VAMS: Verifiable Auditing of Access to Confidential Data

Alexander Hicks, Vasilios Mavroudis, Mustafa Al-Bassam, Sarah Meiklejohn, Steven J. Murdoch
University College London
{alexander.hicks, v.mavroudis, mustafa.al-bassam.16, s.meiklejohn, 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.

I. 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 such cases, rules cannot reliably be automatically enforced in real-time so the approach commonly taken is to keep records of access attempts and subject the actions to audit. Provided that the audit 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 see are accurate. If individuals at risk of their personal data being misused do not trust that the auditor is faithfully carrying out their duties then the goal of the audit will not be achieved. However, because of the sensitivity of personal data and the records containing the justification for data being processed, not everyone can act as an auditor. Even if it was possible to find an organization whose audit would be widely accepted, an audit based on tampered records would not be reliable.

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. 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.

A. 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) Law-enforcement access to communications data: In the UK 95% of serious and organized criminal cases make use of
communications data [1] – metadata stored by telecommunications providers in their billing system about account holders or their use of communications networks (e.g., phone numbers called, address associated with an account, location of a mobile phone). Telecommunications providers are required to store this data for 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 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 publish reports with statistics of how these powers were used [2].

Communications data plays an important role in the investigation of criminal offenses, but may also be used as evidence in legal proceedings, for the prosecution or defense. Should questions be raised about the integrity of the evidence, a senior representative of the telecommunications provider will be asked to appear in court and 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 it and it being required in court.

To improve the process, industry standards allow 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 [3]. 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 it 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).

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 (e.g., according to 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.

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.

B. 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 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. While Hyperledger Fabric provides more flexible policies and a wider variety of trust assumptions, Trillian is more scalable and would be much easier to deploy today. We also evaluate our privacy scheme, measuring accuracy and privacy loss in different scenarios.

II. RELATED WORK

This section examines existing prior work in the area and discusses how this paper compares to it. This prior work is typically based on one of three applicable techniques: data anonymization, verifiable computations and tamper-evident logging. We discuss each of these in turn.

A. Data anonymization

Given our need for both data privacy and public verifiability of statistics, data anonymization could be a solution for our use cases. Data anonymization techniques aim to remove any information linkable to individual users, while retaining as much of the information content. Auditors could release, along with their statistics, anonymized versions of the datasets they processed so that everyone can verify the reported results.

Data pseudonymization is a straightforward solution to this and aims to protect the data subjects of all user identifiers (e.g., names) with randomly generated ones. However, as it was found in the case of the Netflix Prize [4], an adversary can often leverage side information to de-anonymize individuals and thus link a user’s random identifier to their real identity. To address this problem, many other anonymization techniques have been introduced, with the most popular being k-anonymity [5] and its extensions l-diversity [6] and t-closeness [7]. However, k-anonymity and l-diversity have already been shown to be vulnerable to various attacks, while t-closeness ends up leaving very little useful information in the dataset [8].

Another promising line of work are techniques for privacy-preserving association rule mining [9]. Those techniques generate a randomized or perturbed dataset that protects the privacy of the individual users, while it preserves the associations between the variables. In the context of our work, such a randomized dataset could be release along with the statistics for the users to replicate the computations and verify that their records are correctly represented. One of the very first randomization solutions proposed uniform randomization, where individual user records are uniformly randomized based on a public factor. However, as pointed out by Evfimievski et al. [10], 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 another class of randomization operators that achieve much better privacy. However, even these operators achieve unrealistically low privacy [11], while they require datasets of at least one million records.

B. Verifiable computations

Another class of potential solutions are provided by works on verifiable computations. The goal of such systems is to allow clients to verify the correctness of outsourced computations, without replicating the computations themselves. As discussed by Walfish and Blumberg [12], verifiable computing schemes come with an very high overhead for the prover, especially in cases when input and output privacy are needed. We focus on the limited number of non-interactive verifiable schemes that achieve output privacy (i.e., anyone can verify the correctness of the computations without learning anything about the original dataset). Boyle et al. [13] present constructions including succinct function private functional signatures, and Barbosa et al. [14] present a delegable homomorphic encryption primitive. Both works rely on advanced cryptographic techniques (SNARKs, fully homomorphic encryption, and functional encryption) and achieve strong privacy guarantees. However, they also come with a very high computational overhead, while having only limited language expressiveness. Moreover, these works provide input privacy, which is a property that is unnecessary for our use cases. Finally, [15] introduces VerDP a system that allows analysts to submit queries and receive the results in a differentially-private form along with zero-knowledge proofs of correctness. VerDP suffers from the same high-overhead drawback as the generation of a proof (for each of the published statistics) requires several days for datasets with thousands of records. More importantly, VerDP (based on Fuzz [16]) does not sufficiently support the query types used for association rule mining.

Overall, most of the existing verifiable computation schemes opt for a generic solution that covers as many computation types as possible. This comes at the cost of high overhead [17], while it still does not cover several other computation types. Instead, we prefer to use a lightweight solution tailored to the needs of our two use cases.

C. Tamper-evident logging

Crosby and Wallach [18] consider the case of a untrusted logger serving clients storing events in a log kept honest through auditing, which they refer to as tamper-evident logging. They use a centralized hash-tree based log and do not address secrecy of logged events or replication. Bates et al. [19] look at accountable logs of wiretapping. 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 subject to oversight after the fact.
More recently, Goldwasser and Park [20] have considered the use of append-only ledgers of zero-knowledge proofs to provide auditability for actions related to secret laws. In similar work Frankle et al. [21] propose a system that allows accountability of secret processes based on multi-party computations and zero-knowledge proofs. Both of the above differ from our work in that they are inspired by the US setting and take a different approach to providing accountability, based on cryptographic proofs rather than verifiable audits.

III. BACKGROUND

In this section, we introduce the building blocks of our system: Hyperledger Fabric, Trillian, and the ThreeBallot voting scheme.

A. Hyperledger Fabric

Hyperledger Fabric (HLF) [22] describes itself as a modular, extensible open-source system for deploying and operating permissioned blockchains and includes architectural differences to most existing solutions [23], [24].

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

Peers have identities and 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.

B. Trillian

Trillian [25] is an open-source project that implements a generalization of Certificate Transparency (CT) [26], based on a verifiable log backed by a verifiable map [27].

The verifiable log is an append-only log implemented as a Merkle tree (as in CT) that allows clients to efficiently verify that an entry is included in the log with a proof showing the Merkle path to the tree’s entry, detect log equivocation (i.e., conflicting tree heads) and verify that the log is append-only through Merkle consistency proofs. The verifiable map is a key-value store implemented as a sparse Merkle tree i.e., a Merkle tree pre-populated with all possible keys as leaves e.g., all $2^{256}$ possible SHA-256 hashes. Although a tree with $2^{256}$ unique leaves would in principle not be practical to compute, only the non-empty leaves have to be computed as all others will have the same value (e.g., zero) [28]. Clients can then verify that a certain value is included (or not) in the map at any point in time, with proofs containing Merkle paths.

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

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

C. ThreeBallot voting system

ThreeBallot [29], [30] is a paper-based voting scheme proposed by Rivest for end-to-end auditable elections.

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

The scheme’s security properties (i.e., auditability and vote privacy) have been extensively studied in various works [30]–[36]. From all the attacks introduced in the literature, the reconstruction and pattern-based attacks are applicable to our use cases. Henry et al. [33] published a thorough analysis on these (against two candidate races), extending on the previous work by Strauss [35]. These works provide a lower bound for security as a function of the ballot size (number of binary choices – candidates). In Section 3, we use these bounds to derive the maximum number of elements that our system can support, while retaining its security properties.

IV. SETTING AND THREAT MODEL

A. Setting and notation

Our proposed system is composed of agents, data providers, users, auditors, log servers and optional data brokers. External to the system are also regulators that are not active in the system but define regulations (e.g., the Investigatory Powers Act of 2016) that determine the rules obeyed by the parties in the system. The system’s parties and functionalities (detailed in Table I) are defined as follows:

Agents e.g., public authorities, companies or generally any party wishing to obtain user data from data providers (request).
To serve as a data provider; e.g., providing data to an agent can rely on to deal with data requests if they are willing to participate.

Data brokers are non-essential intermediaries which users can rely on to deal with data requests if they are willing to serve as a data provider; e.g., providing data to an agent running a study. The data broker can then deal with these requests (broker) according to pre-set rules from the user; e.g., in which type of study they are willing to participate.

The log The log is a key-value store of information such that keys are used by parties to identify stored values that are relevant to them. The information stored could take any form, but in this work we are primarily concerned with (possibly encrypted) records corresponding to requests made by agents.

These are tuples of elements that split up the information in the record; e.g., the attributes of a data request (type, urgency, etc.) or some medical questionnaire. Other information that may be published in the log by auditors are datasets and statistics. These can be simply stored as value in plaintext or, if size is an issue, the value can contain a link to the statistics and dataset along with a hash to verify their integrity.

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.

We assume that all parties may be malicious, with two important caveats. First, we do not allow malicious agents and data providers to collude, as they could then simply choose to not use the system. Second, malicious servers may attempt to corrupt the system by giving different parties different views of the log, but our system allows users and auditors to detect this. This is thus functionally equivalent to modeling log servers as honest-but-curious, as they could be removed from the system as a result of any detected misbehavior.

### B. Security goals

We now define the two main security goals in this setting, which are auditability and privacy. In essence, auditability is about ensuring the integrity of the information on the system and the results of audits, while privacy is about ensuring that the only information that can be gained by anyone is either already known to them or known to everyone.

In terms of auditability, we allow all parties to be malicious except for data providers, and consider separately the cases of auditors and users performing audits. For an auditor, we require that the auditor be able to perform audit properly, meaning they can output accurate and verifiable statistics on the log. For a user, we require that they be able to perform check and monitor, meaning they can extract their own identifiers from the log and verify the published statistics.

In terms of privacy, the goal is to ensure that no information that is not already public can be gained by any malicious party. This may mean linking requests together in order to gain information about the agents, data providers or users linked to a request, or gaining more information about individual entries than is revealed by the published statistics. We 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. The adversary
must guess the last field with a noticeably higher probability than if they had access only to the transformed dataset.

V. OUR SYSTEM: VAMS

We now present an overview of the form our system will take, before specifying the proposed mechanisms that satisfy our security goals. In particular, as the separate functionalities offered by our system already exist separately in various forms (discussed in Section II), we justify our design choices which allow us to combine everything into one cohesive ensemble.

A. Overview of system requirements

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

Clearly, the system should contain some form of database of requests submitted by agents through request, which is maintained by log servers performing host. As auditors and users should be able to detect malicious log servers through detect and detect, the 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 audit and 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 I-A) are court cases where law enforcement or a healthcare company is 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 publish. Users must be able to verify these statistics through monitor, there must be evidence of the results they publish, as well as the data necessary to verify their results, without compromising privacy.

To achieve the requirements above, a data structure such as a blockchain or a form of Merkle tree is required. We choose to use existing solutions: blockchain-based permissioned ledgers (Hyperledger Fabric) and verifiable log-backed maps (Trillian), introduced in Sections III-A and III-B. Both store key-value maps, either as the state of the ledger, 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 that allows users performing monitor to verify the results without having to trust the auditor that published them. This is in line with the use case for permissioned blockchains outlined by Wüst and Gervais [37], as a state must be stored with multiple known and untrusted writers, no always-online trusted third party, and verifiability requirements.

B. 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 VI.

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

Publishing requests to the log: When performing 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 ida and iddp, 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 (ida, iddp), the pair can be re-used by the agent and data provider to generate new common identifiers (which we denote idc) by concatenating the session identifier to the encrypted identifier; i.e., they can compute idc = Enc(ida, iddp ∥ n).

As ida and iddp may be short and have little entropy, a key derivation function (KDF) such as PBKDF2 [38] should be used to obtain a more resilient ciphertext, with ida being fed through the KDF before being used as an encryption key.

The unlinkability that results from this usage of identifiers is argued in security argument 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.
Checking requests in the log: Once requests are made, auditors and users can check the log using audit and check, assuming they have first determined the log servers to not be malicious by performing detect, which we argue they can in security argument 1. Accessing the key-value map, they can then rely on the integrity properties of the log, as argued in security argument 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 provide, and also allow users to act as data providers if they are willing to, for example, participate in a study. One downside of this is that the data broker must then be trusted with the private identifier tied to requests that the user positively answers. However, no other trust is required as VAMS allows the user to check the activity of the broker, which will be logged and be auditable under the same guarantees as other log entries.

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 publish. An example of what might be published are the statistics provided by the IPCO in its annual report [2], 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 monitor (as argued in security argument 2). To ensure statistics reveal no more than the statistics themselves (as argued in security argument 3) 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 that of univariate statistics. The transformation generates a privacy-preserving dataset \( D_{\text{priv}} \) 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 \( \text{id}_{\text{share}} = \text{Hash(id}_{i} \mid \text{i}) \) constructed from the common identifier \( \text{id}_{i} \) of the user and index \( i \) of the share. The auditor then adds \( D_{\text{priv}} \) to the ledger along with the analysis results (statistics). Users can then check the presence of their shares in \( D_{\text{priv}} \), and re-compute the published statistics from \( D_{\text{priv}} \) to check their validity. In the case of Hyperledger Fabric, the use of chaincode allows users to publicly report the results of the verification of their records, and thus collectively confirm (or not) the integrity of the published data.

If multivariate statistics are required, a scheme based on ThreeBallot (presented in Section III-C) can be used to generate \( D_{\text{priv}} \). Each record is split into three shares that 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 \( \epsilon \), the false value for \( e \). Figure 2 illustrates this. As argued in security arguments 3 and 2, this process both prevents an adversary from breaching the privacy of individuals and preserves the correlation between the elements. Given \( D_{\text{priv}} \), it is then possible to compute multivariate statistics for the original dataset \( D \), with only a small error. Auditors can then publish \( D_{\text{priv}} \), so that users can verify that their shares are accurately represented, and statistics can be verified.

Association rule mining [9] is one of the most commonly used approaches to identify if-then rules and relationships between variables in large datasets (e.g., healthcare data). Given an element set \( E = \{e_1, e_2, \ldots \} \) of binary elements of a record and a dataset \( D = \{R_1, R_2, \ldots \} \) of records containing elements that form a subset of \( E \), a rule is an implication \( e \Rightarrow e' \) where \( e, e' \subseteq E \). 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 V.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 \( \epsilon \in E \).

\[
\text{support}(\epsilon) = \frac{|\{R \in \delta : \epsilon \in R\}|}{|\delta|} \quad (V.1)
\]

The support of a rule \( \epsilon \Rightarrow \epsilon' \), is simply the support of the joint element sets i.e., \( \text{support}(\epsilon \Rightarrow \epsilon') = \text{support}(\epsilon \cup \epsilon') \). From the above, we can also compute confidence (defined in equation V.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 \( \epsilon \).

\[
\text{conf}(\epsilon \Rightarrow \epsilon') = \frac{\text{supp}(\epsilon \Rightarrow \epsilon')}{\text{supp}(\epsilon)} \tag{V.2}
\]

Computing these values on \( D \) is straightforward but some pre-processing is needed to extract them from \( D_{\text{priv}} \), which contains elements \( \tau \), so 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 \( D_{\text{priv}} \). 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 \( o_{\text{priv}} \) with all bitstring occurrences for \( D_{\text{priv}} \).

\[
o_{D}, o_{\text{priv}} = \begin{bmatrix} \#[0] \\ \vdots \\ \#[2^{t} - 1] \end{bmatrix} \tag{V.3}
\]

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

\[
M = \begin{bmatrix} \mathbb{E}(\#[0])_{0} & \cdots & \mathbb{E}(\#[0])_{2^t-1} \\ \vdots & \ddots & \vdots \\ \mathbb{E}(\#[2^t - 1])_{0} & \cdots & \mathbb{E}(\#[2^t - 1])_{2^t-1} \end{bmatrix} \tag{V.4}
\]

We can then estimate \( o_{\text{priv}} \) from the product of \( M \) and \( o_{D} \) by computing \( o_{\text{priv}} = M \cdot o_{D} \). In our case, \( o_{\text{priv}} \) is obtained from \( D_{\text{priv}} \) and it is \( o_{D} \) that we are interested in. We can solve the previous equation for \( o_{D} \) by inverting\(^3\) \( M \) and multiplying it with \( o_{\text{priv}} \). This gives \( o_{D} \approx M^{-1} \cdot o_{\text{priv}} \).

Based on the inferred \( o_{D} \), we can now compute the support and confidence measures for any element sets \( \epsilon, \epsilon' \). The accuracy of this method is evaluated in Section VI.

It should be noted that as \( D_{\text{priv}} \) is used only for verification of the reported statistics and not for mining new associations, this leaves plenty of room for minimizing the information content of \( D_{\text{priv}} \). In particular, if \( D \) is composed of records with a large number of elements, but only few of these have interesting correlations that are relevant in the published statistics, then only these need to be included in \( D_{\text{priv}} \). This can significantly reduce the size of \( D_{\text{priv}} \) compared to \( D \), especially in cases of datasets sparse in relationships. Moreover, our technique can in certain cases support statistics involving continuous variables. For example, while during the rule mining phase the researcher may need to examine

\(^3\) \( M \) 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.

C. Security arguments

We now argue the security of our system in the adversarial model described in Section IV, based on the mechanisms proposed above. We split our arguments into three parts: detecting log misbehavior, auditability and privacy.

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

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

In the Trillian case, a log server that equivocates would have to produce signed tree heads and Merkle consistency proofs for the alternative Merkle trees. Different Merkle consistency proofs leading from the same Merkle tree generate different views of the log, but these differing logs are no longer able to accept the same Merkle consistency proofs to extend the logs because the leaves are different. As the tree heads are signed by the log server, the two signed inconsistent tree heads can be used as evidence to implicate the log server [39], [40].

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

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

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

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

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

For users, our argument is split according to the two types of audits that are possible: check and monitor. In the case of check, the argument is the same as for auditors and audit. In the case of monitor, by the same arguments we again have that the user has a correct and complete view of the log. A malicious auditor could nonetheless perform publish adversarially, publishing incorrect statistics or the wrong dataset. The user can then detect this malicious behavior by performing monitor. A user that was included in the used dataset can check the integrity of the transformed dataset, identifying their shares to verify their correctness. Once the integrity of the data is confirmed, any other user can replicate the computations of the analysis and compare their results with those published, detecting any false statistics.

Security argument 3 (Privacy). Given that the information on the log is encrypted, our argument for privacy is split into two parts. First, we argue that it is not possible to gain information from the log; i.e., that log entries are unlinkable. Second, we argue that our scheme for statistics preserves privacy.

For unlinkability, we assume that AES is a secure pseudorandom permutation, and that session identifiers are used only once for each pair of (pseudorandom) private identifiers and argue it is not possible to link two or more requests (i.e., common identifiers) that appear in the log except for the relevant users.

A user or an adversary knowing and will be able to find requests relevant to the user with probability 1 by performing check. For an adversary knowing only of and , determining the other input would require iterating over possible values of as well as , which would require () encryption operations, where is the length of and is the range of values takes. This would only reveal one pair of private identifiers for a user. With regards to agent-data provider unlikeliness, each pairing is equally as likely, which gives probability (where and are the sets of agents and data providers) of determining one. If an adversary knows neither nor , 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 with any reasonable probability.

We now examine if our publicly verifiable statistics scheme fulfills our security goals as they were defined in Section IV-B. We argue that even an adversary with knowledge of all but one elements of a user’s record cannot exploit the remaining (unknown) element (i.e., inference risk).

The argument is straightforward for univariate statistics as every share is split from others, and tagged with unlinkable pseudorandom identifiers as long as the hash function used is pre-image resistant. The more interesting case is that of the ThreeBallot scheme used for multivariate statistics. For a successful inference attack the adversary needs to link ballots (shares) from to their original multiballot (record) from , uncovering the remaining element of the user’s record. The feasibility and effectiveness of such attacks have been analysed in several works. Overall, those works examine different reverse-mapping methods that exploit the ballot marking rules (e.g., no blank rows are allowed) to rule out invalid combinations. Given a large enough number of concurrent races and a relatively small number of voters, these attacks can correctly retrieve from .

The most recent work on those bounds is that of Henry et. al [33]. Their work studies the feasibility of the Reconstruction and the ThreePattern attacks (the most efficient attacks known) against elections with multiple two-candidate races. Their analysis concludes that the upper bounds are , and simultaneous races given voters respectively. These bounds guarantee the privacy of a voter , even if an adversary coerces into using a specific voting pattern i.e., is handed three ballots marked with the least-frequent patterns. In particular, Henry et. al prove (both theoretically and through simulations) that all possible ballot patterns will appear in at least once (with probability negligibly smaller than 1), regardless of ’s actual vote. Thus, cannot distinguish if it voted as agreed or not.

We now explain why these results and bounds hold also when ThreeBallot is used for verifiable association mining, instead of elections. In that case, the adversary has no information about the victim’s ballots (or shares) in , while in the election setting holds one of the victim’s three ballots/shares. This reduces the information available to the adversary. However, we assume that knows all but one of the elements in ’s record in (without knowing how these were split in ). Adversary can use this knowledge of the user’s elements to combine shares from into records that could potentially belong to the victim user. Obviously, if there is only one possible combination of shares that results in a record matching ’s knowledge for , then the remaining element is trivially uncovered. Henry’s bound [33] guarantees that the probability of this happening is negligible. In particular, if could identify three shares , and that match their knowledge of , there will always be at least another share that differs from only on the last element. As that the elements hold binary values, and and are equally likely to belong to , does not learn anything new.

An important detail is that Henry et. al [33] assumed statistical independence between the races. While this may be true in certain settings, in our use cases it is likely that different variables will exhibit correlation. Strauss [35] studied various such settings with correlated races and showed that even heavily correlated races had only a minor effect on the security of the scheme [35]. In cases where a percentage of the records in are known to the adversary, only the number of the remaining (unknown) records must be considered when
estimating the upper bounds.

VI. IMPLEMENTATION AND PERFORMANCE

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

A. Hyperledger Fabric

The modularity of Hyperledger Fabric (HLF, introduced in Section III-A) makes it a good candidate for our use case. The network maintains a key-value store that can be populated by requests linked to common identifiers. These requests can then easily be retrieved by querying specific keys or a range of keys in lexical order. A key history function is also available. State updates correspond to transactions on the underlying blockchain, making them verifiable. 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 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 request), the second is used to retrieve a range of key values (as part of audit), the third is used to retrieve values for specific keys (as part of check) and the fourth is used to retrieve a key’s history (as part of audit and check) to see which blockchain transaction resulted in state updates for a given key. Chaincode invocations result in state updates recorded in blocks on the blockchain, that then appear in the log, which is the state of the ledger i.e., the key-value store.

Peers have identities, X.509 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 require multiple signatures, they could hold multiple parties accountable e.g., if data providers were considered responsible for accepting invalid requests, they could be required to sign the corresponding request transactions. An ordering service of specific peers (e.g., auditors) could also be used to detect and flag invalid requests as they are initially processed (and endorsement policies are checked) before committing the requests. These are not present in our implementation, but give an idea of what may be possible as Hyperledger Fabric undergoes continued development and implements further cryptographic tools.

B. Trillian

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

As part of request, agents append signed requests they have sent to the log, which data providers can then check. There is no built-in identity system, so the log server service responsible for receiving new requests must check that they are signed. A map server then monitors the log for new entries, and updates the map according to the new entries—the common identifiers are used as the keys in the map. It then periodically publishes signed map heads that are written to the second verifiable log, solely responsible for keeping track of published signed map heads. To perform check, users can then query the map to efficiently check their possible common identifier values. The map will return a Merkle proof of non-inclusion.

---

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 [42], it is not yet available.
for common identifiers that do not map to requests (i.e. the common identifier maps to a zero value), or a Merkle proof of inclusion of requests that the common identifiers do map to. Auditors performing audit can in turn check that the map is operated correctly by replaying all log entries, verifying that they correspond to the same map heads that were written to the second verifiable log tracking signed map heads.

C. Trade-offs

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

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

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

Table II 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.

D. Performance measurements

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

b) Throughput: Table III 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.

| Features       | HLF | Trillian |
|----------------|-----|----------|
| User privacy   | ●   | ●        |
| Agent privacy  | ●   | ●        |
| Data provider privacy |       | ●        |
| Statistical privacy | ●   | ●        |
| User auditability | ●   | ●        |
| External auditability |       | ●        |
| Verifiability  | ●   | ●        |
| Access control | ●   | 0        |

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

Table III: 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.
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 [2], 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 [3]) 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.

c) Private statistics accuracy: To evaluate the effectiveness and applicability of our scheme, we measure the accuracy of the rule association metrics computed on $D_{\text{priv}}$.

For our experiments, we generate multiple synthetic datasets with several frequent element sets [44], then mine those itemsets using the Apriori algorithm [45]. 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 $D$ (described in Section V-B). We then compute the support and confidence measures on $D_{\text{priv}}$ 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 [2], and the healthcare use case e.g., proportions of people registered with diabetes that achieved blood glucose, pressure and cholesterol targets [46]. We opt to use synthetic datasets to examine the accuracy offered by our ThreeBallot scheme more thoroughly, by simulating different scenarios rather than relying on public datasets that have already been sanitized. However, we also verify our reported results using commonly used public datasets, such as the Extended Bakery dataset [47] and the T10I4D100K dataset [48]. In all our experiments, we measure the error for both the support and the confidence metrics. We include only the graphs for support, as those for confidence are identical. Each experiment is repeated 100 times.

In our first experiment, we study the percent error for the support over two elements when varying the number of rule occurrences for a dataset of 1 million users. 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. As element sets become more frequent, the percent error ($<2\%$) and the variance both 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 occurrences relative to the overall number of users (i.e., support). 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 in every dataset. The percent error shrinks as the support increases, but the absolute size of the element set seems to play a more decisive role in the accuracy of the statistics. In the cases of the $100k$ and $1M$ user datasets, the support seems to have a minimal effect on the accuracy.

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

Based on the above results and the list of statistics reported in the IPCO report [2], our ThreeBallot scheme is suitable for the law-enforcement use case. To better evaluate its suitability for healthcare data, we now look into the relevant medical and biostatistics literature. We consider two types of studies: Studies on Genes and Protein networks, and Epidemiology studies. In the first type, datasets commonly contain between 100,000 and a few million records, while the support threshold is usually around 0.5%. In most cases, valid association rules are comprised of only two elements, while their support is higher than the minimum threshold. This is important as the minimum threshold is relevant only during the rule mining phase, while in the verification phase the users compute measures over the relationships that are reported by the researcher as strongly associated [49]–[52]. 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 [53]–[55]. We conclude that our ThreeBallot scheme is also suitable for these types of studies, with a slightly higher expected percent error compared to the law-enforcement use case.

VII. DISCUSSION

A. Deployability

For a system like VAMS to be deployed, agents and data providers would need to implement the necessary infrastructure. They may do so voluntarily, so as to increase public confidence that their access to personal data is legitimate [56]. Participants also may choose to implement VAMS to allow them to demonstrate that personal data has not been tampered with when used as evidence, with the transparency properties just being a desirable side-effect of this activity.

Alternatively, participants may have a statutory obligation to provide transparency e.g., compliance with ETSI requirements may be a condition of providing a telecommunication service, and their standards do include provision for requiring authenticity and transparency of access to personal data [3]. In the UK, the Investigatory Powers Commissioner has the right to require that public authorities and telecommunication operators provide the commissioner’s office with any assistance required to carry out audits and this could include implementing IT infrastructure [57, Section 235(2)]. Another possible route for imposing a statutory requirement to provide transparency could be through enforcement action by a regulator such the Federal Trade Commission or a data protection authority.

B. Access control

Access control is a topic that we have not addressed in great detail as it is difficult, and perhaps unrealistic, to implement strict access control to the system that matches real use cases where paperwork is used and examined by humans against policies that can be vague and 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 control to information in the system may also be discussed. We have handled this by encrypting all the information on the log and relying on public key encryption for access to information, but a solution allowing a fully public log without harming privacy would also be of interest.

Other cryptographic techniques such as identity-based encryption [58], [59], attribute based encryption [60], [61], functional encryption [62] and the more recent controlled functional encryption [63] and access control encryption [64] 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.

C. Private identifiers

Private identifiers are used to obtain the common identifiers as described in Section V-B. The requirement for the private identifiers to be known by the agent, data provider and user is very strong, but ensures that no communication is needed between the user performing check and other parties. Candidates for private identifiers could be passport numbers, national insurance numbers, health service numbers or customer numbers but these are clearly very low entropy and cannot be expected to be very private. Ideally, specific identifiers could be supplied in the context of digital identity services that have been discussed in various countries.

D. Delay requirements

In some cases, there may be a conflict between the desire to offer publicly verifiable information and the need for secrecy in, for example, an ongoing law enforcement operation where it be preferable that a user does not realize his data is being requested. In such cases, replacing the request on the ledger with a commitment to be opened later may be appropriate, as the commitment would be able to later show that the request was correctly performed at the time.
VIII. CONCLUSION

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

ACKNOWLEDGMENTS

The authors would like to thank Jonathan Bootle for the helpful discussions and suggestions. Alexander Hicks is supported by OneSpan and UCL through an EPSRC Research Studentship, Vasileslav Mavrodiev 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 a Google Faculty Award, and Steven Murdoch is supported by The Royal Society [grant number UF160505].

REFERENCES

[1] 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. [Online]. Available: https://www.gov.uk/government/publications/investigatory-powers-bill-overarching-documents

[2] Interception of Communications Commissioner’s Office. (2017) Report of the interception of communications commissioner - annual report for 2016. [Online]. Available: https://assets.publishing.service.gov.uk/government/uploads/system/uploads/attachment_data/file/670219/IOCCO_annual_report_2016_PDF

[3] ETSI. (2018) TS 103 307: Security aspects for LI and RD interfaces. V1.3.1. [Online]. Available: https://www.etsi.org/deliver/etsi_ts/103300/103307/01.03.01../ts_103307v010301p.pdf

[4] A. Narayanan and V. Shmatikov, “Robust de-anonymization of large sparse datasets,” in Security and Privacy, 2008. SP 2008. IEEE Symposium on. IEEE, 2008, pp. 111–125.

[5] L. Sweeney, “k-anonymity: A model for protecting privacy,” International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems, vol. 10, no. 05, pp. 557–570, 2002.

[6] N. Li, T. Li, and S. Venkatasubramanian, “c-closeness: Privacy beyond k-anonymity and l-diversity,” in Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on. IEEE, 2007, pp. 106–115.

[7] A. Machanavajjhala, J. Gehrke, D. Kifer, and M. Venkitasubramanian, “l-diversity: Privacy beyond k-anonymity,” in Data Engineering, 2006. ICDE ’06. Proceedings of the 22nd International Conference on. IEEE, 2006.

[8] J. Domingo-Ferrer and V. Torra, “A critique of k-anonymity and some of its enhancements,” in Availability, Reliability and Security, 2008.ARES’08. Third International Conference on. IEEE, 2008, pp. 990–993.

[9] R. Agrawal, T. Imielinski, and A. Swami, “Mining association rules between sets of items in large databases,” in ACM SIGMOD Record, vol. 22, no. 2. ACM, 1993, pp. 207–216.

[10] A. Evfimievski, R. Srikant, R. Agrawal, and J. Gehrke, “Privacy preserving mining of association rules,” Information Systems, vol. 29, no. 4, pp. 343–364, 2004.

[11] N. Zhang, S. Wang, and W. Zhao, “A new scheme on privacy preserving association rule mining,” in European Conference on Principles of Data Mining and Knowledge Discovery. Springer, 2004, pp. 484–495.

[12] M. Walfish and A. J. Blumberg, “Verifying computations without reexecuting them,” Communications of the ACM, vol. 58, no. 2, pp. 74–84, 2015.

[13] E. Boyle, S. Goldwasser, and I. Ivan, “Functional signatures and pseudorandom functions,” in International Workshop on Public Key Cryptography. Springer, 2014, pp. 501–519.

[14] M. Barbosa and P. Farshim, “Delegatable homomorphic encryption with applications to secure outsourcing of computation,” in Cryptographers’ Track at the RSA Conference. Springer, 2012, pp. 296–312.

[15] A. Narayan, A. Feldman, A. Papadimitriou, and A. Haeberlen, “Verifiable differential privacy,” in Proceedings of the Tenth European Conference on Computer Systems. ACM, 2015, p. 28.

[16] A. Haeberlen, B. C. Pierce, and A. Narayan, “Differential privacy under fire,” in 20th USENIX Security Symposium, San Francisco, CA, USA, August 8-12, 2011, Proceedings., 2011. [Online]. Available: http://static.usenix.org/events/sec11/full_papers/Haeberlen.pdf

[17] D. Demirel, L. Schabbiuser, and J. Buchmann, Privately and Publicly Verifiable Computing Techniques: A Survey. Springer, 2017.

[18] S. A. Crosby and D. S. Wallach, “Efficient data structures for tamper-evident logging,” in USENIX Security Symposium, 2009, pp. 317–334.

[19] A. Bates, K. R. Butler, M. Sherr, C. Shields, P. Traynor, and D. Wallach, “Accountable wiretapping—or i know they can hear you now,” Journal of Computer Security, vol. 23, no. 2, pp. 167–195, 2015.

[20] S. Goldwasser and S. Park, “Public accountability vs. secret laws: Can they coexist? A cryptographic proposal,” in Proceedings of the 2017 Workshop on Privacy in the Electronic Society. ACM, 2017, pp. 99–110.

[21] J. Frankie, S. Park, D. Shaar, S. Goldwasser, and D. J. Weitzen, “Practical accountability of secret processes,” Cryptology ePrint Archive, Report 2018/697, 2018, https://eprint.iacr.org/2018/697.

[22] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. De Caro, D. Eneyart, C. Ferris, G. Laventman, Y. Manevich, S. Muralidharan, C. Murthy, B. Nguyen, M. Sethi, G. Singh, K. Smith, A. Sorniotti, C. Stathakopoulos, M. Vukolic, S. Weed Coco, and J. Yellick, “Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains,” ArXiv e-prints, Jan. 2018.

[23] C. Cachin, “Architecture of the Hyperledger blockchain fabric,” in Workshop on Distributed Cryptocurrencies and Consensus Ledgers, 2016.

[24] M. Vukolic, “Rethinking permissioned blockchains,” in Proceedings of the ACM Workshop on Blockchain, Cryptocurrencies and Contracts. ACM, 2017, pp. 3–7.

[25] Google. (2017) Trillian. [Online]. Available: https://github.com/google/trillian

[26] B. Laurie, A. Langley, and E. Kasper, “Rfc 6962 – Certificare transparency,” https://tools.ietf.org/html/rfc6962, Tech. Rep., 2013.

[27] B. L. Adam Eijdenberg and A. Cutter. (2017) Trillian – verifiable data structures. [Online]. Available: https://github.com/google/trillian/blob/master/docs/VerifiableDataStructures.pdf

[28] B. Laurie and E. Kasper, “Revocation transparency,” Google Research, September, 2012.

[29] R. L. Rivest, “The ThreeBallot voting system,” 2006.

[30] R. L. Rivest and W. D. Smith, “Three voting protocols: VBallot, VAY, and Twin,” USENIX/ACCURATE Electronic Voting Technology (EVT 2007), 2007.

[31] H. Jones, J. Jiang, and G. Belote, “ThreeBallot in the field,” Term paper for MIT course, vol. 6, 2006.

[32] A. W. Appel, “How to defeat Rivest’s ThreeBallot voting system,” Manuscript, pazdziernik, 2006.

[33] K. Henry, D. R. Stinson, and J. Sui, “The effectiveness of receipt-based attacks on ThreeBallot,” IEEE Transactions on Information Forensics and Security, vol. 4, no. 4, pp. 699–707, 2009.

[34] J. Chichor, M. Kutylyoski and B. Wglorz,” “Short ballot assumption and ThreeBallot voting protocol,” in International Conference on Current Trends in Theory and Practice of Computer Science. Springer, 2008, pp. 585–598.

[35] C. E. Strauss, “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, vol. 74, 2006.

[36] C. Strauss, “The trouble with triples: A critical review of the triple ballot (3Ballot) scheme, part 1,” Verified Voting New Mexico, 2006.

[37] K. West and A. Gervais, “Do you need a blockchain?” IACR Cryptology ePrint Archive, vol. 2017, p. 375, 2017.

[38] K. Moriarty, B. Kaliski, and A. Rusch, “Pcks# 5: Password-based cryptography specification version 2.1,” 2017.
[39] B. Dowling, F. Günther, U. Herath, and D. Stebila, “Secure logging schemes and certificate transparency,” in ESORICS 2016, Part II, ser. LNCS, I. G. AsokASYLAKIS, S. Ioannidis, S. K. Katsikas, and C. A. Meadows, Eds., vol. 9879. Heraklion, Greece: Springer, Heidelberg, Germany, Sep. 26–30, 2016, pp. 140–158.

[40] M. Chase and S. Meiklejohn, “Transparency overlays and applications,” in ACM CCS 16, E. R. Weippl, S. Katzenbeisser, C. Kruegel, A. C. Myers, and S. Halevi, Eds. Vienna, Austria: ACM Press, Oct. 24–28, 2016, pp. 168–179.

[41] A. . D. P. W. Party, “Opinion 05/2014 on anonymisation techniques,” 2014.

[42] J. Sousa, A. Bessani, and M. Vukolić, “A byzantine fault-tolerant ordering service for the hyperledger fabric blockchain platform,” arXiv preprint arXiv:1709.06921, 2017.

[43] P. Yee, “Updates to the internet x. 509 public key infrastructure certificate and certificate revocation list (CRL) profile,” 2013.

[44] J. Heaton, “Comparing dataset characteristics that favor the apriori, eclat or fp-growth frequent itemset mining algorithms,” in SoutheastCon 2016, March 2016, pp. 1–7.

[45] A. Nagar, M. Hahsler, and H. Al-Mubaid, “Association rule mining of gene ontology and protein networks: Promises and challenges.” in Proceedings of the 13th ACM conference on Computer and communications security. ACM, 2006, pp. 89–98.

[46] A. Dekhtyar and J. Verburg, “Extended bakery dataset,” https://wiki.csc.calpoly.edu/datasets/wiki/ExtendedBakery, 2009.

[47] F. Flouvat, F. De March, and J.-M. Petit, “A thorough experimental study of datasets for frequent itemsets,” in Data Mining, Fifth IEEE International Conference on. IEEE, 2005.

[48] A. Toti, R. Vilalta, P. Lindner, B. Lefer, C. Macias, and D. Price, “Analysis of correlation between pediatric asthma exacerbation and exposure to pollutant mixtures with association rule mining,” Artificial intelligence in medicine, vol. 74, pp. 44–52, 2016.

[49] P. B. Jensen, L. J. Jensen, and S. Brunak, “Mining electronic health records: towards better research applications and clinical care,” Nature Reviews Genetics, vol. 13, no. 6, p. 395, 2012.

[50] V. Goyal, O. Pandey, A. Sahai, and B. Waters, “Attribute-based encryption for fine-grained access control of encrypted data,” in Proceedings of the 13th ACM conference on Computer and communications security. ACM, 2006, pp. 89–98.

[51] D. Boneh and M. Franklin, “Identity-based encryption from the weil pairing,” in Annual international cryptology conference. Springer, 2001, pp. 213–229.

[52] D. Boneh, A. Sahai, and B. Waters, “Functional encryption: Definitions and challenges,” in Theory of Cryptography Conference. Springer, 2011, pp. 253–273.

[53] A. Shamir, “Identity-based cryptosystems and signature schemes,” in Workshop on the theory and application of cryptographic techniques. Springer, 1984, pp. 47–53.

[54] D. Boneh and M. Franklin, “Identity-based encryption from the weil pairing,” in Annual international cryptology conference. Springer, 2001, pp. 213–229.

[55] V. Goyal, O. Pandey, A. Sahai, and B. Waters, “Attribute-based encryption for fine-grained access control of encrypted data,” in Proceedings of the 13th ACM conference on Computer and communications security. ACM, 2006, pp. 89–98.

[56] J. Bethencourt, A. Sahai, and B. Waters, “Ciphertext-policy attribute-based encryption,” in Security and Privacy, 2007. SP'07. IEEE Symposium on. IEEE, 2007, pp. 321–334.

[57] D. Boneh, A. Sahai, and B. Waters, “Functional encryption: Definitions and challenges,” in Theory of Cryptography Conference. Springer, 2011, pp. 253–273.

[58] M. Naveed, S. Agrawal, M. Prabhakaran, X. Wang, E. Ayday, J.-P. Hubaux, and C. Gunter, “Controlled functional encryption,” in Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security. ACM, 2014, pp. 1280–1291.

[59] I. Damgård, H. Haagh, and C. Orlandi, “Access control encryption: Enforcing information flow with cryptography,” in Theory of Cryptography Conference. Springer, 2016, pp. 547–567.