp. 47–61 (2016)
28. Ugarte R, H.E.: BLONDiE - blockchain ontology with dynamic extensibility. GitHub repository (2016). https://github.com/hedugaro/Blondie. Accessed Feb 2017
29. Bizer, C., Cyganiak, R.: TriG: RDF dataset language. W3C (2013). http://www.w3.org/TR/trig/. Accessed May 2017
30. Sambra, A., Story, H., Berners-Lee, T.: WebId 1.0: web identity and discovery (2014). https://www.w3.org/2005/Incubator/webid/spec/identity/. Accessed Mar 2017
31. Virtuoso Universal Server, OpenLink Software. https://virtuoso.openlinksw.com. Accessed Feb 2017
32. Stathopoulos, V., Kotzanikolaou, P., Magkos, E.: Secure log management for privacy assurance in electronic communications. Comput. Secur. **27**, 298–308 (2008)
33. Manu, S.: Building better blockchains. In: Linked Data in Distributed Ledgers Workshop Keynote, WWW 2017 (2017)
34. Buterin, V.: Ethereum white paper. GitHub repository (2013). https://github.com/ethereum/wiki/wiki/White-Paper. Accessed July 2017
35. Merkle, R.C.: Protocols for public key cryptosystems. In: 1980 IEEE Symposium on Security and Privacy, pp. 122–133. IEEE (1980)