# VAMS: Verifiable Auditing of Access to Confidential Data

Alexander Hicks
University College London
alexander.hicks@ucl.ac.uk

Vasilis Mavroudis
University College London
v.mavroudis@ucl.ac.uk

Mustafa Al-Bassam
University College London
mustafa.al-bassam.16@ucl.ac.uk

Sarah Meiklejohn
University College London
s.meiklejohn@ucl.ac.uk

Steven J. Murdoch
University College London
s.murdoch@ucl.ac.uk

## ABSTRACT

The sharing of personal data has the potential to bring substantial benefits both to individuals and society, but these can be achieved only if people have confidence their data will not be used inappropriately. As more sensitive data is considered for sharing (e.g., communication records and medical records), and as it is increasingly used for making important decisions, there is a growing need for effective ways to hold data processors accountable for their actions, while protecting the privacy of individuals and the integrity of their data. We propose a system, VAMS, that allows individuals to check accesses to their sensitive 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 build two prototype systems, one based on the Hyperledger Fabric distributed ledger and another based on the Trillian verifiable log-backed map, and evaluate their performance on simulated workloads based on real-world data sets. We find that while the one based on Hyperledger Fabric may have more favorable trust assumptions in certain settings, the one based on Trillian is more scalable, achieving up to 102 transactions per second, as opposed to Hyperledger's 40.

## 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 though, 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 because 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 cases such as this, rules cannot be reliably automatically enforced in real-time so the approach commonly taken is to keep detailed records of access attempts and subject some or all of the actions to audit. Provided that the audit is likely to find improper activities and violations are harshly punished, abuse can be effectively deterred. Furthermore, statistics published about the audit can provide confidence to society that access to data is being controlled and organizations who can have access to data will be held to account.

This however raises questions about who performs the audit and how the auditor can be assured that the records they rely upon 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 the personal data and the records containing the justification for data being processed, we cannot allow just anyone to act as an auditor. Even if it was possible to find an organization whose audit would be widely accepted, if the audit is based on tampered records then it would not be reliable.

The discussion so far has focused on the confidentiality of personal data, but its integrity is also important. When actions are taken on the basis of this data, whether making a medical treatment decision or conducting legal proceedings, relying on tampered data may lead to severe consequences. In some cases 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 protecting 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

To further motivate the design for our system, we consider two challenging scenarios: controlling the access of law-enforcement personnel to communication records and controlling the access of healthcare professionals to medical data.

#### 1.1.1 *Law-enforcement access to communications data.*

In the UK 95% of serious and organized criminal cases make use of communications data [42] – metadata stored by telecommunications providers in their billing system about account holders or their use of communications networks (such as phone numbers called, address associated with an account, or location of a mobile phone). Telecommunications providers are required to store this data for a period of time (up to 2 years), but once this period has expired and there is no business reason to store this personal data, they are required to delete it. Within the period that data is stored, law-enforcement personnel are permitted to request access, provided that they can demonstrate that their actions are legally justified[1]. At the time such a request is made, there is, however, no external oversight. Instead, information about the request and the justification for access are stored and made available for audit by the Investigatory Powers Commissioner's Office (IPCO)[2]. IPCO then assess whether law enforcement make appropriate use of the powers they were given, and also publish reports with statistics of how these powers were used [43].

Communications data plays an important role in the investigation of criminal offenses, but may also be used as evidence in legal proceedings, whether for the purposes of prosecution or defense. Should any question be raised about the integrity of the communications data evidence, a senior representative of the telecommunications provider will be asked to appear in court and asked to verify the evidence against company records and attest to its accuracy. If technical issues arise related to this evidence, one of the parties to the case may also request that the court request specialist assistance from an expert witness. This process of verification is expensive and time consuming, and even impossible if the provider has deleted the original data in between the law enforcement agency requesting and it being required in court.

In order to improve the process, industry standards allow for providers to sign or hash communications data when it is provided in response to a request from law enforcement. Someone who needs to verify an item of data can compare the hash to the one stored by the provider, or verify the digital signature using the provider's public key [30]. However, if the provider's private key or hash database is compromised, any evidence presented subsequent to the compromise will be brought into doubt, even if the evidence was generated before the time of compromise.

Our system can be applied in this scenario, allowing the integrity of communications data evidence to be demonstrated, even if the communications provider which produced the data no longer exists or has been compromised. Furthermore, the system will give assurance to the auditor that records of requests to access communications data have not been tampered with, and assure society that reported statistics have not been improperly manipulated by the auditor. We also show how the system protects the privacy of individuals whose data is requested, and also protects the confidentiality of ongoing law-enforcement investigations.

[1]Similar legal powers are available in the US through the use of administrative subpoenas, but as there are no publicly available statistics for their use and there is no centralized oversight, we focus on the UK case.

[2]Prior to September 2017 this role of IPCO was the responsibility of the Interception of Communications Commissioner's Office (IOCCO).

#### 1.1.2 *Access to healthcare records.*

In our second scenario we consider how to empower individuals by giving them control over how their medical records are used and shared.

In a healthcare system, once consent has been given by a patient, various actors should be able to access various records associated with that patient; e.g., their general practitioner should be able to access scans that were run at a hospital, and researchers running academic studies or clinical trials in which the patient has enrolled should be able to access records relevant to the study. Currently patients can only give permission for broad types of activities and so they may legitimately have concerns that their information is being used inappropriately. Conversely, patients with serious diseases (cancer, motor neuron disease, etc.) often have trouble getting the treatment they need, as universities conducting academic studies are legally 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 would also open up the potential for abuse, so it is important for patients to have visibility into how their data is being used in order to understand the implications of their consent. For clinical practice, the default could be that patients opt in to sharing their data, although they can always opt 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 (for example according to location or type of study) to which studies they want to opt in.

One issue with having patients opt in individually is that, for some studies, this process may simply not result in a large enough sample. Equally, if patients are deluged with requests for consent, they 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.

We will show how 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 the first system, VAMS, to provide a range of auditability, privacy and verifiability guarantees across the whole timeline of requesting access to data, auditing such requests and verifying the statistics produced by auditors.

In more detail, VAMS uses append-only logs of data access requests, which are instantiated as either blockchain-based distributed ledgers or verifiable log-backed maps. These allow users to examine the log in order to discover requests relevant to them, while auditors can do so in order to detect any misuse or errors in the requests. The integrity properties of the log mean that logged requests may be used as evidence. Requests on the log are unlinkable to each other and to users, and a scheme based on ThreeBallot and association rule learning makes it possible to compute publicly verifiable statistics from the log data without revealing any information but the statistics.

Two sample implementations using Hyperledger Fabric and Trillian are presented, with an evaluation of performance and security trade-offs. We find that, while Hyperledger Fabric provides more flexible policies and offers a wider variety of trust assumptions, Trillian is more scalable and a system based on it would be much easier to deploy today. We also evaluate our privacy scheme, measuring accuracy and privacy loss in different scenarios.

## 2 RELATED WORK AND BACKGROUND

This section examines existing prior work in the area and discusses how this paper compares to it. We also introduce the building blocks of our system: Hyperledger Fabric, Trillian, and the ThreeBallot voting scheme.

### 2.1 Verifiable computation

Verifiable computation schemes aim to allow clients to verify the correctness of an outsourced computation. Many existing verifiable computation schemes for general classes of programs rely on advanced cryptographic techniques, such as fully homomorphic encryption or succinct zero-knowledge proofs [9, 14, 21, 25, 33, 35, 62, 78]. These protocols require a logarithmic (or higher) number of messages to be exchanged between the prover and verifier [17, 20, 65, 66, 76], while alternatives built on interactive protocols assume the existence of random oracles. As a result, using these schemes comes at a significant cost [79], which we consider to be too high for our proposed system and use cases.

### 2.2 Data anonymization

It is often the case that datasets containing sensitive information about individuals are useful in various applications, such as research, decision-making and advertising. However, due to the sensitive nature of the information they contain, datasets must be stripped of any information that identifies individuals before being released publicly.

A straightforward approach is data pseudonymization, where all user identifiers (e.g., names) are replaced by random identifiers. However, as was found in the case of the Netflix Prize [58], an adversary will often leverage side information to de-anonymize individuals and thus circumvent any pseudonymization.

To address this problem, a wealth of anonymization techniques have been introduced, the most popular of which are k-anonymity [72] and its extensions l-diversity [53] and t-closeness [54]. Unfortunately, k-anonymity and l-diversity have been shown to be vulnerable to various attacks, while t-closeness ends up leaving very little useful information in the dataset.

More recently, differential privacy was introduced by Dwork et al. [28] to provide a notion of privacy related to outputs of algorithms on datasets. This approach has seen more success and has been applied to many use cases [27] but is still not widely used in practice. In particular there are trade-offs to consider between privacy and utility [4], as well as one-shot and continuous observation [29]. As a result, obtaining a meaningful privacy parameter [52] is rarely achieved in practice [73].

### 2.3 Privacy-preserving and verifiable statistics

Whilst both the problem of privacy-preserving statistics [15, 18, 44, 68] and of verifiable computation [21, 48, 75] have been extensively studied, few papers provide a single scheme fulfilling both requirements. However, in many cases untrusted third parties must be able to verify the correctness of publicly released statistics, which is often in contradiction with privacy requirements. Unfortunately, all existing solutions come with trade-offs between security, privacy, flexibility and efficiency that make them hard to use in practice [26]. Most of those schemes are typically based on differential privacy or homomorphic encryption [35, 41, 47, 57] and are either very limited, or come with an unrealistic computational overhead.

In this paper, we focus on non-interactive privacy-preserving schemes that achieve *output privacy* (i.e., the verifier checking the correctness of the computations does not learn anything about the input), which is achieved by very few papers in the literature. Boyle et al. [13] present constructions including succinct function private functional signatures, and Barbosa et al. [7] present a new delegatable homomorphic encryption primitive. In both cases, they rely on advanced cryptographic techniques (SNARKs, fully homomorphic encryption, and functional encryption), and do not include implementations. This renders them difficult to deploy, in contrast to our lightweight scheme.

While those generic schemes enable simple arithmetic operations between variables, extracting association rules in a privacy preserving manner is a complex and different task [2]. One of the very first techniques used for privacy-preserving association rule mining was *uniform randomization*, where individual user records are uniformly randomized based on a public factor. The aims is to transform the dataset so as to conceal the information of individuals, while preserving the associations themselves. However, as pointed out by Evfimievski et al. [31], this does not provide adequate privacy, and it is easy for an adversary to recover several of the original records. In the same work, Evfimievski et al. proposed a class of randomization operators that achieve much better privacy. However, even these operators achieve unrealistically low privacy [82], while they require datasets of at least one million records. Zhang et al. [82] proposed a new scheme that is not item-invariant and considers the existing association rules when perturbing each transaction. This scheme provides much better privacy bounds compared to previous works, but also distorts the strength of the association rules, overestimating strong relationships and under representing less frequent ones. This makes the technique unsuitable for our needs, as the added noise would it impossible for individual users to verify the accuracy of their own records, as outlined in Section ??.

### 2.4 Tamper evident logging

Tamper evident logging has previously featured in work by Crosby and Wallach [22] and Bates et al. [8].

Crosby and Wallach consider the case of a untrusted logger serving clients storing events in the log, that is kept honest through auditing. Similar to us, they rely on a hash-tree based log, but assume a single centralized log and do not address secrecy of logged events or replication.

Bates et al. are concerned with accountable logs of wiretapping, which is a different use case to ours. They discuss interception data which would be inadmissible as evidence in some legal systems (including the UK) and is subject to judicial oversight prior to authorization of requests. We focus on retained communication data, which is more internationally applicable and is subject to oversight only after the fact.

## 2.5 Hyperledger Fabric

Hyperledger Fabric (HLF) [5] describes itself as a modular, extensible open-source system for deploying and operating permissioned blockchains and includes architectural differences to most existing solutions [16, 77].

A HLF network is a network of peers (and an ordering service), with identities assured by a *Membership Service Provider* PKI, which maintain a key-value store as the state of the shared ledger. The state can be updated and queried through transactions on the underlying blockchain, where by transactions we simply mean *chaincode* (smart contract) executions.

As peers have identities, they can be split up into organizations, as well as roles on the network with regards to transactions. To execute a transaction, an *endorsing peer* (or many) executes the deterministic chaincode inside a docker container and signs the transactions 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 makes it possible to define endorsement policies, which limit which peers can invoke certain chaincode, and which peers must sign transactions for a given chaincode.

## 2.6 Trillian

Trillian [36] is an open-source project that implements a generalized version of Certificate Transparency [51], based on two data structures [1]: a verifiable log backed by a verifiable map.

The verifiable log is an append-only log implemented as a Merkle tree, as described in Certificate Transparency. It 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) [50]. 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.

**Table 1: Functionalities of the parties in the system.**

| Function | Description |
|---|---|
| *request* | Place and record a request about a user |
| *provide* | Provide data to an agent as required by a request |
| *check* | Check the log for requests tied to a user identifier |
| *monitor* | Verify the statistics published by auditors |
| *detect* | Detect if a log server is misbehaving |
| *audit* | Audit and publish the log of requests |
| *host* | Host a log of requests submitted by agents |
| *broker* | Act as an intermediary for users |

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

## 2.7 ThreeBallot voting system

ThreeBallot [63, 64] is a paper-based voting scheme proposed by Rivest for end-to-end auditable elections.

Voters are given three ballots arranged as three columns, with each row corresponding to a candidate. Votes are cast according to simple rules. Each row must be selected at least once and no row must be selected thrice, selecting a row in two columns is a vote for and selecting a row in one column is a vote against. The outcome is the same as that of a standard election while voters use receipts to check their vote was counted (assuming a public bulletin board) and ballots are unlinkable.

The scheme's security has been extensively studied in various works [6, 19, 40, 46, 64, 70, 71]. From all the attacks examined, the reconstruction and pattern-based attacks are applicable to our use cases. These attacks have been examined by Henry et al. [40] for two candidate races (i.e., binary choices), and extend on work by Strauss [71]. These works provide a lower bound for security as a function of the ballot size (number of binary choices), and we later use them to determine the secure usage boundaries for our system.

## 3 SETTING AND THREAT MODEL

### 3.1 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, who are not active in the use of the system but serve to define regulations, such as the Investigatory Powers Act of 2016, which determine the rules obeyed by the parties in the system.

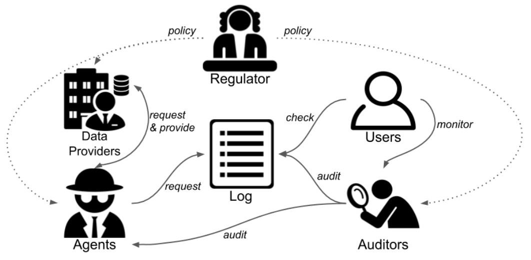


**Figure 1: All essential parties in our setting and their functionalities, along with the regulator and their policy that are external to our setting. The optional data broker would act as a user.**

Each party in our system and the functions they perform (detailed in Table 1) are defined as follows:

*Agents*, the set of which we denote A, could be public authorities, companies or generally any party wishing to obtain user data from data providers. For an agent $a \in$ A, we denote this functionality *a.request*.

*Data providers*, the set of which we denote DP, could be telecommunication companies, healthcare providers or generally any party collecting user data They are responsible for receiving and answering data requests from agents. For a data provider $dp \in$ DP, we denote this functionality *dp.provide*.

*Log servers*, the set of which we denote S, are responsible for providing access to the log of requests made by agents. For a log server $s \in$ S, we denote this functionality *s.host*.

*Auditors*, the set of which we denote O, are organizations such as the IPCO, which audit requests made by agents to check for errors and publish statistical reports. They must also be able to detect if log servers are behaving dishonestly. For an auditor $o \in$ O, we denote these functionalities *o.audit* and *o.detect*, respectively.

*Users*, the set of which we denote U, are members of the public. If a user is generating data(e.g., using the Internet or participating in the healthcare system), they may wish to check the requests that have been made about them. We denote this functionality as *u.check*. Additionally, any user may wish to check if the log server is misbehaving, or that the reports published by the auditor are correct. We denote these respective functionalities by *u.detect* and *u.monitor*.

*Data brokers*, the set of which we denote B, are non-essential intermediaries which users can rely on to deal with data requests if they are willing to serve as a data provider, for example providing data to an agent running a study. The data broker can then be chosen to deal with these requests according to pre-set rules from the user, for example on which type of study they are willing to participate in. For a data broker $b \in$ B, we denote this functionality *b.broker*.

For the above parties, we say that they are honest-but-curious if they attempt to gain information that is not inherently visible to them, for example by inferring information from reports published by auditors or by linking requests that they are not involved in. Malicious parties aim to trick the system by dishonestly performing their functionalities. This means agents submitting invalid requests, data providers providing invalid data, log servers hosting erroneous logs, auditors publishing inaccurate reports or users checking requests of other users. In general, we always assume a computationally bounded adversary that has access to all released data, statistics and logged requests. Moreover, the adversary may also have full or partial information about the records of users.

## 3.2 Threat model

With the above notation and setting laid out, we now define our threat model where we address three criteria: auditability, privacy and verifiability. In essence, auditability is about ensuring the integrity of the information on the system, privacy is about ensuring the only information that can be gained by anyone is already in the clear, and verifiability is about ensuring that information the auditor publishes about hidden data is accurate.

We begin with log servers. These may be malicious, but our system allows users and auditors to detect this (although not prevent it). In what follows, we thus assume log servers are honest, as they can otherwise be removed from the system for misbehaving. We also assume that malicious agents and data providers do not collude, as they could then simply choose to not use the system.

Starting with auditability, we split this into two cases. For an auditor, we allow all other parties to be malicious, but require that the auditor can still run *o.audit* properly. For a user, we allow everyone but data providers to be malicious, as the identifier tied to the request must be correct in order for them to be able to run *u.check*. We again require that the user must be able to properly run this function, as well as *u.monitor*.

In terms of privacy, the goal is to ensure that no information that is not already public can be gained by any malicious party not involved in the request. We thus allow any malicious party to attempt to link requests together in order to gain information about the agents, data providers or users linked to a request. We also allow malicious parties to attempt to gain more information than is revealed by the statistics themselves when they are published by an auditor. The goal in both cases is to prevent them from learning this information. We can model this as a game, in which an adversary is given a transformed dataset containing private information about a user and, in the clear, all but one of the database fields about the user. To win, the adversary must guess the last field with noticeably higher advantage than if they had not seen the database fields (so just had the transformed dataset).

Finally, for verifiability, we require that auditors maliciously performing *o.audit* will get caught by users, who check the validity of the statistics by performing *u.monitor*.

## 4 OUR SYSTEM: VAMS

With our setting and threat model in mind, we now present an overview of the form our system will take, before specifying the proposed mechanisms that provide our security guarantees. In particular, as the separate functionalities offered by our system already exist separately in various forms (discussed in Section 2), we justify our design choices which allow us to combine everything into one cohesive ensemble.

## 4.1 Overview of system requirements

Looking at the functionalities (Table 1) that the parties in the system must be able to perform and our threat model, we can already lay out a basic framework which supports the required operations without placing trust in other potentially malicious parties.

Clearly, the system should contain some form of database of requests submitted by agents through *a.request*, which is maintained by log servers performing *s.host*. As auditors and users should be able to detect malicious log servers through *u.detect* and *o.detect*, the database structure 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.

To store requests, key-value stores are a natural choice as requests are tied to unique identifiers that form a set of keys to which we assign request values. Retrieving requests is then made simple for auditors and users performing *o.audit* and *u.check* respectively, as the key value map can easily be queried for identifiers, or a range of identifiers. Performing these functions with integrity also involves being able to check that the retrieved requests are the original ones submitted by agents, and have not been tampered. This reinforces the need for a data structure that is append only.

The need for an append only data structure can also be seen in cases where evidence is required. Examples of this (introduced in Section 1.1) are court cases where law enforcement or a healthcare companies 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, for example 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 recorded in the log, and the auditor has performed their audit, they will publish the resulting statistics through *o.publish.* Users must be able to verify these statistics through *u.monitor*, there must be evidence of the results they publish, as well as the data necessary to verify their results, without compromising privacy.

To achieve the requirements above, a data structure such as a blockchain or a form of Merkle tree is required. We make the choice of using existing solutions: verifiable log-backed maps (Trillian) and blockchain based permissioned ledgers (Hyperledger Fabric), which were introduced in Sections 2.5 and 2.6. Both store key-value maps, either as the state of the ledger which is updated by transactions on the underlying blockchain, or as a sparse Merkle tree, where each branch leads to a key. These also support our need for verifiability of audits, as the results of an audit can be included in the system with the same robustness guarantees as any other data i.e., the requests, which allows users performing *u.monitor* to verify the results without having to trust they have been altered by the auditor that publishing them. This is in line with the use case for permissioned blockchains outlined by Wüst and Gervais [80], as a state must be stored with multiple known and untrusted writers, no always-online trusted third party, and verifiability requirements.

## 4.2 Mechanisms to build VAMS

Now that we have settled on a set of design choices, we move on to the specific mechanisms we provide and argue that they achieve the security guarantees from our threat model. Two implementations, using HLF and Trillian, of the design presented here are provided in Section 5.

For generality, we abstract both HLF and Trillian as the underlying key-value store underlying the system.

*4.2.1 Publishing requests to the log.* When performing *a.request*, agents append requests to the log by assigning a request to a key in the key-value store. To make sure requests are tied to unique identifiers, each key corresponds to a different request using *common identifiers*, which are built from existing private identifiers. We assume that agents and data providers refer to users by private identifiers $id_a$ and $id_{dp}$, that are also known to the user, and simply encrypts one using the other as key with a secure encryption scheme such as AES. By maintaining a *session identifier n*, an integer that changes deterministically (e.g., increases by one) with each request involving a pair ($id_a$, $id_{dp}$), the pair can be re-used by the agent and data provider to generate new common identifiers (which we denote $id_c$) by concatenating the session identifier to the encrypted identifier; i.e., they can compute $id_c = Enc(id_a, id_{dp} \| n)$. As $id_a$ and $id_{dp}$ may be short and have little entropy, a key derivation function (KDF) such as PBKDF2 [55] should be used to obtain a more resilient ciphertext, with $id_a$ being fed through the KDF before being used as an encryption key.

The unlinkability that results from this usage of identifiers is argued in Theorem 4.3.

In order to provide basic access control to the information on the log (such that information cannot be inferred from a live view), requests are then encrypted under the public key of auditors and the relevant user, so that only they may decrypt the requests they should have access to.

*4.2.2 Checking requests in the log.* Once requests are made, auditors and users can check the log using *o.audit* and *u.check*, assuming they have first determined the log servers to not be malicious through *o.detect* and *u.detect*, which we argue they can in Theorem 4.1. Accessing the key-value map, they can then rely on the integrity properties of the log, as argued in Theorem 4.2, to audit the requests made. To find their own requests, the user has to iterate over possible values of the session identifier *n* until no request is found, to determine the possible requests relevant to them.

If users do not wish to take on this computational burden, they may optionally choose to outsource this role to a data broker. These parties act as intermediaries between agents and users that would otherwise perform *u.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

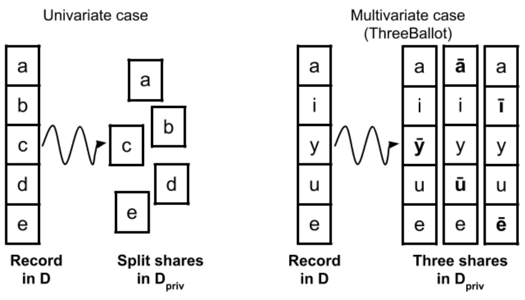

Univariate case
Multivariate case (ThreeBallot)
a
b
c
d
e
i
y
u
ā
ī
ȳ
ū
ē
Record in D
Split shares in $D_{priv}$
Three shares in $D_{priv}$

**Figure 2: Constructing shares in $D_{priv}$ from a record in $D$. Left illustrates splitting the elements of the record into individual shares for univariate statistics, right illustrates generating three shares from a record according to the ThreeBallot scheme.**

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.

*4.2.3 Statistics on the logs.* In our motivational use cases, and many others, auditors may be required to publish statistics obtained from data accessed when performing *o.publish*. An example of what might be published are the statistics provided by the IPCO in its annual report [43], or results of a study in the healthcare scenario. For operational and privacy reasons, however, the original data used to compute statistics cannot be published. This means that users whose data was used, and any other user in general cannot verify the correctness of these statistics and instead must trust the auditors.

Instead, auditors can publish their results (i.e., statistics) and a transformation of the data used. From the transformed data, users can verify the correctness of the results when performing *u.monitor* (as argued in Theorem 4.5). To ensure statistics reveal no more than the statistics themselves (as argued in Theorem 4.4) we use two schemes that enable a range of lightweight privacy preserving and verifiable statistics. The transformation operates on the original dataset $D$ of $n$ records, where each record $R_{i \in [1,n]}$ is comprised of multiple elements. For instance, an element may note the existence (or absence) of a particular gene or mutation, while another one may report on a particular phenotype.

The simplest case is when only univariate statistics are required. In this case, the transformation generates a privacy-preserving dataset $D_{priv}$ by splitting each record into shares, one for each element of the record as in Figure 2. This prevents any information leakage from correlations between the shares, or inference. Each of those shares 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 analysis results (statistics). 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. In the case of Hyperledger Fabric, chaincode may be used to allow users to publicly report on the results of the verification of their records, and thus collectively confirm (or not) the integrity of the published data.

If a more complex analysis is needed (i.e., multivariate statistics), a scheme analogous to the ThreeBallot scheme (presented in Section 2.7) can be used to generate $D_{priv}$. In this case, each record is split into three *shares*, which are comprised of as many elements as the original record. For each element $e$ of the record, two of those shares (randomly selected) are set to $e$, while the remaining one is set to $\overline{e}$, the false value for $e$. Figure 2 illustrates this. As argued in Theorems 4.4 and 4.5, this process both prevents an adversary from breaching the privacy of individuals and preserves the correlation between the elements. Given $D_{priv}$, it is then possible to compute multivariate statistics for the original dataset $D$, with only a small error. As in the univariate case, auditors can then publish $D_{priv}$, so that users can verify that their shares are accurately represented, and statistics can be re-computed.

In more detail, we use ideas from *association rule learning* (ARL) [2], one of the most commonly used techniques for multivariate analysis of datasets. Given an element set $E = \{e_1, e_2, \dots\}$ of binary attributes (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 $\epsilon \Rightarrow \epsilon'$ where $\epsilon, \epsilon' \subseteq E$. 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 4.1) 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 $\epsilon \in E$.

$$supp(\epsilon) = \frac{|\{R \in \delta : \epsilon \in R\}|}{|\delta|} \tag{4.1}$$

The support of a rule $\epsilon \Rightarrow \epsilon'$, is simply the support of the joint element sets i.e., $supp(\epsilon \Rightarrow \epsilon') = supp(\epsilon \cup \epsilon')$. From the above, we can also compute confidence (defined in equation 4.2) to indicate how often a rule is found to be true. Given a rule $\epsilon \Rightarrow \epsilon'$, as above, it is straightforwardly defined from the support of the rule $\epsilon \Rightarrow \epsilon'$ over the support of antecedent $\epsilon$.

$$conf(\epsilon \Rightarrow \epsilon') = \frac{supp(\epsilon \Rightarrow \epsilon')}{supp(\epsilon)} \tag{4.2}$$

Computing these values on $D$ is straightforward, but some pre-processing is needed to extract them from $D_{priv}$. As elements $\overline{e}$ appear in the shares generated to construct $D_{priv}$ from $D$, not all of the observed values for $\epsilon$ and $\epsilon'$ match those of the original record.

Our goal is to estimate the true counts of $\epsilon$, $\epsilon'$ and $\epsilon \cup \epsilon'$ 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 $D_{priv}$.

$$\boldsymbol{o}_D,\ \boldsymbol{o}_{priv} = \begin{bmatrix} \#[0] \\ \vdots \\ \#[2^t - 1] \end{bmatrix} \tag{4.3}$$

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 $M$.

$$M = \begin{bmatrix} \mathbb{E}[\#[0]]_0 & \dots & \mathbb{E}(\#[0])_{2^t-1} \\ \vdots & \vdots & \vdots \\ \mathbb{E}[\#[2^t - 1]]_0 & \dots & \mathbb{E}(\#[2^t - 1])_{2^t-1} \end{bmatrix} \tag{4.4}$$

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

$$\boldsymbol{o}_{priv} = M \cdot \boldsymbol{o}_D \tag{4.5}$$

In our case, $\boldsymbol{o}_{priv}$ is known as we can simply count the occurrences in $D_{priv}$ and it is $o_D$ that we are interested in. We can solve 4.5 for $o_D$ by inverting[3] $M$ and multiplying it with $o_{priv}$.

$$\boldsymbol{o}_D \approx M^{-1} \cdot o_{priv} \tag{4.6}$$

Based on the computed $o_{priv}$, we can now compute the support and confidence measures for any element sets $\epsilon, \epsilon'$. The accuracy of this method is evaluated in Section 5.

## 4.3 Security arguments

We now argue the security of our system in the adversarial model described in Section 3, based on the mechanisms proposed above. For Theorems 4.1 and 4.2, the arguments are split between the HLF and Trillian based systems.

Theorem 4.1. *An auditor $o \in \mathrm{O}$ and a user $u \in \mathrm{U}$ can detect malicious log behavior (i.e., equivocation) when performing o.detect and u.detect, respectively.*

Proof. In the HLF case, there are two options. Either the ordering service maintains consensus, and the log cannot equivocate, or there is a fork of the blockchain. In the event of a fork, both the main chain and the alternative chain are visible, so equivocation can be detected.

In the Trillian case, a log 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 will no longer be able to accept the same Merkle consistency proofs to extend the logs because the leafs will be different. As the tree heads are signed by equivocating log server, it will be detected. □

Theorem 4.2. *Assuming an honest data provider $dp \in \mathrm{DP}$, an auditor $o \in \mathrm{O}$ and a user $u \in \mathrm{U}$ can detect malicious agent behavior (i.e., invalid requests) when performing o.audit and u.check, respectively.*

[3] $M$ 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.

Proof. In both the HLF and Trillian case, auditors and users with access to the system can perform $o.audit$ and $u.check$ by querying the state of the ledger or log-backed map containing the requests, which are encrypted under their public keys. Both can then verify that requests are valid, and detect any invalid request. A request that cannot be decrypted, either under the auditor's public key or under the user's public key for a relevant common identifier, can also be considered invalid and reported.

In the event that a malicious party attempts to tamper requests appearing on the log (i.e., modify the value of a key), auditors and users can then rely on properties of HLF and Trillian.

In the HLF case, we rely on the integrity properties of the underlying blockchain that records the requests that update the ledger's state. Auditors and users 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 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. □

Theorem 4.3. *Assuming that AES is a secure pseudorandom permutation, and that session identifiers n are used only once for each pair of (pseudorandom) private identifiers $id_a$ and $id_{dp}$, it is not possible to link two or more requests (i.e., common identifiers) that appear in the log.*

Proof. A user $u$ or adversary knowing both $id_a$ and $id_{dp}$ will be able to find requests about $u$ with probability 1 by iterating over possible values of $n$ and computing common identifiers.

An adversary knowing one of $id_a$ and $id_{dp}$, determining the other input (say $id_a$ if the adversary knows $id_{dp}$) would require iterating over possible values of $id_a$ as well as $n$, which would require $O(|id_a| range(n))$ AES encryptions, where $|id_a|$ is the size of $id_a$ and $range(n)$ is the range of values $n$ takes. This would only reveal one pair of private identifiers for a user. With regards to agent-data provider unlinkability, each pairing is equally as likely, which gives probability $\frac{1}{|\mathrm{A}||\mathrm{DP}|}$ of determining one.

If an adversary knows neither $id_a$ or $id_{dp}$, then the security of AES is enough to argue that the adversary will not be able to determine two common identifiers shared one or more inputs. □

Theorem 4.4. *Knowledge of all but one element of the record belonging to a user $u \in \mathrm{U}$ does not make it possible to learn the last one from the statistics and private dataset $D_{priv}$ published by an auditor $o \in \mathrm{O}$ as part of o.publish.*

Proof. Shares are not manually generated, so a user cannot be coerced into selecting specific shares (as in the ThreePattern attack), but the risk of producing shares that would be identifiable (i.e., permitting reconstruction attacks) remains. An adversary knowing all but one elements of the original record could compute the possible share arrangements for these, looks for them in $D_{priv}$ and use these to infer the last element unknown to them. To prevent this, even the least probable patterns of shares must appear more than once in $D_{priv}$.

Henry et al. [40] focus on the reconstruction and ThreePattern attack in ballots involving multiple two-candidate races i.e., ballots with binary choice. This matches our use cases as records would involve elements of the form "has gene $X$: true or false", that are binary. In cases where elements take more than two possible values, they can be split into multiple binary elements. For example, "has preference $X$, $Y$ or $Z$" can be split into three binary elements of the form "has preference $X$" where only one takes value true and the other two take value false.

Henry et al. present upper bounds on the safe number of elements in a record for 100, 1000 and 10 000 users i.e., a user with shares arranged in the least likely pattern will find that it appears at least one less time than the upper bound with probability almost 1. For reconstruction attacks, the upper bounds are 7, 11 and 15 elements. These are noticeably higher than the upper bounds in the case of coercion i.e., if shares were manually generated. If repeated patterns exist due to coercion, the upper bounds are 2, 4 and 6 elements for a single pattern, and 2, 3 and 5 elements for five patterns. Strauss studied various settings with correlated races and proved that even heavily correlated races had a minor effect on the security of the scheme [71]. Based on these findings we reduce the aforementioned number of traits by one to account for extreme correlation cases, which leaves us with upper bounds 6, 10 and 14 elements per record for 100, 1000 and 10 000 users, respectively.

Finally, as $D_{priv}$ is used only for verification, this leaves us plenty of room for minimizing the amount of information included. 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 could be published in $D_{priv}$. This can significantly reduce the size of $D_{priv}$ compared to $D$, especially in cases of datasets sparse in relationships. Moreover, the fact the $D_{priv}$ is used only for verification and not for association mining allows us also to support even continues variables by turning them into binary. For example, while during the rule mining phase the researcher will consider all the blood pressure values, once a relevant blood pressure threshold is identified, all the values can be expressed as larger or smaller than that e.g., "blood pressure > 180". ◻

Theorem 4.5. *A user $u \in U$ can verify the correctness of the statistics published by auditors when performing $u.monitor$.*

Proof. A user that was included in the used dataset $D$ used can check the integrity of the transformed dataset $D_{priv}$, identifying their shares in $D_{priv}$ 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. ◻

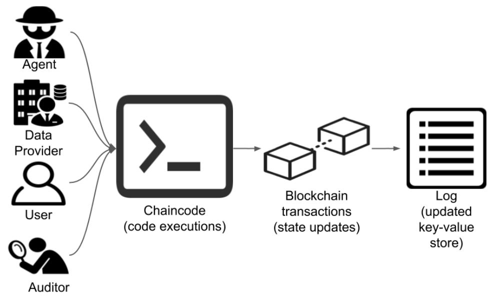

Agent
Data Provider
User
Auditor
Chaincode (code executions)
Blockchain transactions (state updates)
Log (updated key-value store)

**Figure 3: The Hyperledger Fabric based implementation.**

## 5 IMPLEMENTATION AND PERFORMANCE

Here we present two implementations: one based on Hyperledger Fabric and one based on Trillian. Both are evaluated on identical Amazon AWS t2.medium instances.[4] The ThreeBallot-based privacy scheme is also evaluated. The code for each of these will be open-sourced after publication.

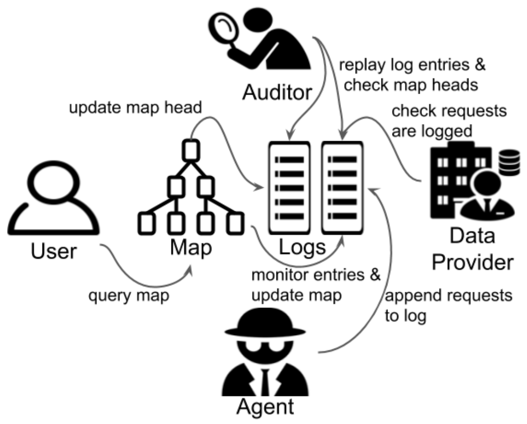
### 5.1 Hyperledger Fabric

The modularity of Hyperledger Fabric (HLF, introduced in Section 2.5) makes it a good candidate for our use case. The network maintains a key-value store which can be populated by requests linked to common identifiers. These requests can then easily be retrieved, querying specific keys or a range of keys in lexical order. As updates to the state happen as transactions on the underlying blockchain, the expected verifiability guarantees for state updates are present. There is also key history function that returns the updates to a key's value, along with the blockchain transactions which resulted in the update rather than simply the state of the database. Furthermore, as the HLF project has ongoing development by IBM, improvements in scalability, privacy and integrity can be expected. In particular, private channels exist, but limited cross-channel support prevents them from being used in our case. Specific improvement proposals for encrypted transactions and state values have also been discussed and are being developed for future releases, as well as attribute based access control.

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[5] and a Kafka broker), and a client from which commands are sent to peers.

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 *a.request*), the second is used to retrieve a range of key values (as part of *o.audit*), the third is used to retrieve values for specific keys (as part of *u.check*) and the fourth is used to retrieve a key's history (as part of *o.audit* and *u.check*) to see which blockchain transaction resulted in state updates for a given key. Chaincode invocations result in state updates recorded in blocks on the blockchain, that

[4]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.
[5]Ideally, a Byzantine fault tolerant ordering service would be used. Although one has been proposed for HLF [69], it is not yet available.



Auditor
replay log entries & check map heads
update map head
check requests are logged
User
Map
Logs
Data Provider
query map
monitor entries & update map
append requests to log
Agent

**Figure 4: The Trillian based implementation.**

then appear in the log, which is the state of the ledger i.e., the key-value store.

Peers have identities, X.509 [81] public key certificates, and sign transactions accordingly. 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 more generally require multiple signatures, they could also hold multiple parties accountable. For example, 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 state updates. 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.

### 5.2 Trillian

The second implementation of the system is based on Trillian, using a verifiable log-backed map (introduced in Section 2.6) as its underlying data structure. Figure 4 summarizes the system.

As part of *a.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 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 *u.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 *o.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.

**Table 2: 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 | ● | ◐ |

### 5.3 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.

On the other hand, HLF supports more flexible chaincode policies for governing write access to the log, as it comes with built-in authentication and public key infrastructure know 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 2 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: Micro-benchmarks of basic operations for the Hyperledger Fabric and Trillian based implementations. The max 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 |

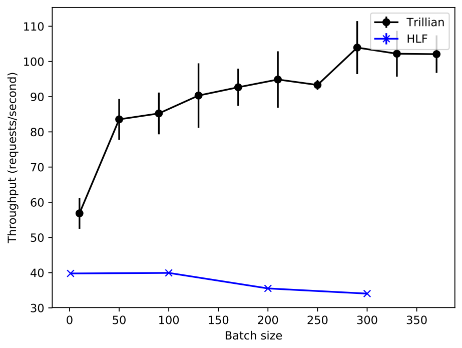


**Figure 5: Throughput evaluation for the HLF and Trillian based implementations.**

## 5.4 Performance measurements

*Micro-benchmarks.* Table 3 presents micro-benchmarks for the basic operations of our systems. These include state updates (i.e., adding a request as part of *a.request*), state retrievals (i.e., retrieving requests as part of performing *u.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 10*ms*. 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 3 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 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 however. 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 [43], 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 [30]) 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.

*ThreeBallot scheme.* To evaluate the effectiveness and applicability of our verifiable statistics schemes, 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 [39], then mine those itemsets using the Apriori algorithm [3]. The algorithm 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 $D$, as described in Section **??**. 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 [43], and the healthcare use case e.g., proportions of people registered with diabetes that achieved blood glucose, pressure and cholesterol targets [60]. 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 [24] and the T10I4D100K dataset [34]. In all our experiments, we measured the error for both the support and the confidence metrics. However, we include only the graphs for support, as those for confidence are identical.

In our first experiment, we study the percent error for the support of element sets with varying occurrence frequencies based on a dataset of 1 million transactions/users. For each of those settings, we repeat the experiment 100 times (the same process is followed in the other two experiments). As seen in Figure 6, element sets that occur less often are prone to higher percent error, with a high variance in the reported support values. However, as element sets

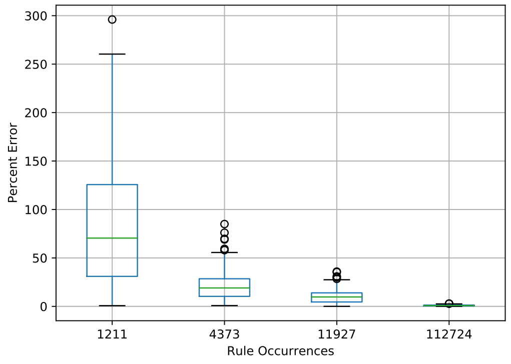


**Figure 6: The ThreeBallot scheme percent error when computing the support over two elements and a varying number of users.**

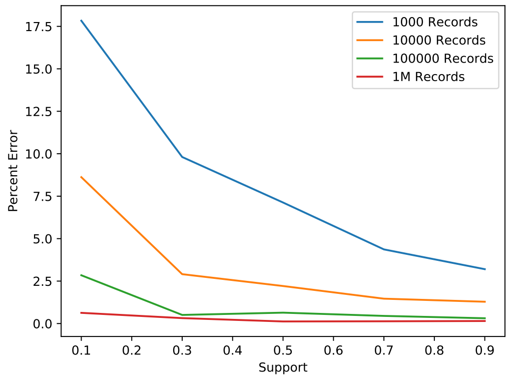


**Figure 7: The ThreeBallot scheme percent error for elements that appear with varying frequency in datasets with different number of users.**

become more frequent, the accuracy of both the percent error ($< 2\%$) and the variance shrink.

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 relative occurrences to the overall number of users (i.e., support). For this reason, we generate four datasets of 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 and every dataset. As the support increases, the percent error shrinks. However, the absolute size of the element set seems to play a much more decisive role in the accuracy of the statistics. This is more visible in the cases of the 100k and 1M datasets, where the support seems to have a minimal effect on the accuracy.

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 $R_i$,

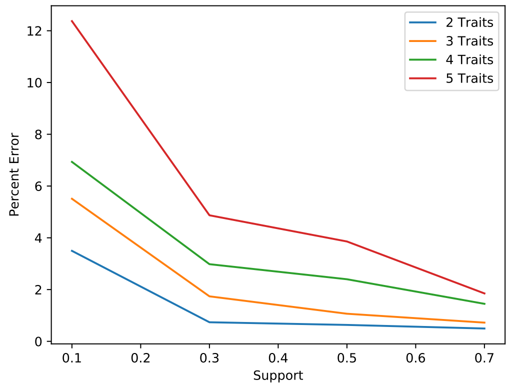


**Figure 8: The ThreeBallot scheme percent error for element sets of varying size that appear with the same frequency i.e., have the same support.**

based on the observed share. Understandably, the inference error for each field adds up with the number of traits.

Based on the above results and the list of statistics reported in the IPCO report [43], 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 traits, 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 [32, 38, 49, 56]. 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 [45, 61, 74]. From this, we conclude that our ThreeBallot scheme is also suitable for the aforementioned types of studies, with a slightly higher expected percent error compared to the law-enforcement use case.

## 6 CONCLUSION

We have proposed and implemented (twice) a system, VAMS, which achieves all our auditability, privacy and verifiability 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.

One aspect that we have mostly ignored is access control. HLF provides some basic access control through endorsement policies, and plans to include more mechanisms in future releases, such as attribute based access control in chaincode. For Trillian, any significant access control would have to be built on top of it. In general, it is also difficult to design and implement realistic access control systems for use cases such as ours. Requests for data are made through paperwork that is read and interpreted by a human, according to policies that are necessarily flexible. The additional need to handle urgent requests also means that any thorough access control system would need to include a way to bypass it anyway. Nonetheless, by providing a system that produces evidence of actions, parties may still be held accountable.

Access to the information in the log is something that we handle by encrypting the requests under the public key of parties that they are relevant too (i.e., auditors and specific users). Whilst enabling a fully public log could lead to greater transparency, it could also lead to more privacy risks, as is often the trade-off. A solution to this is left for future work.

Cryptography also offers techniques such as identity-based encryption [11, 67], attribute based encryption [10, 37], functional encryption [12] and the more recent controlled functional encryption [59] and access control encryption [23] that could be used to control access to information. However these are still rarely used, are often very inefficient and can require a central party controlling a master private key.

## ACKNOWLEDGMENTS

The authors would like to thank Jonathan Bootle for the helpful discussions and suggestions.

Alexander Hicks is supported by VASCO Data Security[6] and UCL through an EPSRC Research Studentship, Vasilis Mavroudis is supported by the European Commission through the H2020-DS-2014-653497 PANORAMIX project, Mustafa Al-Bassam is supported by a scholarship from The Alan Turing Institute, Sarah Meiklejohn is supported in part by EPSRC Grant EP/N028104/1 and in part by a Google Faculty Award, and Steven Murdoch is supported by The Royal Society [grant number UF160505].

## REFERENCES

[1] Ben Laurie Adam Eijdenberg and Al Cutter. 2017. Trillian – Verifiable Data Structures. (2017). Retrieved May 2, 2018 from https://github.com/google/trillian/blob/master/docs/VerifiableDataStructures.pdf

[2] Rakesh Agrawal, Tomasz Imieliński, and Arun Swami. 1993. Mining association rules between sets of items in large databases. In *Acm sigmod record*, Vol. 22. ACM, 207–216.

[3] Rakesh Agrawal and Ramakrishnan Srikant. 1994. Fast Algorithms for Mining Association Rules. *Proc. of the 20th VLDB Conference* (1994), 487–499.

[4] Mário S Alvim, Miguel E Andrés, Konstantinos Chatzikokolakis, Pierpaolo Degano, and Catuscia Palamidessi. 2011. Differential privacy: on the trade-off between utility and information leakage. In *International Workshop on Formal Aspects in Security and Trust*. Springer, 39–54.

[5] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. De Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich, S. Muralidharan, C. Murthy, B. Nguyen, M. Sethi, G. Singh, K. Smith, A. Sorniotti, C. Stathakopoulou, M. Vukolić, S. Weed Cocco, and J. Yellick. 2018. Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains. *ArXiv e-prints* (Jan. 2018). arXiv:cs.DC/1801.10228

[6] Andrew W Appel. 2006. How to defeat Rivest's ThreeBallot voting system. *Manuskrypt, pazdziernik* (2006).

[7] Manuel Barbosa and Pooya Farshim. 2012. Delegatable homomorphic encryption with applications to secure outsourcing of computation. In *Cryptographers' Track at the RSA Conference*. Springer, 296–312.

[8] Adam Bates, Kevin RB Butler, Micah Sherr, Clay Shields, Patrick Traynor, and Dan Wallach. 2015. Accountable wiretapping–or–I know they can hear you now. *Journal of Computer Security* 23, 2 (2015), 167–195.

[9] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, Eran Tromer, and Madars Virza. 2013. SNARKs for C: Verifying program executions succinctly and in zero knowledge. In *Advances in Cryptology–CRYPTO 2013*. Springer, 90–108.

[6]https://www.vasco.com/

[10] John Bethencourt, Amit Sahai, and Brent Waters. 2007. Ciphertext-policy attribute-based encryption. In *Security and Privacy, 2007. SP'07. IEEE Symposium on*. IEEE, 321–334.

[11] Dan Boneh and Matt Franklin. 2001. Identity-based encryption from the Weil pairing. In *Annual international cryptology conference*. Springer, 213–229.

[12] Dan Boneh, Amit Sahai, and Brent Waters. 2011. Functional encryption: Definitions and challenges. In *Theory of Cryptography Conference*. Springer, 253–273.

[13] Elette Boyle, Shafi Goldwasser, and Ioana Ivan. 2014. Functional signatures and pseudorandom functions. In *International Workshop on Public Key Cryptography*. Springer, 501–519.

[14] Benjamin Braun, Ariel J Feldman, Zuocheng Ren, Srinath Setty, Andrew J Blumberg, and Michael Walfish. 2013. Verifying computations with state. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*. ACM, 341–357.

[15] Martin Burkhart, Mario Strasser, Dilip Many, and Xenofontas Dimitropoulos. 2010. SEPIA: Privacy-preserving aggregation of multi-domain network events and statistics. *Network* 1, 101101 (2010).

[16] Christian Cachin. 2016. Architecture of the Hyperledger blockchain fabric. In *Workshop on Distributed Cryptocurrencies and Consensus Ledgers*.

[17] Ran Canetti, Ben Riva, and Guy N Rothblum. 2011. Practical delegation of computation using multiple servers. In *Proceedings of the 18th ACM conference on Computer and communications security*. ACM, 445–454.

[18] Cheng-Kang Chu, Joseph K. Liu, Jun Wen Wong, Yunlei Zhao, and Jianying Zhou. 2013. Privacy-preserving smart metering with regional statistics and personal enquiry services. In *8th ACM Symposium on Information, Computer and Communications Security*. 369–380. https://doi.org/10.1145/2484313.2484362

[19] Jacek Cichoń, Mirosław Kutyłowski, and Bogdan WÈĽglorz. 2008. Short ballot assumption and ThreeBallot voting protocol. In *International Conference on Current Trends in Theory and Practice of Computer Science*. Springer, 585–598.

[20] Graham Cormode, Michael Mitzenmacher, and Justin Thaler. 2012. Practical verified computation with streaming interactive proofs. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*. ACM, 90–112.

[21] Craig Costello, Cédric Fournet, Jon Howell, Markulf Kohlweiss, Benjamin Kreuter, Michael Naehrig, Bryan Parno, and Samee Zahur. 2015. Geppetto: Versatile verifiable computation. In *Security and Privacy (SP), 2015 IEEE Symposium on*. IEEE, 253–270.

[22] Scott A Crosby and Dan S Wallach. 2009. Efficient Data Structures For Tamper-Evident Logging.. In *USENIX Security Symposium*. 317–334.

[23] Ivan Damgård, Helene Haagh, and Claudio Orlandi. 2016. Access control encryption: Enforcing information flow with cryptography. In *Theory of Cryptography Conference*. Springer, 547–576.

[24] A Dekhtyar and J Verburg. 2009. Extended bakery dataset. https://wiki.csc.calpoly.edu/datasets/wiki/ExtendedBakery. (2009).

[25] Antoine Delignat-Lavaud, Cédric Fournet, Markulf Kohlweiss, and Bryan Parno. 2016. Cinderella: Turning shabby X.509 certificates into elegant anonymous credentials with the magic of verifiable computation. In *Security and Privacy (SP), 2016 IEEE Symposium on*. IEEE, 235–254.

[26] Denise Demirel, Lucas Schabhüser, and Johannes Buchmann. 2017. *Privately and Publicly Verifiable Computing Techniques: A Survey*. Springer.

[27] Cynthia Dwork. 2008. Differential Privacy: A survey of results. In *International Conference on Theory and Applications of Models of Computation*. Springer, 1–19.

[28] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. 2006. *Calibrating Noise to Sensitivity in Private Data Analysis*. Springer Berlin Heidelberg, Berlin, Heidelberg, 265–284. https://doi.org/10.1007/11681878_14

[29] Cynthia Dwork, Moni Naor, Toniann Pitassi, and Guy N Rothblum. 2010. Differential privacy under continual observation. In *Proceedings of the forty-second ACM symposium on Theory of computing*. ACM, 715–724.

[30] ETSI. 2016. ETSI TS 103 307 Security Aspects for LI and RD Interfaces. (2016). Retrieved May 2, 2018 from http://www.etsi.org/deliver/etsi_ts/103300_103399/103307/01.02.01_60/ts_103307v010201p.pdf

[31] Alexandre Evfimievski, Ramakrishnan Srikant, Rakesh Agrawal, and Johannes Gehrke. 2004. Privacy preserving mining of association rules. *Information Systems* 29, 4 (2004), 343–364.

[32] Daniel Faria, Andreas Schlicker, Catia Pesquita, Hugo Bastos, António EN Ferreira, Mario Albrecht, and André O Falcão. 2012. Mining GO annotations for improving annotation consistency. *PloS one* 7, 7 (2012), e40519.

[33] Dario Fiore, Cédric Fournet, Esha Ghosh, Markulf Kohlweiss, Olga Ohrimenko, and Bryan Parno. 2016. Hash first, argue later: Adaptive verifiable computations on outsourced data. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 1304–1316.

[34] Frederic Flouvat, F De March, and Jean-Marc Petit. 2005. A thorough experimental study of datasets for frequent itemsets. In *Data Mining, Fifth IEEE International Conference on*. IEEE.

[35] Rosario Gennaro, Craig Gentry, and Bryan Parno. 2010. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In *Annual Cryptology Conference*. Springer, 465–482.

[36] Google. 2017. Trillian. (2017). Retrieved May 2, 2018 from https://github.com/google/trillian

[37] Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. 2006. Attribute-based encryption for fine-grained access control of encrypted data. In *Proceedings*

*of the 13th ACM conference on Computer and communications security*. ACM, 89–98.

[38] Pietro Hiram Guzzi, Milano Marianna, and Mario Cannataro. 2014. Mining Association Rules from Gene Ontology and Protein Networks: Promises and Challenges. *Procedia Computer Science* 29 (2014), 1970–1980.

[39] J. Heaton. 2016. Comparing dataset characteristics that favor the Apriori, Eclat or FP-Growth frequent itemset mining algorithms. In *SoutheastCon 2016*. 1–7. https://doi.org/10.1109/SECON.2016.7506659

[40] Kevin Henry, Douglas R Stinson, and Jiayuan Sui. 2009. The effectiveness of receipt-based attacks on ThreeBallot. *IEEE Transactions on Information Forensics and Security* 4, 4 (2009), 699–707.

[41] Briland Hitaj, Giuseppe Ateniese, and Fernando Pérez-Cruz. 2017. Deep models under the GAN: information leakage from collaborative deep learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 603–618.

[42] Home Office. 2016. Operational case for the use of communications data by public authorities. https://www.gov.uk/government/publications/investigatory-powers-bill-overarching-documents. (2016). Retrieved May 2, 2018 from https://www.gov.uk/government/publications/investigatory-powers-bill-overarching-documents

[43] Interception of Communications Commissioner's Office. 2017. Report of the Interception of Communications Commissioner - Annual Report for 2016. (2017). Retrieved May 2, 2018 from https://assets.publishing.service.gov.uk/government/uploads/system/uploads/attachment_data/file/670219/IOCCO_annual_report_2016_2.PDF

[44] Marek Jawurek and Florian Kerschbaum. 2012. Fault-tolerant privacy-preserving statistics. In *International Symposium on Privacy Enhancing Technologies Symposium*. Springer, 221–238.

[45] Peter B Jensen, Lars J Jensen, and Søren Brunak. 2012. Mining electronic health records: towards better research applications and clinical care. *Nature Reviews Genetics* 13, 6 (2012), 395.

[46] Harvey Jones, Jason Juang, and Greg Belote. 2006. ThreeBallot in the Field. *Term paper for MIT course* 6 (2006).

[47] Gabriel Kaptchuk, Matthew Green, and Aviel Rubin. 2017. Outsourcing Medical Dataset Analysis: A Possible Solution. In *International Conference on Financial Cryptography and Data Security*. Springer, 98–123.

[48] Ahmed E Kosba, Dimitrios Papadopoulos, Charalampos Papamanthou, Mahmoud F Sayed, Elaine Shi, and Nikos Triandopoulos. 2014. TRUESET: Faster Verifiable Set Computations.. In *USENIX Security Symposium*. 765–780.

[49] Anand Kumar, Barry Smith, and Christian Borgelt. 2004. Dependence relationships between Gene Ontology terms based on TIGR gene product annotations. In *Proceedings of CompuTerm 2004: 3rd International Workshop on Computational Terminology*.

[50] Ben Laurie and Emilia Kasper. 2012. Revocation transparency. *Google Research, September* (2012).

[51] Ben Laurie, Adam Langley, and Emilia Kasper. 2013. *RFC 6962 – Certificate transparency*. Technical Report.

[52] Jaewoo Lee and Chris Clifton. 2011. How much is enough? choosing $\varepsilon$ for differential privacy. In *International Conference on Information Security*. Springer, 325–340.

[53] Ninghui Li, Tiancheng Li, and Suresh Venkatasubramanian. 2007. t-closeness: Privacy beyond k-anonymity and l-diversity. In *Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on*. IEEE, 106–115.

[54] Ashwin Machanavajjhala, Johannes Gehrke, Daniel Kifer, and Muthuramakrishnan Venkitasubramaniam. 2006. l-diversity: Privacy beyond k-anonymity. In *Data Engineering, 2006. ICDE'06. Proceedings of the 22nd International Conference on*. IEEE.

[55] Kathleen Moriarty, Burt Kaliski, and Andreas Rusch. 2017. PKCS# 5: Password-Based Cryptography Specification Version 2.1. (2017).

[56] Anurag Nagar, Michael Hahsler, and Hisham Al-Mubaid. 2015. Association rule mining of gene ontology annotation terms for SGD. In *Computational Intelligence in Bioinformatics and Computational Biology (CIBCB), 2015 IEEE Conference on*. IEEE, 1–7.

[57] Arjun Narayan, Ariel Feldman, Antonis Papadimitriou, and Andreas Haeberlen. 2015. Verifiable Differential Privacy. In *Proceedings of the Tenth European Conference on Computer Systems*. ACM, 28.

[58] Arvind Narayanan and Vitaly Shmatikov. 2008. Robust de-anonymization of large sparse datasets. In *Security and Privacy, 2008. SP 2008. IEEE Symposium on*. IEEE, 111–125.

[59] Muhammad Naveed, Shashank Agrawal, Manoj Prabhakaran, XiaoFeng Wang, Erman Ayday, Jean-Pierre Hubaux, and Carl Gunter. 2014. Controlled functional encryption. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 1280–1291.

[60] National Health Service (NHS). 2017. National Diabetes Audit Report. (2017). Retrieved May 2, 2018 from https://digital.nhs.uk/data-and-information/publications/statistical/national-diabetes-audit/national-diabetes-audit-report-1-care-processes-and-treatment-targets-2016-17

[61] So Hyun Park, Shin Yi Jang, Ho Kim, and Seung Wook Lee. 2014. An association rule mining-based framework for understanding lifestyle risk behaviors. *PloS one* 9, 2 (2014), e88859.

[62] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. 2013. Pinocchio: Nearly practical verifiable computation. In *Security and Privacy (SP), 2013 IEEE Symposium on*. IEEE, 238–252.

[63] Ronald L Rivest. 2006. The ThreeBallot voting system. (2006).

[64] Ronald L Rivest and Warren D Smith. 2007. Three voting protocols: ThreeBallot, VAV, and Twin. *USENIX/ACCURATE Electronic Voting Technology (EVT 2007)* (2007).

[65] Srinath Setty, Benjamin Braun, Victor Vu, Andrew J Blumberg, Bryan Parno, and Michael Walfish. 2013. Resolving the conflict between generality and plausibility in verified computation. In *Proceedings of the 8th ACM European Conference on Computer Systems*. ACM, 71–84.

[66] Srinath TV Setty, Richard McPherson, Andrew J Blumberg, and Michael Walfish. 2012. Making argument systems for outsourced computation practical (sometimes).. In *NDSS*, Vol. 1. 17.

[67] Adi Shamir. 1984. Identity-based cryptosystems and signature schemes. In *Workshop on the theory and application of cryptographic techniques*. Springer, 47–53.

[68] Elaine Shi, HTH Chan, Eleanor Rieffel, Richard Chow, and Dawn Song. 2011. Privacy-preserving aggregation of time-series data. In *Annual Network & Distributed System Security Symposium (NDSS)*. Citeseer.

[69] Joao Sousa, Alysson Bessani, and Marko Vukolić. 2017. A byzantine fault-tolerant ordering service for the hyperledger fabric blockchain platform. *arXiv preprint arXiv:1709.06921* (2017).

[70] Charlie Strauss. 2006. The trouble with triples: A critical review of the triple ballot (3ballot) scheme, part 1. *Verified Voting New Mexico* (2006).

[71] Charlie EM Strauss. 2006. 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).

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

[73] Jun Tang, Aleksandra Korolova, Xiaolong Bai, Xueqiang Wang, and Xiaofeng Wang. 2017. Privacy Loss in Apple's Implementation of Differential Privacy on macOS 10.12. *arXiv preprint arXiv:1709.02753* (2017).

[74] Giulia Toti, Ricardo Vilalta, Peggy Lindner, Barry Lefer, Charles Macias, and Daniel Price. 2016. Analysis of correlation between pediatric asthma exacerbation and exposure to pollutant mixtures with association rule mining. *Artificial intelligence in medicine* 74 (2016), 44–52.

[75] Jelle van den Hooff, M Frans Kaashoek, and Nickolai Zeldovich. 2014. Versum: Verifiable computations over large public logs. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 1304–1316.

[76] Victor Vu, Srinath Setty, Andrew J Blumberg, and Michael Walfish. 2013. A hybrid architecture for interactive verifiable computation. In *Security and Privacy (SP), 2013 IEEE Symposium on*. IEEE, 223–237.

[77] Marko Vukolić. 2017. Rethinking Permissioned Blockchains. In *Proceedings of the ACM Workshop on Blockchain, Cryptocurrencies and Contracts*. ACM, 3–7.

[78] Riad S Wahby, Max Howald, Siddharth Garg, Abhi Shelat, and Michael Walfish. 2016. Verifiable asics. In *Security and Privacy (SP), 2016 IEEE Symposium on*. IEEE, 759–778.

[79] Michael Walfish and Andrew J Blumberg. 2015. Verifying computations without reexecuting them. *Commun. ACM* 58, 2 (2015), 74–84.

[80] Karl Wüst and Arthur Gervais. 2017. Do you need a Blockchain? *IACR Cryptology ePrint Archive* 2017 (2017), 375.

[81] Peter Yee. 2013. Updates to the internet X. 509 public key infrastructure certificate and Certificate Revocation List (CRL) profile. (2013).

[82] Nan Zhang, Shengquan Wang, and Wei Zhao. 2004. A new scheme on privacy preserving association rule mining. In *European Conference on Principles of Data Mining and Knowledge Discovery*. Springer, 484–495.