Verifiable Auditing of Access to Confidential Data

Abstract

The sharing of personal data has the potential to bring substantial benefits both to individuals and society, but only if people have confidence that their data will not be used inappropriately. As more sensitive data is considered for sharing (e.g., communication records and medical records) and used to make important decisions, there is a growing need for transparency in the way that the data is processed, while protecting the privacy of individuals and the integrity of their data. We propose a system, VAMS, which allows individuals to check accesses to their personal data, and enables auditors to detect violations of policy. Furthermore, our system protects the privacy of individuals and organizations, while allowing published statistics to be publicly verified. We demonstrate the practicality of our system with two prototypes, based Hyperledger Fabric and Trillian.

1 Introduction

Personal data is playing an increasing role in activities where there is a high cost of failure, such as health-care, the prevention and detection of crime, and legal proceedings. In many important situations, the organizations who need access to this data are not the ones who generate or hold the data, so data must be shared in order for it to be effectively used. Such sharing must however be done with great care as improper sharing or modification of sensitive data could result in harm, whether through breaches of confidentiality or incorrect decisions as a result of tampered data. The harm from such failures can have wider implications than just the individuals whose data is involved. If there is widespread abuse of personal data, people may become unwilling to allow their data to be collected and processed even when it would benefit themselves and society.

Simple restrictions on sharing of personal data can be automatically enforced through access control and cryptographic protections, such as preventing unauthorized parties from accessing databases in which personal data is held. However, other equally important restrictions involve human interpretations of rules or depend on information not available to the computer system enforcing them. For example, access to medical records may be permitted only when it would be in the interests of the patient or access to communication records may be permitted only if it is necessary and proportionate for the purposes of preventing crime. In such cases, rules cannot reliably be automatically enforced in real-time so the approach commonly taken is to keep records of access attempts and subject the actions to audit. Provided that the audit can detect improper activities and violations are harshly punished, abuse can be effectively deterred. Statistics published about the audit can also provide confidence to society that access to data is being controlled and that organizations who can access data will be held to account.

This raises questions about who performs the audit and how the auditor can be assured that the records they see are accurate. If individuals at risk of their personal data being misused do not trust that the auditor is faithfully carrying out their duties then the goal of the audit will not be achieved. However, because of the sensitivity of personal data and the records containing the justification for data being processed, not everyone can act as an auditor. Even if it was possible to find an organization whose audit would be widely accepted, an audit based on tampered records would not be reliable.

When actions are taken on the basis of this data, integrity of the data is also important. Whether making a medical treatment decision or conducting legal proceedings, relying on tampered data may lead to severe consequences. It may be possible to refer back to the organization that collected the data to verify its integrity, but if that organization no longer holds the data or has gone out of business, such verification is not possible. Digital signatures can provide some confidence that data is genuine, but if the private key is compromised then any data signed by that key is subject to doubt, even if it was created before the point of key compromise.

In this paper we propose a system, VAMS, to verify that audits are performed faithfully and are based on accurate records of how personal data was accessed, while protect-

ing the privacy of the individuals and organizations involved. Furthermore, VAMS allows the integrity of personal data to be verified and demonstrated, when necessary.

1.1 Motivating scenarios

Although our system is applicable to many settings that involve the sharing of data between parties, we now briefly introduce to motivating scenarios that are at the origin of this work, law-enforcement access to communications data and access to healthcare records.

Law-enforcement access to communications data In the UK, 95% of serious and organized criminal cases use communications data [32] i.e., metadata temporarily stored by telecommunications providers and requested by law enforcement. At the time of the request, no external oversight takes place but information about the request and the justification for access are stored and made available to an auditor, the Investigatory Powers Commissioner's Office (IPCO), which assesses whether law enforcement make appropriate use of their powers and publishes a yearly report [34].

Communications data is also used as evidence in legal proceedings where a senior representative of the telecommunications provider is asked to verify the evidence in court if questions are raised about its integrity. Assistance from an expert witness may be also requested, which is expensive, time consuming, and impossible if the data has been deleted between the original request and it being required in court.

Industry standards allow providers to sign or hash communications data [21], but if the provider's private key or hash database is compromised, evidence subsequently presented will be brought into doubt even if it was generated before the time of compromise. In this scenario, our system allows the integrity of evidence to be demonstrated, even if the communications provider that produced the data no longer exists or has been compromised. The system will also give assurance to auditors that records of requests to access communications data have not been tampered with, and provide assruance that an auditor's reported statistics are accurate.

Access to healthcare records In a healthcare system, once consent has been given by a patient, various parties should have access to records associated with that patient e.g., their general practioner should have access to scans from a hospital and academics running studies should have access to a patient's relevant records. Currently, patients can only give permission for broad types of activities and may have concerns that their information is used inappropriately. Conversely, patients with serious diseases (e.g., cancer) often have trouble getting the treatment they need, as academics conducting studies are blocked from contacting them and patients are unaware that such studies are going on.

Opening up access to medical databases may fulfill the needs of some patients but it also opens up the potential for abuse, so it is important for patients to have visibility into how their data is used.For clinical practice, the default could be that patients opt in to sharing their data, opting out if they wish. For academic studies and clinical trials the default should be that they are opted out, but can opt in. They can even choose at some granular level which studies they want to opt in e.g., according to the type of study.

An issue with having patients opt in individually is that this process may simply not result in a large enough sample. Equally, patients that are deluged with requests for consent are likely to resort to some default behavior ("click-through syndrome") without really understanding what they have consented to. As such, patients could outsource these decisions to data brokers i.e., organizations that pay attention to the studies being conducted and are authorized to provide consent on behalf of any patients registered with them. Our system can be applied to allow patients to share their data in such a way to protect their privacy, while ensuring that unauthorized parties are prevented from having access and that authorized parties abusing their access can be detected.

1.2 Our contributions

We present a system, VAMS, which provides transparency across the whole timeline of requesting access to data, auditing requests and verifying audits. We show that this is possible by combining existing primitives into a lightweight overlay that can be applied to many systems.

VAMS uses append-only logs of data access requests, instantiated as either blockchain-based distributed ledgers or verifiable log-backed maps. The log, whose integrity properties mean that it can be used as evidence, can be used by users to find requests relevant to them, and auditors to detect misuse or errors in requests. Privacy is attained by ensuring that items on the log are unlikable, and a MultiBallot scheme based on ThreeBallot and association rule mining allows publicly verifiable and privacy preserving statistics. We evaluate the performance and security trade-offs of two sample implementations using Hyperledger Fabric and Trillianas well as our MultiBallot scheme.

2 Related Work

In this section, we examine existing prior work that relates to ours. Our system involves auditability (i.e., verifiability) and privacy requirements, so we focus on work related to this, as well as systems that have been proposed and are similar to ours. In particular, we focus on the requirements of our system, laid out in more detail in Section 4, which are auditability (and more generally, transparency) as well as privacy.

On the privacy side, various works have proposed data anonymization techniques such as k-anonymity [54], l-

diversity [40], t-closeness [41] and ρ-uncertainty [10]. However, as discussed by Domingo-Ferrer and Torra [16] these techniques provide privacy only when the utility of the dataset is significantly reduced. Thus, in most cases the anonymized dataset does not contain enough information to extract statistically significant insights about the population.

Another line of work that attempts to address this limitation is privacy-preserving association rule mining [3] (we introduce association rule mining in Section 5). Such techniques generate randomized or perturbed datasets that protect the privacy of users, while preserving some of the associations between the variables that are of interest. Originally, privacy-preserving association rule mining was performed through uniform randomization of the dataset based on a public factor. As shown by Evfimievski et al. [22] this naive approach does not protect the users' privacy effectively. They instead proposed *randomization operators* [22] that were also proven ineffective and require a initial dataset of at least one million records [58]. Zhang et al. proposed a scheme that considers the existing association rules when perturbing the data and as a result provides better privacy bounds [58]. Unfortunately, this scheme has limited applicability as it severely distorts the strength of the association rules, overestimating strong relationships and underrepresenting less frequent ones. Overall, the weak privacy guarantees and the poor accuracy achieved by those schemes make them unsuitable for our needs.

A more promising line of work is techniques based on differential privacy [19]. Such schemes have been studied extensively in the past years and have been proven to be secure in a variety of settings [18]. However, they still impose trade offs between privacy and utility [5], as well as one-shot and continuous observation [20]. Achieving a meaningful privacy parameter can also be hard in practice [39], particularly when the aim is to provide a general solution like ours. This problem is tackled by Chen et al. [12], who take into consideration the underlying dataset to provide stronger privacy guarantees and increased utility. However, none of these solutions provide verifiability, thus making it hard for the public to verify the integrity of the published data or statistics. In fact, the analyst who adjusts the noise term may accidentally or intentionally sample from distributions that drastically skew the statistics computed [45]. Narayan et al. [45] solve this problem with a scheme that uses a subset of Fuzz [29] to generate publicly verifiable validity proofs. Unfortunately, VerDP has limited expressiveness and severely constrains the access to the dataset. More specifically, once the *privacy budget* of a particular dataset gets depleted, no further queries or analysis can be conducted. This may exclude researchers from using the data and prevent the application of novel analysis techniques on older, depleted datasets.

Finally, there are other systems that rely on append only logs like us. Crosby and Wallach [13] proposed an efficient construction of a log in the case of a untrusted logger serving clients storing events in a log kept honest through auditing. They use a centralized hash-tree based log (which inspired the one used in Certificate Transparency [38]) but do not address secrecy of logged events or replication. Bates et al. [7] look at accountable logs of wiretapping, which are a specific example that would not be admissible in some legal systems e.g., the UK, and are subject to oversight before authorization of requests. We consider sharing of data prior to oversight, which for example in the case of retained communications data is more internationally applicable. CONIKS [42] deals with the specific case of key transparency, allowing users to monitor their key bindinds, and does not deal with other problems we address e.g., verifiable audits.

Two papers that are closer to ours are related to showing compliance with (secret) laws. Goldwasser and Park [26] propose using append-only ledgers (they give Ethereum as an example) and zero-knowledge proofs to provide auditability for actions related to secret laws. In similar work Frankle et al. [25] propose a system that allows accountability of secret processes based on multi-party computations and zero-knowledge proofs. They achieve their goals, but only in a very specific setting that matches the setup of their proposed hierarchical multi-party computation, and propose that compliance with the law be embedded in a cryptographic proof rather than relying on existing auditing infrastructures and process. Their system does not provide as much transparency as ours, and their reliance on zk-SNARKs requires a trusted setup. Morevoer, the requirements (e.g., secret laws) that motivate the use of zero-knowledge techniques are not generally applicable, as data sharing is usually done according to public laws or contracts. Our system is also more scalable, by not requiring zero-knowledge proofs to be generated, stored and verified, and easier to deploy.

3 Background

3.1 Hyperledger Fabric

Hyperledger Fabric (HLF) [6] is a modular, open-source system for deploying and operating permissioned blockchains that differs architecturally from other solutions [9, 56]. A HLF network is composed of peers and an ordering service, with identities assured by a *Membership Service Provider* PKI that maintains a key-value store as the state of the ledger. The state can be updated and queried through transactions on the underlying blockchain.

Peers have identities and can be split up into organizations, as well as roles on the network with regards to transactions. A transaction requires an *endorsing peer* (or many) to executes deterministic *chaincode* (i.e., smart contracts) and sign the transaction containing the resulting state update. Transactions are then sent to the ordering service, which acts as a consensus mechanism and packages transactions into blocks that are committed by *validating peers*, updating the state of

the ledger accordingly. As only endorsing peers are required to execute code for a transaction, other peers do not handle any computational burden other than receiving transactions and block events from the network. The endorsement mechanism also allows for endorsement policies that limit which peers can invoke, or sign transaction for, a certain chaincode.

3.2 Trillian

Trillian [27] is an open-source project that implements a generalization of Certificate Transparency (CT) [38], based on a verifiable log backed by a verifiable map [2].

The verifiable log is an append-only log implemented as a Merkle tree (as in CT) that allows clients to efficiently verify that an entry is included in the log with a proof showing the Merkle path to the tree's entry, detect log equivocation (i.e., conflicting tree heads) and verify that the log is append-only through Merkle consistency proofs. The verifiable map is a key-value store implemented as a sparse Merkle tree i.e., a Merkle tree pre-populated with all possible keys as leaves e.g., all 2^{256} possible SHA-256 hashes. Although a tree with 2^{256} unique leaves would in principle not be practical to compute, only the non-empty leaves have to be computed as all others will have the same value (e.g., zero) [37]. Clients can then verify that a certain value is included (or not) in the map at any point in time, with proofs containing Merkle paths.

Combining a verifiable log with a verifiable map leads to a verifiable log-backed map, where the log contains an ordered set of operations applied to the map. Clients can then verify that the entries in the map they view are the same as those viewed by others auditing (i.e., replaying) the log, allowing clients to trust the key-value pairs returned by the map.

Trillian includes three components: the log of entries, the map and the log of map heads. As it is more centralized, it does not require a form of consensus like distributed ledgers, relying instead on gossip between clients and auditors to detect misbehaving servers by comparing the views of the log they have received. If they detect different views, a cryptographic proof that the server has equivocated exists as every tree head (the root hash of the Merkle tree of log entries) is signed by the server and published to a verifiable log.

3.3 ThreeBallot voting system

ThreeBallot [50, 51] is a paper-based voting scheme proposed by Rivest for end-to-end auditable elections. Voters are given three blank ballots arranged as three columns, each row corresponding to a single candidate. Marking two of the three columns in a row is a vote *for* a candidate, while marking only one of the columns is a non-vote. In the standard version of ThreeBallot, each row must have either one or two marks and blank rows or rows with three marks are not allowed. After the voting process, the collection of all ballots is placed on a public bulletin board, so that anyone can verify the outcome of the elections and check if their vote was represented correctly (while keeping their vote private).

We use ThreeBallot as the starting point for our MultiBallot scheme, presented in Section 5 and Appendix A, which allows us to obtain a privacy preserving dataset that can be used to verify statistics.

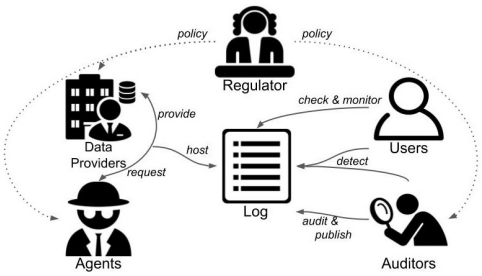

Figure 1: All essential parties in our setting and their functionalities, along with the external regulator and their policy. The optional data broker would act as a user.

4 Setting and Threat Model

4.1 Setting and notation

Our proposed system is composed of agents, data providers, users, auditors, log servers and optional data brokers. External to the system are also regulators that are not active in the system but define regulations (e.g., the Investigatory Powers Act of 2016) that determine the rules obeyed by the parties in the system. Below is a description of the system's parties, functionalities and malicious behaviors, which are also summarised in Table 1.

Agents are parties that request access to user data from data providers (*request*) e.g., law enforcement and medical researchers. A malicious agent's aim would be to access the data without it being logged or to submit an invalid request.

Data providers are parties that collect user data and that receive data requests from agents(*provide*) e.g., telecommunications providers and healthcare providers. A malicious data provider's aim would be to give access to the data without it being logged.

Log servers are responsible for providing access to the log of requests made by agents (*host*). A malicious log server's aim would be to give inconsistent views of the log to auditors and users.

Auditors are parties that audit data access requests (*audit*) and publish statistical reports (*publish*) e.g., the IPCO. They must also be able to detect if log servers are behaving dishonestly (*detect*). A malicious auditor's aim would be to publish an inaccurate report.

Users are members of the public that wish to check the requests that have been made about them (*check*) or that the reports published by the auditor are correct (*monitor*). They

Table 1: The parties in the system, the functions they perform and their malicious behaviour

Party	Function	Malicious behaviour
Agent	*request*: append request to a data provider the log	Providing an invalid request
Data provider	*provide*: answer a request that is on the log	Providing invalid data
	detect: detect if log servers are behaving dishonestly	
User	*check*: look for relevant log entries	Attempting to access requests relevant to other users
	monitor: verify the statistics published by auditors	Attempting to infer information from the statistics
	detect: detect if log servers are behaving dishonestly	
Auditor	*audit*: check the log entries for misuse or errors	Attempting to infer information from the log
	publish: publish statistics about entries on the log	Publishing inaccurate statistics
	detect: detect if log servers are behaving dishonestly	
Log server	*host*: return the log to parties wishing to inspect it	Providing inconsistent views of the log
Data broker	*broker*: respond to requests in place of a user according to their preferences	Misrepresenting the preferences of the user

must also be able to detect if log servers are behaving dishonestly (*detect*). A malicious user's aim would be to learn information about another user through the log e.g., access requests made about other users, or by inferring information from published reports.

Data brokers are non-essential intermediaries that users can rely on to deal with data requests if they are willing to serve as a data provider e.g., providing data to a medical study. The data broker can then deal with these requests (*broker*) according to pre-set rules from the user e.g., type of study they are willing to participate in. A malicious broker's aim would be to misrepresent the user's preference, for example by accepting requests for a study that the user would not want to participate in.

The *log* is a key-value store of, for example, access to data requests. We call log entries *records*, that relevant parties can find on the log by identifying the corresponding keys. Records are tuples of *elements* that are, for example, the attributes of a data request (e.g., type, urgency) or answers to a medical questionnaire. Other information that may be published in the log by auditors are datasets and statistics. These can be simply stored as value in plaintext or, if size is an issue, the value can contain a link to the statistics and dataset along with a hash to verify their integrity.

We assume that all parties may be malicious according to the definitions above, some exceptions. First, we do not allow malicious agents and data providers to collude, as they could then simply bypass the system. Second, malicious servers may attempt to corrupt the system by giving different parties different views of the log, but as this is detectable by auditors and users (see Argument 1), we take them to simply be honest but curious for the rest of this paper. We also take data brokers to be honest, as a user could easily detect if they were misrepresented by checking access' to their data, or simply receiving notifications for the requests that the data broker accepts.

4.2 Security goals

We now define our security goals that are centered around *auditability*, *transparency* and *privacy* guarantees. Users and auditors should be able to access the information that is relevant to them and verify its integrity, allowing for accurate audits. Moreover, information about the state of the system and the actions that take place in it should be visible to all parties, to allow for the system itself to be evaluated. This must of course be balanced with the privacy of the parties involved in the system. We describe each property in more detail below.

Auditability Our auditability goal is to ensure that each of these functions can be performed by the relevant parties, returning the correct results. To ensure that parties have the correct results, they must be able to verify that it is the case i.e., detect any tampering of the information they access, either because they are given an incorrect view of the log by the log server, or because the information on the log has been modified. This is handled by *detect*, *check* and *audit*. The published audits themselves must be auditable i.e., anyone should be able to verify the accuracy of the statistics provided by auditors. This is handled by *monitor*.

Transparency Our transparency goal is an extension of our auditability goal. While auditability ensures that users and auditors can access information relevant to themselves, transparency should ensure that anyone in the system has information about the state of the system and the actions that take place in the system. This allows the system itself to be evaluated, so that if issues arise an argument can be made not only about the specific issue (e.g., an erroneous record) but about the system in general (e.g., systematic errors).

This is handled by ensuring that the system reveals all information that is not strictly required to be confidential i.e., everything but (some of) the record's elements. Information about the state of the system and the actions that take place

can then be evaluated. Moreover, audits reveal general information about the elements of records, and are publicly verifiable so transparency is handled for the system as a whole.

Privacy Our privacy goal is to ensure that despite our auditability and transparency goals, the privacy of parties in the system is preserved. This means that although all records are visible to all parties, the private information they contain should only be visible to the relevant parties, although general information will be available from published audits.

Although the transparency goal is to maximise the information that is available about the system, it does not necessarily conflict with the privacy goal which is to minimise the information available in the system that relates only to a specific party. Instead, achieving transparency and privacy goals in conjunction amounts to ensuring that no information that isn't available by design can be inferred, as in the case of published audits. This reduces the risk of privacy being compromised through inference attacks as is often the case in privacy-preserving systems that fail.

5 Our System: VAMS

We now give an overview of the general requirements for a system aiming to achieve the security goals presented in the previous section. We define the general mechanisms required to attain these, and propose how they can be instantiated in practice. In particular, we show that using a combination of existing schemes can provide all the necessaries functionalities, making our proposed model very realistic.

5.1 System requirements

Looking at the functionalities that the parties in the system must be able to perform and the potential malicious actions of our threat model outlined in Table 1, we now lay out a basic framework that supports the required operations without trusting other potentially malicious parties.

At the center of the system is the log, maintained by log servers performing *host*, which should contain requests submitted by agents through *request*, which is maintained by log servers performing *host*. As auditors and users should be able to detect malicious log servers through *detect* and *detect*, the log must allow them to detect any misbehavior e.g., the log server equivocating, in the form of an incomplete log or altered log entries. Malicious log servers should also not impede the ability of auditors and users to perform audits, so the system should be resilient to some proportion of malicious log servers.

Key-value stores are a natural choice for the form of the log, as records on the log can be tied to unique keys (e.g., identifiers). Retrieving records is then made simple for auditors and users performing *audit* and *check* respectively, as single keys and ranges of key can easily be queried . Performing these functions with integrity involves being able to check that the retrieved records correspond to those originally submitted by agents when *request* is performed and have not been tampered, so we require an append-only log.

The need for an append-only data structure can also be seen in cases where evidence is required. Examples of this (introduced in Section 1) are cases where law enforcement or a healthcare company are required to prove they accessed data with a valid request without a data provider testifying this is the case, or where a data provider must prove they provided data matching the request. In particular, *urgent* requests are authorized orally, with paperwork only retrospectively authorized, so it is not enough attempt to block invalid requests. Requests should be signed, so that they can be used as evidence to assign liability and to hold the relevant parties accountable. This would only work if the evidence produced is robust so that liability can be properly assigned. Evidence should also exist even if the party that produced it is no longer active e.g., if a data provider declares bankruptcy, a public authority is abolished or simply if some servers fail, are destroyed or act maliciously. Thus, log servers should not depend solely on the party tied to the evidence.

Once the requests are records in the log, auditors perform their audits and publish the resulting statistics through *publish*. Users must be able to verify these statistics through *monitor*, there must be evidence of the results they publish, as well as the data necessary to verify their results, without compromising privacy. Moreover, users should also be able to check that their data was included in the data used to obtain the published statistics to ensure that auditors have not simply generated data that fits their statistics.

To summarise, our requirements are as follows. First, a transparency overlay [11] (e.g., a blockchain or Merkle tree) is needed to instantiate the log. Second, a mechanism that allows the log be efficiently queried with integrity and without revealing any link between entries on the log is required to ensure audits are possible and privacy preserving. Third, a mechanism to publish the reports of an audit in a way that they are verifiable and do not reveal any information that was not already obtainable from the log is required.

5.2 Instantiating VAMS

We now describe how solutions to each of the three requirements can be instantiated, and argue that they fulfil the requirements of our security goals. We build VAMS based on primitives introduced in Section 3, namely an append-only log (HLF or Trillian) used as a key-value store and a publicly verifiable privacy preserving statistics scheme inspired by the ThreeBallot voting scheme. We supplement this with a simple scheme that allows entries on the log to be identifiable by relevant parties and unlinkable to others. Security arguments are presented in the subsequent subsection.

Appending to the log When performing *request*, agents append requests to the log by assigning a request to a key in the key-value store. Our auditability goals require that the requests can then be audited by the parties they pertain to i.e., users performing *check* to find requests for their data and auditors performing *audit*. Moreover, our privacy goals also require that the private contents of the requests are not visible to any other parties, and that information cannot be inferred about the requests by linking them with other requests, users, agents or data providers. We also aim to make audits easy, so the information required by users to check requests should not rely on other parties e.g., communicating with agents or data providers to find the requests.

To this end, we opt for the use of straightforward encryption that only requires information that we can reasonably assume to be known by agents, data providers and users tied to a request. Specifically, we assume that agents and data providers refer to a given user using private agent and data provider identifiers that we denote id_a and id_{dp}, we also assume that these are known to the user. It is then possible to obtain a *common identifiers* $id_c = Enc(id_a, id_{dp} \| n)$, for example by using AES, where n is a *session identifier* that changes deterministically with every request involving the same pair (id_a, id_{dp}). This ensure different requests involving the same parties are unlinkable, but allows users to check relevant requests without communicating with agents and data providers, which also reduces the risk of information leaking. The private contents of the requests are then simply encrypted under the public keys of the user and auditors.

Because id_a and id_{dp} may be short and have low entropy, a key derivation function (KDF) such as PBKDF2 [43] should be used to obtain a more resilient ciphertext, with id_a being fed through the KDF before being used as an encryption key. It is also not unrealistic for many agents and data providers to provide resilient values id_a and id_{dp} to users, for example long pseudorandom strings or passwords adhering to modern security guidelines. This would increase the burden of key management but software like password managers could be relied on. The German Electronic Identity Card, for example, can be used for online authentication and has been analysed from a cryptographic point of view [14].

Querying the log Once requests are logged, users and auditors can verify that the log servers are not malicious by performing *detect*, then perform *audit* and *check* as required. This fulfils part of our auditability goals, which also require that users and auditors have access to untampered data. Accessing the key-value store, they can assure themselves that the information obtained from the log is correct due to the integrity properties of the log and audit requests. Auditors perform their task over the entire log, or the subset of the log that they have not yet audited, but users look only for specific requests. To do so, they must iterate over possible values of the session identifier n until no request is found, to determine the possible requests relevant to them.

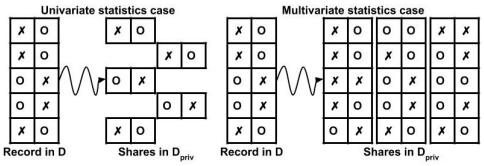

Figure 2: Constructing shares in D_{priv} from a record in D. The left side illustrates splitting the elements of the record into individual shares and the right side illustrates generating three shares from a record.

If users do not wish to take on this computational burden, they can choose to outsource this role to a data broker. These parties act as intermediaries between agents and users that would otherwise perform *provide*, and also allow users to act as data providers if they are willing to, for example, participate in a study. One downside of this is that the data broker must then be trusted with the private identifier tied to requests that the user positively answers. However, no other trust is required as VAMS allows the user to check the activity of the broker, which will be logged and be auditable under the same guarantees as other log entries.

Publishing and verifying audits Auditors are expected to perform *publish*, revealing statistics computed as a result of their audit. An example of what might be published are the statistics provided by the IPCO in its annual report [34], or results of a study in the healthcare scenario. Our auditability goals require that these statistics be verifiable, but for operational and privacy reasons the original data used to compute statistics cannot be published. In order to accomplish our goals, we propose a way of publishing a transformed dataset that can be used by users performing *monitor* to verify the data used to compute the statistics, recompute the statistics themselves and, in accordance with our privacy goals, cannot be used for anything else than recomputing statistics.

Our scheme is based the simple idea of generating a dataset D_{priv} of *shares* from the original dataset D of records, where each record $R_{i\in[1,n]}$ is comprised of multiple elements that may, for example, note the existence or absence of a particular gene or mutation. In both cases, the only overhead required is to generate the shares, which is not very costly.

The simplest case is that of univariate statistics, which can be solved by generating a privacy-preserving dataset D_{priv} by splitting each record into shares, one for each element of the record as in Figure 2, preventing any inference or information leakage from correlations between the shares. Each share is tagged with the element type and a unique share identifier $id_{share} = Hash(id_c|i)$ constructed from the common identifier id_c of the user and index i of the share. The auditor then adds D_{priv} to the ledger along with the statistics resulting of the analysis. Users can then check the presence of

their shares in D_{priv}, and re-compute the published statistics from D_{priv} to check their validity.

The case of multivariate statistics is a bit more complicated. We introduce a MultiBallot scheme based on ThreeBallot (introduced in Section 3.3) that can be used to generate D_{priv}, by generating from each record an odd number $n = 2k+1$ of shares comprised of as many elements as the original record. The constraint on these shares is that given an element e in a record, $k+1$ shares them include e and k shares must include $\overline{e}$, the false value for e, as illustrated in Figure 2. This preserves the accuracy of multivariate statistics, while protecting the privacy of users by making it hard to reconstruct the original record. Auditors can then publish D_{priv}, so that users can verify that their shares are accurately represented and that the statistics are correct.

Appendix B shows how the scheme can be applied in the context of associated rule mining, allowing results of an analysis of D to be estimated from D_{priv}. By generalising the ThreeBallot scheme to a MultiBallot scheme with n shares we allow more control of the privacy levels to accommodate a wider variety of use cases.

5.3 Security of VAMS

We now discuss the security of VAMS with respect to the goals introduced in Section 4, namely auditability, transparency and privacy.

Auditability Attaining our auditability goal requires that the relevant parties (i.e., users and auditors) be able to audit any action in the system that pertains to them. In the case of VAMS, this means being able to ensure that the view of the log returned by the log server is correct, that the information contained on the log is correct and that it is accessible by the parties it is relevant to, and that the results of an audit can be verified. We split this into three arguments.

Auditability argument 1 (Detecting log misbehavior). We argue that an adversary controlling a malicious log server is detectable i.e., an auditor or user performing *detect* returns 0. Log server misbehavior corresponds to equivocation.

In the HLF case, either the ordering service maintains consensus, and there is no equivocation, or there is a fork of the blockchain. In that case, both the main chain and the alternative chain are visible, so equivocation can be detected.

In the Trillian case, a log server that equivocates would have to produce signed tree heads and Merkle consistency proofs for the alternative Merkle trees. Different Merkle consistency proofs leading from the same Merkle tree generate different views of the log, but these differing logs can no longer accept the same Merkle consistency proofs to extend the logs because the leaves are different. As tree heads are signed by the log server, two inconsistent tree heads can be used as evidence to implicate the log server [11, 17].

Auditability argument 2 (Auditability of the log). We now argue for the auditability of the system for auditors and users i.e., that the results of an audit is correct with respect to the view of the log provided by honest log servers.

For auditors, as we assume that data providers do not collude with agents, the log will be complete in terms of the requests that have been made. By the previous argument, we can also assume that the auditor's view of the log server is correct, otherwise the log server can be implicated and removed from the system. Finally, any attempt to modify information on the log also appears in the log due to the completeness properties of both HLF and Trillian.

In the HLF case, we rely on the integrity properties of the underlying blockchain that records state updates. Auditors can obtain the available key-value history function to obtain the transactions that have modified the value of a key. If they do not trust the integrity of that function (the code for which is public), they have access to the blockchain and can inspect it, replaying transactions and detecting a party's misbehavior as they will have signed the relevant transactions.

In the Trillian case, we rely on the integrity properties of the underlying Merkle tree, and the Merkle consistency proofs that give the append-only property of the tree. In the event that a malicious party has tried to tamper with requests, they will have to update a request value, which will appear in the append-only log. If the log server produces a new tree head for a tree that modifies requests in the tree associated with the previous tree head, there cannot be a Merkle consistency proof between the two trees, so it will be detectable. Similarly, if a leaf of an existing tree is removed, the Merkle root of the tree will no longer match the leaves.

Auditors can then perform *audit* by querying the state of the ledger or log-backed map containing the requests, which are encrypted under their public keys, and perform their analysis as required. Requests that cannot be decrypted can simply be classed as invalid and reported. The same argument can be used for users performing *check*.

Auditability argument 3 (Auditability of published audits). In the case of a user performing *monitor*, we again have that the user has a correct and complete view of the log by following through the arguments previously presented.. A malicious auditor could nonetheless perform *publish* maliciously, publishing incorrect statistics or the wrong dataset, but this would be detected by a user performing *monitor*. A user that was included in the used dataset D used can check the integrity of the transformed dataset D_{priv}, identifying their shares to verify their correctness. Once the integrity of the data is confirmed, any other user can replicate the computations of the analysis and compare their results with those published, detecting any false statistics.

Finally, if malicious auditor does not generate shares randomly so that D_{priv} might reveal information, users performing *monitor* can handle this by checking that the distribution of shares in D_{priv} is as expected.

Transparency

Transparency argument 1. The argument that VAMS provides transparency follows from what has already been said in the auditability argument above. Transparency is achieved if not only the users and auditors see information pertaining to themselves, but if they also more generally have information about the system and the functions performed in the system. VAMS achieves this for the following reasons.

First, although the information inside records is encrypted, records themselves are visible to all parties in the system, which can be anyone was we do not place any restrictions on who can be a user. This means that anyone can see the activity going on in VAMS. Second, although information in the records is encrypted, audits reveal aggregate information about the system as a whole. Moreover, this information is publicly verifiable due to the availability of D_{priv}, which comes with integrity properties. This means that anyone can see the overall results of the actions taking place in VAMS.

Privacy

Privacy argument 1 (Privacy of the log). We start by arguing that the log does not compromise the privacy of the parties in VAMS. Because confidential information in the records is encrypted, we focus on the unlinkability of records between themselves, and to users, agents or data providers. We assume that AES is a secure pseudorandom permutation and that session identifiers n are used only once for each pair of private identifiers id_a and id_{dp}, and argue it is not possible to link two or more requests (i.e., common identifiers) that appear in the log except for the relevant users.

A user or an adversary knowing id_a and id_{dp} will be able to find requests relevant to the user with probability 1 by performing *check*. In the case of an adversary knowing only one of id_a and id_{dp}, determining the second identifier would require iterating over the possible values of the unknown identifier (2^b values for a bitstring identifier of length b) and the range of the session identifier i.e., $\mathcal{O}(2^b \cdot range(n))$ encryption operations. This would reveal only one pair of private identifiers, giving the adversary only limited information. If neither id_a or id_{dp} are known to the adversary, another factor of 2^b must be taken into account.

Moreover, if the adversary considers two common identifiers, the security of AES is enough to argue that the adversary will not be able to determine if they shared one or more inputs with any reasonable probability. In the case of agent and data provider unlinkability, each pairing is equally likely. Thus, the probability of correctly determining a pair is $\frac{1}{|\mathsf{A}||\mathsf{DP}|}$, where $|\mathsf{A}|$ and $|\mathsf{DP}|$ are the size of the sets of agents and data providers, respectively.

Privacy argument 2 (Privacy of the published audits). We now argue that published audits i.e., the statistics and the dataset D_{priv}, do not reveal more than the statistics themselves. An adversary could have two kinds of information, knowing up to $n-1$ shares that were generated from a user's record or up to $e-1$ elements of a user's record. In both cases the adversary will take the information they have and try to determine all of the user's shares to reconstruct their record. We consider our scheme to be privacy preserving if this cannot happen with any reasonable probability.

Table 2: Number of elements for 3Ballot and 5Ballot such that the probability of a successful reconstruction is less than 0.01%. The numbers in brackets indicate the adversary's advantage (i.e., known shares) and the probability of success.

Scheme	10 users	100 users	1000 users	10 000 users
3Ballot (1)	3 ($3 \cdot 10^{-5}$)	6 ($5 \cdot 10^{-13}$)	10 ($6 \cdot 10^{-10}$)	14 ($1 \cdot 10^{-7}$)
3Ballot (2)	1 ($8 \cdot 10^{-5}$)	2 ($2 \cdot 10^{-12}$)	4 ($2 \cdot 10^{-10}$)	6 ($5 \cdot 10^{-9}$)
5Ballot (1)	6 ($4 \cdot 10^{-6}$)	11 ($5 \cdot 10^{-20}$)	17 ($3 \cdot 10^{-12}$)	23 ($2 \cdot 10^{-13}$)
5Ballot (4)	1 ($5 \cdot 10^{-5}$)	2 ($3 \cdot 10^{-9}$)	3 ($2 \cdot 10^{-17}$)	5 ($3 \cdot 10^{-7}$)

The case of univariate statistics is straightforward as shares are split from each other. As long as the hash function used to tag the shares is pre-image resistant they will be unlinkable, and the adversary is left to pick shares at random.

The more interesting case is that of multivariate statistics and our MultiBallot scheme, where shares contain multiple elements and could therefore be more useful. A successful attack would amount to being able to reconstruct a user's record i.e., identify the shares generated from the record. Henry et al. [31] previously considered reconstruction attacks on the ThreeBallot scheme, computing the adversary's probability of a success given that they have one share.

As we have generalised to the case of any odd number n of shares, we extend their argument to our case and compute new bounds on the safe number of elements that can be include in a share of D_{priv} for different values of n, users, and information (i.e., shares) known to the adversary. These are computed by calculating the number of possible shares and the probability of these shares in a MultiBallot scheme, then calculating the probability that combinations of shares will be valid and from that the probability of a successful reconstruction attack. Details and explicit formulas are given in Appendix A. We present bounds for 3 and 5Ballot in Table 2, where it is easy to see that increasing the number of shares per ballot noticeably increases the number of elements that a record can contain.

Different bounds can be chosen depending on the required level of privacy i.e., the acceptable probability of a reconstruction. In practice, only a few elements may be relevant to the results of an audit or study, and only those need to be published for the relevant statistics to be publicly verifiable. Finally, we have assumed statistical independence between elements, which may not always be true. Fortunately, ThreeBallot with correlated ballots was studied by Strauss [52] who showed that even heavily correlated races (elements) had only a minor effect on the security of the scheme.

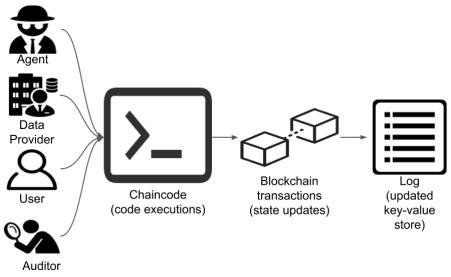

Figure 3: The Hyperledger Fabric based implementation.

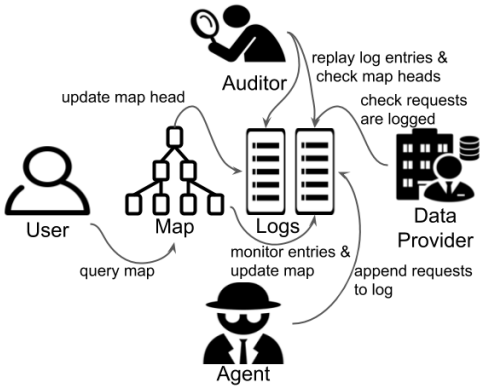

Figure 4: The Trillian based implementation.

6 Implementation and Performance

We now present an evaluation of two implementations of VAMS, one based on Hyperledger Fabric and one based on Trillian. Both are evaluated on modest Amazon AWS t2.medium instances.[1] The ThreeBallot-based privacy scheme is also evaluated. The code for each of these will be open-sourced after publication.

6.1 Implementations

Hyperledger Fabric implementation Rather than being an out of the box solution, Hyperledger Fabric (HLF, introduced in Section 3.1) is a modular platform on top of which systems can be built, making it suitable choice to implement VAMS. As HLF is under ongoing development by IBM, improvements in scalability, privacy and integrity can be expected, allowing for more potential uses.

We implement a test network as proof of concept, with seven separate machines that represent four peers (an agent, a data provider, a user and an auditor), an ordering service (an Apache Zookeeper service and a Kafka broker), and a client from which commands are sent to peers. The network maintains a key-value store that can be populated by requests linked to common identifiers. These requests can then easily be retrieved by querying specific keys or a range of keys in lexical order. A key history function is also available. State updates correspond to transactions on the underlying blockchain, making them verifiable.

For this simplified implementation, all peers are connected to the same channel and there is a single chaincode containing four functions. The first is used to update the state of the ledger (as part of *request*), the second is used to retrieve a range of key values (as part of *audit*), the third is used to retrieve values for specific keys (as part of *check*) and the fourth is used to retrieve a key's history (as part of *audit* and *check*) to see which blockchain transaction resulted in state updates for a given key. The transactions that result in state updates of the log (i.e., changing the value of a key) correspond to chaincode invocations, which are recorded on the underlying blockchain. Chaincode invocations that only query the log (i.e., do not update the value of a key) are also recorded.

Peers are members of organizations (one corresponding to each role) and have identities, X.509 public key certificates [57], and sign transactions accordingly. The public key certificates are issued by the HLF CA, but any organization can specify the CA they wish to issue certificates from, or use another public key infrastructure (PKI) if they wish. Signatures are checked as part of the transaction process, as chaincode invocations must be endorsed (signed) by the appropriate parties. In our implementation, these are the peers invoking the chaincode. Thus, auditors or users can check the transactions that updated the value of a key and easily determine the agent responsible for the update, as they will have endorsed the transaction.

As endorsement policies can require multiple signatures, they could hold multiple parties accountable e.g., if data providers were considered responsible for accepting invalid requests, they could be required to sign the corresponding request transactions. An ordering service of specific peers (e.g., auditors) could also be used to detect and flag invalid requests as they are initially processed (and endorsement policies are checked) before committing the requests. These are not present in our implementation, but give an idea of what may be possible as Hyperledger Fabric undergoes continued development and implements further cryptographic tools.

Trillian implementation The second implementation of the system, summarized in Figure 4, is based on Trillian's verifiable log-backed map (introduced in Section 3.2) as its underlying data structure.

As part of *request*, agents append signed requests they have sent to the log, which data providers can then check. There is no built-in identity system, so the log server service responsible for receiving new requests must check that they

[1] Each instance has 2 vCPUs, 4GB of memory and is running Linux 16.04 LTS with Go 1.7, docker-ce 17.06, docker-compose 1.18 and Fabric 1.06 installed.

are signed. A map server then monitors the log for new entries, and updates the map according to the new entries–the common identifiers are used as the keys in the map. It then periodically publishes signed map heads that are written to the second verifiable log, solely responsible for keeping track of published signed map heads. To perform *check*, users can then query the map to efficiently check their possible common identifier values. The map will return a Merkle proof of non-inclusion for common identifiers that do not map to requests (i.e. the common identifier maps to a zero value), or a Merkle proof of inclusion of requests that the common identifiers do map to. Auditors performing *audit* can in turn check that the map is operated correctly by replaying all log entries, verifying that they correspond to the same map heads that were written to the second verifiable log tracking signed map heads.

6.2 Trade-offs and performance

Trade-offs We now compare both implementations. Trillian has the advantage of having a higher transaction throughput, because no consensus is required among different nodes to agree on the ordering of transactions. Trillian also has better user auditability, because when a user queries the map server for an id_c, the map server returns a Merkle proof of the key and value being included in the map. HLF does not support this, requiring users to replay the entire blockchain in order to verify the inclusion of a key and value. This could be managed if "light clients" were introduced (as in Ethereum). Users could also decide to outsource this task to a trusted data broker, or multiple if they believe that a majority are honest.

HLF supports flexible chaincode policies for governing write access to the log, as it comes with built-in authentication and public key infrastructure known as an identity service. Authenticating must be done separately in Trillian. However, this means that in HLF users must submit queries to audit the log using a key pair associated with their pseudonymous identity, so if they used the same identity for multiple queries, their common identifiers could be linked together.

The two systems also differ in their decentralised (HLF, even though it is permissioned) or centralised (Trillian) approach. A decentralised approach is appealing as it reduces the trust required in single entities. In practice, however, there is only one organisation that legitimately has reason to write records for a particular business relationship. Users will mostly only have a single data provider for a service, which may lend itself more towards the centralised approach.

Table 3 summarizes the features of both implementations Ultimately, Trillian is easier to deploy and has less setup than Hyperledger, as HLF requires the setup of a network of multiple nodes to act as peers, and the maintenance of an identity service to allow nodes to interact with the network.

Table 3: Summary of features for the Hyperledger Fabric and Trillian based implementations.

Features	HLF	Trillian
User privacy	●	●
Agent privacy	◐	●
Data provider privacy	◐	●
Statistical privacy	●	●
User auditability	◐	●
External auditability	●	●
Verifiability	●	●
Access control	●	◐

Table 4: Micro-benchmarks of basic operations for the Hyperledger Fabric and Trillian based implementations. The maximal throughput values are given for a batch size of 1 in the HLF case, and a batch size of 300 in the Trillian case.

Measures	HLF	Trillian
State update (per id_c)	65ms	35ms
Request retrieval (per id_c)	66ms	14ms
Max throughput	40	102

Micro-benchmarks Table 4 presents micro-benchmarks for basic operations. These include state updates (i.e., adding a request as part of *request*), state retrievals (i.e., retrieving requests as part of performing *check*) and the maximal throughput for each system with a batch size (i.e., requests per state update) of one. In the case of state updates and retrievals, the results were obtained by averaging over 500 operations. In both cases, the average for each operation are a few dozen milliseconds. Note that for the HLF system, the results include the time required to create and submit 500 blocks, chaincode execution alone is otherwise under $10ms$. For state retrievals, HLF allows retrieving the values for a range of keys. This scales linearly with the number of values retrieved and only requires one transaction.

Throughput Table 4 also includes the maximal throughput, which is 40 for the HLF system and 102 for the Trillian system. A plot of throughput for different batch sizes is also presented in Figure 5.

For the HLF system, the highest throughput is observed for lower batch sizes, where the bottleneck is simply the client sending requests. Throughput then lowers slightly as batch size increases. For the Trillian system, the batch size of the verifiable derived-map implementation determines how many items at a time the map servers retrieves from the log

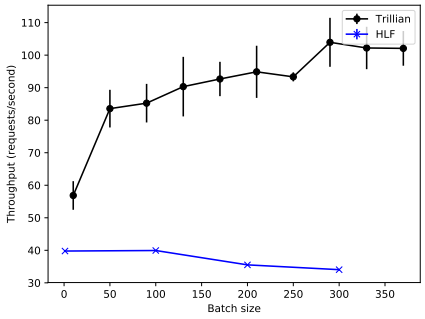

Figure 5: Throughput evaluation for both implementations.

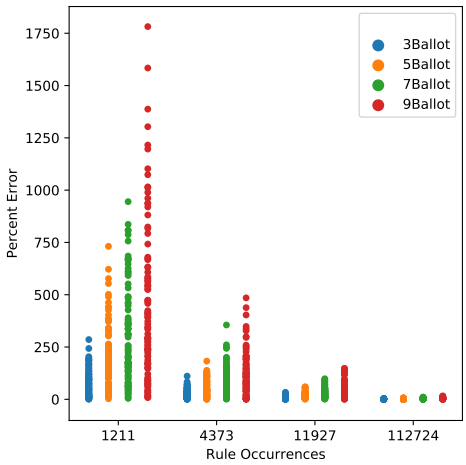

Figure 6: Percent error for the support over two elements as rule occurrences vary in the case 3, 5, 7 and 9Ballot.

to update the map's key-values, until around batch size 300. The bottleneck is then the number of keys updated by the map server per second, and throughput levels out.

There are, however, trade-offs to consider between batch size and throughput. Although a higher throughput that may be obtained with a larger batch size, having requests appear on the system sooner than later may be advantageous for some use cases of our system, and certainly the motivating examples of law enforcement and healthcare. Thus, a lower batch size may be advantageous to ensure requests appear as soon as possible, particularly for urgent requests. A batch timeout can also be used as a compromise, such that a high batch size can be chosen with a guarantee that a request will appear after a time limit if the batch size limit is not reached.

We may also look at the case of law enforcement for indications, using figures from the 2016 UK IPCO report (neither in the healthcare setting, nor for the use of US administrative subpoenas are there equivalent publicly available statistics). There are about 750 000 requests for communication data per year in the UK [33], or 1 request every 9 seconds assuming requests happen during working hours. In this case, a HLF-based server capable of 40 requests per second, placed at the interface for law-enforcement (standardized by ETSI TS 103 307 [21]) would be more than sufficient, with an average waiting time of 25 ms assuming Poisson-distributed requests. For a Trillian-based system with 102 transactions per second the average waiting time would be 10 ms.

Accuracy of MultiBallot statistics To evaluate the effectiveness and applicability of our scheme, we measure the accuracy of the rule association metrics computed on D_{priv}. For our experiments, we generate multiple synthetic datasets with several frequent element sets [30], then mine those itemsets using the Apriori algorithm [4], which works by identifying frequent elements in the dataset and extends them to larger element sets for as long as the element sets appear frequently enough in the dataset. The generated datasets follow the structure of D (described in Section 5). We then compute the support and confidence measures on D_{priv} for the previously extracted element sets, comparing those values with the reported values for the same element sets on D. For this purpose, we use the percent error of the measures.

Element sets are commonly extracted both in the interception use case e.g., proportion of urgent requests, analysis of request rejections, errors and recommendations [33], and the healthcare use case e.g., proportions of people registered with diabetes that achieved blood glucose, pressure and cholesterol targets [46]. We opt to use synthetic datasets to examine the accuracy offered by our ThreeBallot scheme more thoroughly, by simulating different scenarios rather than relying on public datasets that have already been sanitized. However, we also verify our reported results using commonly used public datasets, such as the Extended Bakery dataset [15] and the T10I4D100K dataset [24]. In all our experiments, we measure the error for both the support and the confidence metrics. We include only the graphs for support, as those for confidence are identical. Each experiment is repeated 100 times.

In our first experiment, we study the percent error for the support over two elements when varying the number of rule occurrences for a dataset of 1 million records. As seen in Figure 6, which shows the results in the case of 3, 5, 7 and 9Ballot, element sets that occur less often are prone to higher percent error, with a high variance in the reported support values. As element sets become more frequent (up to around 11%), the percent error ($< 2\%$) and the variance both shrink. The difference in percent error between MultiBallot schemes also shrinks and for the sake of exposition we highlight only the results for 3Ballot in our next experiments.

In our second experiment, we examine if the scheme's accuracy for an element set depends on the number of times the element set occurs, or its occurrences relative to the overall number of users (i.e., support). We generate four datasets of

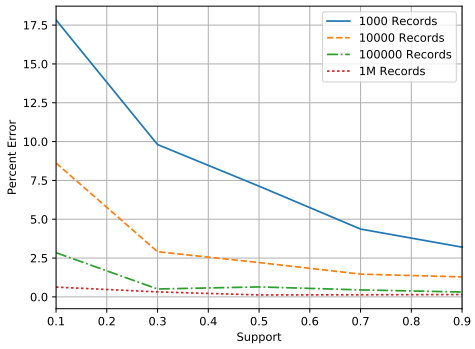

Figure 7: Percent error for elements that appear with varying frequency in datasets with different number of users in the 3Ballot case.

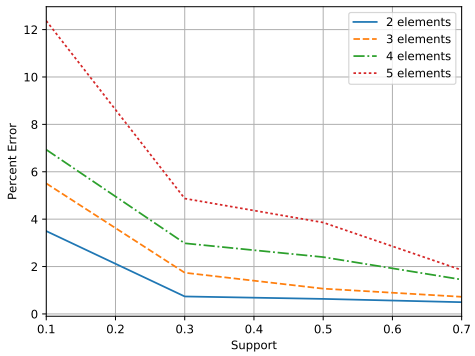

Figure 8: Percent error for element sets of varying size that appear with the same frequency i.e., have the same support, in the 3Ballot case.

various size ($1k$, $10k$, $100k$, $1M$ users), and pick five element sets with support 0.1, 0.3, 0.5, 0.7, 0.9 from each dataset. Figure 7 presents the percent error for those element sets in every dataset. The percent error shrinks as the support increases, but the absolute size of the element set seems to play a more decisive role in the accuracy of the statistics. In the cases of the $100k$ and $1M$ user datasets, the support seems to have a minimal effect on the accuracy. These results are in accordance with the work of Blum et al. [8, Section 3.1.1].

Our final experiment evaluates the performance of our ThreeBallot scheme for different element set sizes using a synthetic dataset of 100k users. As seen in Figure 8, the scheme's accuracy is sensitive to increases in the number of elements. This is expected as the scheme probabilistically estimates the field values of the original record R_i, based on the observed share. Understandably, the inference error for each field adds up with the number of elements.

Based on the above results and the list of statistics reported in the IPCO report [33], our ThreeBallot scheme is suitable for the law-enforcement use case. To better evaluate its suitability for healthcare data, we now look into the relevant medical and biostatistics literature. We consider two types of studies: Studies on Genes and Protein networks, and Epidemiology studies. In the first type, datasets commonly contain between 100,000 and a few million records, while the support threshold is usually around 0.5%. In most cases, valid association rules are comprised of only two elements, while their support is higher than the minimum threshold. This is important as the minimum threshold is relevant only during the rule mining phase, while in the verification phase the users compute measures over the relationships that are reported by the researcher as strongly associated [23, 28, 36, 44]. In epidemiology studies, the average element set size is 3, while the minimum support is around 1%. However, the support of relevant element sets identified is much higher and ranges from 1% to 16%, while datasets contain between 10,000 and 250,000 records [35, 47, 55]. We conclude that our ThreeBallot scheme is also suitable for these types of studies, with a slightly higher expected percent error compared to the law-enforcement use case.

6.3 Deployability

For a system like VAMS to be deployed, agents and data providers would need to implement the necessary infrastructure. They may do so voluntarily, so as to increase public confidence that their access to personal data is legitimate [53]. Participants may also implement VAMS to allow them to demonstrate that personal data has not been tampered with when used as evidence, with the transparency properties just being a desirable side-effect of this activity.

Alternatively, participants may have a statutory obligation to provide transparency e.g., compliance with ETSI requirements may be a condition of providing a telecommunication service, and their standards do include provision for requiring authenticity and transparency of access to personal data [21]. In the UK, the IPCO can require that public authorities and telecommunication operators provide the commissioner's office with any assistance required to carry out audits and this could include implementing IT infrastructure [1, Section 235(2)]. Another possible route for imposing a statutory requirement to provide transparency could be through enforcement action by a regulator such the Federal Trade Commission or a data protection authority.

7 Conclusion

We have proposed and implemented (twice) a system, VAMS, which achieves our auditability and privacy goals, based on realistic use cases. Our results illustrate that the current framework for requesting data can be greatly improved to benefit all parties involved. In particular, VAMS does not have to replace any existing component in the workflow of an organization. Instead, it serves as an overlay that can be used to achieve transparency and privacy goals.

References

[1] Investigatory Powers Act, 2016.

[2] ADAM EIJDENBERG, B. L., AND CUTTER, A. Trillian – verifiable data structures, 2017.

[3] AGRAWAL, R., IMIELIŃSKI, T., AND SWAMI, A. Mining association rules between sets of items in large databases. In *ACM SIGMOD Record* (1993), vol. 22, ACM, pp. 207–216.

[4] AGRAWAL, R., AND SRIKANT, R. Fast algorithms for mining association rules. *Proc. of the 20th VLDB Conference* (1994), 487–499.

[5] ALVIM, M. S., ANDRÉS, M. E., CHATZIKOKOLAKIS, K., DEGANO, P., AND PALAMIDESSI, C. Differential privacy: on the trade-off between utility and information leakage. In *International Workshop on Formal Aspects in Security and Trust* (2011), Springer, pp. 39–54.

[6] ANDROULAKI, E., BARGER, A., BORTNIKOV, V., CACHIN, C., CHRISTIDIS, K., DE CARO, A., ENYEART, D., FERRIS, C., LAVENTMAN, G., MANEVICH, Y., MURALIDHARAN, S., MURTHY, C., NGUYEN, B., SETHI, M., SINGH, G., SMITH, K., SORNIOTTI, A., STATHAKOPOULOU, C., VUKOLIĆ, M., WEED COCCO, S., AND YELLICK, J. Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains. *ArXiv e-prints* (Jan. 2018).

[7] BATES, A., BUTLER, K. R., SHERR, M., SHIELDS, C., TRAYNOR, P., AND WALLACH, D. Accountable wiretapping–or–i know they can hear you now. *Journal of Computer Security 23*, 2 (2015), 167–195.

[8] BLUM, A., DWORK, C., MCSHERRY, F., AND NISSIM, K. Practical privacy: the sulq framework. In *Proceedings of the twenty-fourth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems* (2005), ACM, pp. 128–138.

[9] CACHIN, C. Architecture of the Hyperledger blockchain fabric. In *Workshop on Distributed Cryptocurrencies and Consensus Ledgers* (2016).

[10] CAO, J., KARRAS, P., RAÏSSI, C., AND TAN, K.-L. ρ-uncertainty: inference-proof transaction anonymization. *Proceedings of the VLDB Endowment 3*, 1-2 (2010), 1033–1044.

[11] CHASE, M., AND MEIKLEJOHN, S. Transparency overlays and applications. In *ACM CCS 16* (Vienna, Austria, Oct. 24–28, 2016), E. R. Weippl, S. Katzenbeisser, C. Kruegel, A. C. Myers, and S. Halevi, Eds., ACM Press, pp. 168–179.

[12] CHEN, R., MOHAMMED, N., FUNG, B. C., DESAI, B. C., AND XIONG, L. Publishing set-valued data via differential privacy. *Proceedings of the VLDB Endowment 4*, 11 (2011), 1087–1098.

[13] CROSBY, S. A., AND WALLACH, D. S. Efficient data structures for tamper-evident logging. In *USENIX Security Symposium* (2009), pp. 317–334.

[14] DAGDELEN, Ö. *The cryptographic security of the German electronic identity card*. PhD thesis, Technische Universität Darmstadt, 2013.

[15] DEKHTYAR, A., AND VERBURG, J. Extended bakery dataset. https://wiki.csc.calpoly.edu/datasets/wiki/ExtendedBakery, 2009.

[16] DOMINGO-FERRER, J., AND TORRA, V. A critique of k-anonymity and some of its enhancements. In *Availability, Reliability and Security, 2008. ARES 08. Third International Conference on* (2008), IEEE, pp. 990–993.

[17] DOWLING, B., GÜNTHER, F., HERATH, U., AND STEBILA, D. Secure logging schemes and certificate transparency. In *ESORICS 2016, Part II* (Heraklion, Greece, Sept. 26–30, 2016), I. G. Askoxylakis, S. Ioannidis, S. K. Katsikas, and C. A. Meadows, Eds., vol. 9879 of *LNCS*, Springer, Heidelberg, Germany, pp. 140–158.

[18] DWORK, C. Differential privacy: A survey of results. In *International Conference on Theory and Applications of Models of Computation* (2008), Springer, pp. 1–19.

[19] DWORK, C., MCSHERRY, F., NISSIM, K., AND SMITH, A. *Calibrating Noise to Sensitivity in Private Data Analysis*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006, pp. 265–284.

[20] DWORK, C., NAOR, M., PITASSI, T., AND ROTHBLUM, G. N. Differential privacy under continual observation. In *Proceedings of the forty-second ACM symposium on Theory of computing* (2010), ACM, pp. 715–724.

[21] ETSI. TS 103 307: Security aspects for LI and RD interfaces, 2018. V1.3.1.

[22] EVFIMIEVSKI, A., SRIKANT, R., AGRAWAL, R., AND GEHRKE, J. Privacy preserving mining of association rules. *Information Systems 29*, 4 (2004), 343–364.

[23] FARIA, D., SCHLICKER, A., PESQUITA, C., BASTOS, H., FERREIRA, A. E., ALBRECHT, M., AND FALCÃO, A. O. Mining go annotations for improving annotation consistency. *PloS one 7*, 7 (2012), e40519.

[24] FLOUVAT, F., DE MARCH, F., AND PETIT, J.-M. A thorough experimental study of datasets for frequent itemsets. In *Data Mining, Fifth IEEE International Conference on* (2005), IEEE.

[25] FRANKLE, J., PARK, S., SHAAR, D., GOLDWASSER, S., AND WEITZNER, D. J. Practical accountability of secret processes. Cryptology ePrint Archive, Report 2018/697, 2018. https://eprint.iacr.org/2018/697.

[26] GOLDWASSER, S., AND PARK, S. Public accountability vs. secret laws: Can they coexist?: A cryptographic proposal. In *Proceedings of the 2017 on Workshop on Privacy in the Electronic Society* (2017), ACM, pp. 99–110.

[27] GOOGLE. Trillian, 2017.

[28] GUZZI, P. H., MILANO, M., AND CANNATARO, M. Mining association rules from gene ontology and protein networks: Promises and challenges. *Procedia Computer Science 29* (2014), 1970–1980.

[29] HAEBERLEN, A., PIERCE, B. C., AND NARAYAN, A. Differential privacy under fire. In *20th USENIX Security Symposium, San Francisco, CA, USA, August 8-12, 2011, Proceedings* (2011).

[30] HEATON, J. Comparing dataset characteristics that favor the apriori, eclat or fp-growth frequent itemset mining algorithms. In *SoutheastCon 2016* (March 2016), pp. 1–7.

[31] HENRY, K., STINSON, D. R., AND SUI, J. The effectiveness of receipt-based attacks on ThreeBallot. *IEEE Transactions on Information Forensics and Security 4*, 4 (2009), 699–707.

[32] HOME OFFICE. Operational case for the use of communications data by public authorities. https://www.gov.uk/government/publications/investigatory-powers-bill-overarching-documents, 2016.

[33] INTERCEPTION OF COMMUNICATIONS COMMISSIONER'S OFFICE. Report of the interception of communications commissioner - annual report for 2016, December 2017.

[34] INVESTIGATORY POWERS COMMITIONER'S OFFICE. Annual report of the investigatory powers commissioner 2017, January 2019.

[35] JENSEN, P. B., JENSEN, L. J., AND BRUNAK, S. Mining electronic health records: towards better research applications and clinical care. *Nature Reviews Genetics 13*, 6 (2012), 395.

[36] KUMAR, A., SMITH, B., AND BORGELT, C. Dependence relationships between gene ontology terms based on tigr gene product annotations. In *Proceedings of CompuTerm 2004: 3rd International Workshop on Computational Terminology* (2004).

[37] LAURIE, B., AND KASPER, E. Revocation transparency. *Google Research, September* (2012).

[38] LAURIE, B., LANGLEY, A., AND KASPER, E. Rfc 6962 – Certificate transparency. Tech. rep., 2013.

[39] LEE, J., AND CLIFTON, C. How much is enough? choosing ε for differential privacy. In *International Conference on Information Security* (2011), Springer, pp. 325–340.

[40] LI, N., LI, T., AND VENKATASUBRAMANIAN, S. t-closeness: Privacy beyond k-anonymity and l-diversity. In *Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on* (2007), IEEE, pp. 106–115.

[41] MACHANAVAJJHALA, A., GEHRKE, J., KIFER, D., AND VENKITASUBRAMANIAM, M. l-diversity: Privacy beyond k-anonymity. In *Data Engineering, 2006. ICDE'06. Proceedings of the 22nd International Conference on* (2006), IEEE.

[42] MELARA, M. S., BLANKSTEIN, A., BONNEAU, J., FELTEN, E. W., AND FREEDMAN, M. J. Coniks: Bringing key transparency to end users. In *USENIX Security Symposium* (2015), vol. 2015, pp. 383–398.

[43] MORIARTY, K., KALISKI, B., AND RUSCH, A. Pkcs# 5: Password-based cryptography specification version 2.1.

[44] NAGAR, A., HAHSLER, M., AND AL-MUBAID, H. Association rule mining of gene ontology annotation terms for sgd. In *Computational Intelligence in Bioinformatics and Computational Biology (CIBCB), 2015 IEEE Conference on* (2015), IEEE, pp. 1–7.

[45] NARAYAN, A., FELDMAN, A., PAPADIMITRIOU, A., AND HAEBERLEN, A. Verifiable differential privacy. In *Proceedings of the Tenth European Conference on Computer Systems* (2015), ACM, p. 28.

[46] (NHS), N. H. S. National diabetes audit report, 2017.

[47] PARK, S. H., JANG, S. Y., KIM, H., AND LEE, S. W. An association rule mining-based framework for understanding lifestyle risk behaviors. *PloS one 9*, 2 (2014), e88859.

[48] QUINLAN, J. R. Induction of decision trees. *Machine learning 1*, 1 (1986), 81–106.

[49] QUINLAN, J. R. *C4. 5: programs for machine learning*. Elsevier, 2014.

[50] RIVEST, R. L. The ThreeBallot voting system.

[51] RIVEST, R. L., AND SMITH, W. D. Three voting protocols: ThreeBallot, VAV, and Twin. *USENIX/ACCURATE Electronic Voting Technology (EVT 2007)* (2007).

[52] STRAUSS, C. E. A critical review of the triple ballot voting system, part 2: Cracking the triple ballot encryption. *Unpublished draft, http://cems.browndogs.org/pub/voting/tripletrouble.pdf 74* (2006).

[53] SULEYMAN, M., AND LAURIE, B. Trust, confidence and verifiable data audit, 2017.

[54] SWEENEY, L. k-anonymity: A model for protecting privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems 10*, 05 (2002), 557–570.

[55] TOTI, G., VILALTA, R., LINDNER, P., LEFER, B., MACIAS, C., AND PRICE, D. Analysis of correlation between pediatric asthma exacerbation and exposure to pollutant mixtures with association rule mining. *Artificial intelligence in medicine 74* (2016), 44–52.

[56] VUKOLIĆ, M. Rethinking permissioned blockchains. In *Proceedings of the ACM Workshop on Blockchain, Cryptocurrencies and Contracts* (2017), ACM, pp. 3–7.

[57] YEE, P. Updates to the internet x. 509 public key infrastructure certificate and certificate revocation list (CRL) profile.

[58] ZHANG, N., WANG, S., AND ZHAO, W. A new scheme on privacy preserving association rule mining. In *European Conference on Principles of Data Mining and Knowledge Discovery* (2004), Springer, pp. 484–495.

A MultiBallots

In this appendix, we present an extension of the ThreeBallot scheme to allow for any odd number $n = 2k+1$ of shares, which we call MultiBallot. The goal of this appendix is to provide more details about the security of the scheme as argued in Section 5, following Henry et al. [31]. We follow the terminology used in Section 5 for consistency i.e., a ballot of $2k+1$ shares is generated from a record, rather than the voting related terminology found in most of the work related to ThreeBallot. We also restrict our analysis to the setting of binary elements.

We start off with the case of an adversary that knows several shares of a ballot that tries to reconstruct the record by determining the missing shares. Each element of a share can appear in four forms, $\boxtimes\square$, $\square\boxtimes$, $\boxtimes\boxtimes$ or $\square\square$. For the sake of presentation, we will for now consider only ballots for one element, so a share simply corresponds to an element. It is easy to see that a ballot must contain the same number of $\boxtimes\boxtimes$ and $\square\square$ elements (which we call doubles), and a number of $\square\boxtimes$ and $\boxtimes\square$ elements (which we call singles) that differ by 1. As ballots that count for either value ($\square\boxtimes$ or $\boxtimes\square$) of the element are symmetric since it suffices to flip the single elements, there will be as many $\square\boxtimes$ and $\boxtimes\square$ shares over all possible ballots.

We now determine the form of the possible valid ballots, taking into account the fact that in a valid ballot the element's value in the record must appear $k+1$ times, while its inverse must appear k times. The doubles even themselves out, while the singles make the difference, so there can only be up to $2k$ doubles in a ballot (i.e., k of each), and there must be a least one single to determine the original value of the element. As there can be no doubles, the number of singles then ranges from 1 to $k+1$ for the single representing the original value of the element, and from 0 to k for the other. A simple way of iterating over the possible forms of a ballot is then to take an index $i \in [1, k+1]$ that counts the number of singles corresponding to the original value of the element in the record. There are then $i-1$ of the other single, as well as $k+1-i$ of each double.

To obtain all possible ballots, it is simply necessary to compute the number of permutations for each form a ballot can take. To take into account the repetition of shares in the ballot (e.g., multiple $\square\boxtimes$ shares), we use the standard formula for multiset permutations. Given a ballot with i singles of the original value of the element, the number of permutations of this ballot is as follows.

$$\frac{(2k+1)!}{i!(i-1)!(k+1-i)!(k+1-i)!} \tag{A.1}$$

Computing the total number of possible valid ballots V is then only a matter of summing over all possible forms of ballots i.e., summing over i from 1 to $k+1$, and multiplying the result by a factor of two as elements can take two values. This can also be computed simply as $2\binom{2k+1}{k+1}\binom{2k+1}{k}$, but we proceed using the sum form in the next steps.

$$V = 2\sum_{i=1}^{k+1} \frac{(2k+1)!}{i!(i-1)!(k+1-i)!(k+1-i)!} \tag{A.2}$$

We can now compute the probability distribution of shares by counting the number of times they appear in the valid ballots i.e., in the $(2k+1)\cdot V$ shares of valid ballots. This amounts to again taking the sum of permutations of ballots, but we multiply these by the number of times the share appears in the ballot. As the doubles appear in the same count for ballots corresponding to both values of the element, we multiply

the result by a factor of two to account. For the singles, they appear in different counts depending on the value of the element in the record, so we sum both cases. We denote the number of $\boxtimes\square$, $\square\boxtimes$, $\boxtimes\boxtimes$ or $\square\square$ shares by $V_{\boxtimes\square}$, $V_{\square\boxtimes}$, $V_{\boxtimes\boxtimes}$ or $V_{\square\square}$.

$$V_{\boxtimes\square} = V_{\square\boxtimes} = \sum_{i=1}^{k+1}(2i-1)\frac{(2k+1)!}{i!(i-1)!(k+1-i)!(k+1-i)!} \tag{A.3}$$

$$V_{\boxtimes\boxtimes} = V_{\square\square} = 2\sum_{i=1}^{k+1}(k+1-i)\frac{(2k+1)!}{i!(i-1)!(k+1-i)!(k+1-i)!} \tag{A.4}$$

The probabilities for each share are then as follows.

$$\Pr(\boxtimes\square) = \Pr(\square\boxtimes) = \frac{V_{\boxtimes\square}}{(2k+1)V} \tag{A.5}$$

$$\Pr(\boxtimes\boxtimes) = \Pr(\square\square) = \frac{V_{\boxtimes\boxtimes}}{(2k+1)V} \tag{A.6}$$

With this, we can compute the probability P_{valid} that randomly chosen shares will form a valid ballot by simply summing over the possible ballots, weighted by the probability of each share. In the case where shares contain more than one element, the probability in the case of a number of elements e in the share is simply P^e.

$$P = 2\sum_{i=1}^{k+1}\frac{(2k+1)!}{i!(i-1)!(k+1-i)!(k+1-i)!} \Pr(\boxtimes\square)^{i}\Pr(\square\boxtimes)^{i-1}\Pr(\boxtimes\boxtimes)^{k+1-i}\Pr(\square\square)^{k+1-i} \tag{A.7}$$

Finally, we can compute the probability of an adversary successfully reconstructing a ballot. As the adversary knows up to $a \in [1,2k]$ shares of the ballot, they can attempt to reconstruct the user's ballot from this by choosing $2k+1-a$ other shares from the dataset, which gives $\binom{(2k+1)r-a}{2k+1-a}$ possibilities, where r is the number of records, from which we subtract 1 as there must be at least one valid ballot. Thus for e elements per record, the adversary's chance of success S is given as the probability of finding a valid ballot amongst the possible choices of shares they can make.

$$S = (1-P^e)^{\binom{(2k+1)r-a}{2k+1-a}-1} \tag{A.8}$$

This can easily be computed, and the code we have used to do so will be available upon publication, although dealing with float underflows makes it difficult to compute bounds for higher values of r and e.

B Association Rule Mining

Association rule mining [3] is one of the most commonly used approaches to identify *if-then* rules and relationships between variables in large datasets (e.g., healthcare data). Given an element set $E = \{e_1, e_2, \dots\}$ of binary elements of a record and a dataset $D = \{R_1, R_2, \dots\}$ of records containing elements that form a subset of E, a rule is an implication $\varepsilon \Rightarrow \varepsilon'$ where $\varepsilon, \varepsilon' \subseteq E$. Such association rules are used to find interesting relationships between variables, like linking a set of genes with a particular disease. Two measures are commonly used to select interesting rules: *support* and *confidence*.

Support (defined in equation B.1) indicates how frequently a subset of elements appears in the dataset i.e., the proportion of records $R \in \delta$, where $\delta \subseteq D$, that contain a subset of elements $\varepsilon \in E$.

$$supp(\varepsilon) = \frac{|\{R \in \delta : \varepsilon \in R\}|}{|\delta|} \tag{B.1}$$

The support of a rule $\varepsilon \Rightarrow \varepsilon'$, is simply the support of the joint element sets i.e., $supp(\varepsilon \Rightarrow \varepsilon') = supp(\varepsilon \cup \varepsilon')$. From the above, we can also compute confidence (defined in equation B.2) to indicate how often a rule is found to be true. Given a rule $\varepsilon \Rightarrow \varepsilon'$, as above, it is straightforwardly defined from the support of the rule $\varepsilon \Rightarrow \varepsilon'$ over the support of antecedent ε.

$$conf(\varepsilon \Rightarrow \varepsilon') = \frac{supp(\varepsilon \Rightarrow \varepsilon')}{supp(\varepsilon)} \tag{B.2}$$

Computing these values on D is straightforward but some pre-processing is needed to extract them from D_{priv}, which contains elements $\bar{e}$, so not all of the observed values for ε and ε' match those of the original record.

Our goal is to estimate the true counts of ε, ε' and $\varepsilon \cup \varepsilon'$ in D, based on observations from D_{priv}. Computing the support and confidence measures defined above is then straightforward. This process is often referred to as *support recovery* in the literature. For simplicity, we represent both the original records and the shares as bitstrings e.g., a record with five elements, all being true, will be represented as $[1,1,1,1,1]$ in binary representation, or as 31 in decimal representation. We also define a vector o_D that contains the occurrences of all possible bitstring permutations in D, and a vector o_{priv} with all bitstring occurrences for D_{priv}.

$$o_D,\ o_{priv} = \begin{bmatrix} \#[0] \\ \vdots \\ \#[2^t-1] \end{bmatrix} \tag{B.3}$$

We also compute the expected bitstring occurrences in D_{priv}, denoted $\mathbb{E}(\#bitstring)$, for all possible bitstring permutations and a fixed number of bits (i.e., elements) t, and store these values in a matrix M.

$$M = \begin{bmatrix} \mathbb{E}[\#[0]]_0 & \dots & \mathbb{E}(\#[0])_{2^t-1} \\ \vdots & \vdots & \vdots \\ \mathbb{E}[\#[2^t-1]]_0 & \dots & \mathbb{E}(\#[2^t-1])_{2^t-1} \end{bmatrix} \tag{B.4}$$

We can then estimate o_{priv} from the product of M and o_D by computing o_{priv}.

$$o_{priv} = M \cdot o_D \tag{B.5}$$

In our case, o_{priv} is obtained from D_{priv} and it is o_D that we are interested in. We can solve the previous equation for o_D by inverting[2] M and multiplying it with o_{priv}, giving us the following expression for o_D.

$$o_D \approx M^{-1} \cdot o_{priv} \tag{B.6}$$

Based on the inferred value of o_D, we can now compute the support and confidence measures for any element sets ε, ε'. The accuracy of this method is evaluated in Section 6.

It should be noted that as D_{priv} is used only for verification of the reported statistics and not for mining new associations, this leaves plenty of room for minimizing the information content of D_{priv}. In particular, if D is composed of records with a large number of elements, but only few of these have interesting correlations that are relevant in the published statistics, then only these need to be included in D_{priv}. This can significantly reduce the size of D_{priv} compared to D, especially in cases of datasets sparse in relationships. Moreover, our technique can in certain cases support statistics involving continuous variables. For example, while during the rule mining phase the researcher may need to examine the exact blood pressure values (e.g.., 0–250mm Hg), once a relevant blood pressure threshold is identified, all the values can be expressed as larger or smaller to that threshold (e.g., "blood pressure over 180"). This practice is common in machine learning algorithms. For example, C4.5 (an extension of ID3 [48]) builds decision trees from sets of data samples containing both continuous and discrete attributes. For those that are continuous, it defines a threshold that maximizes the information gain and then splits the samples to those with attibute value lower than the threshold and those with equal or larger value [49]. Alternatively, continuous variables can be discretized and split into multiple binary elements.

[2] M is a square nonsingular matrix as long as its determinant is non-zero. Singular matrices are considered to be rare, and can be made nonsingular with very slight changes that would not affect the results much in our case.