SAMPL: Scalable Auditability of Monitoring Processes using Public Ledgers

Gaurav Panwar, Roopa Vishwanathan, Satyajayant Misra, Austin Bos
New Mexico State University
Las Cruces, New Mexico
{gpanwar,roopav,misra,abos}@nmsu.edu

ABSTRACT

Organized surveillance, especially by governments poses a major challenge to individual privacy, due to the resources governments have at their disposal, and the possibility of overreach. Given the impact of invasive monitoring, in most democratic countries, government surveillance is, in theory, monitored and subject to public oversight to guard against violations. In practice, there is a difficult fine balance between safeguarding individual's privacy rights and not diluting the efficacy of national security investigations, as exemplified by reports on government surveillance programs that have caused public controversy, and have been challenged by civil and privacy rights organizations.

Surveillance is generally conducted through a mechanism where federal agencies obtain a warrant from a federal or state judge (e.g., the US FISA court, Supreme Court in Canada) to subpoena a company or service-provider (e.g., Google, Microsoft) for their customers' data. The courts provide annual statistics on the requests (accepted, rejected), while the companies provide annual transparency reports for public auditing. However, in practice, the statistical information provided by the courts and companies is at a very high level, generic, is released after-the-fact, and is inadequate for auditing the operations. Often this is attributed to the lack of scalable mechanisms for reporting and transparent auditing.

In this paper, we present SAMPL, a novel auditing framework which leverages cryptographic mechanisms, such as zero knowledge proofs, Pedersen commitments, Merkle trees, and public ledgers to create a scalable mechanism for auditing electronic surveillance processes involving multiple actors. SAMPL is the first framework that can identify the actors (e.g., agencies and companies) that violate the purview of the court orders. We experimentally demonstrate the scalability for SAMPL for handling concurrent monitoring processes without undermining their secrecy and auditability.

CCS CONCEPTS

• **Security and privacy** → *Distributed systems security*; *Security protocols*; *Social aspects of security and privacy*; *Privacy protections*; • **Social and professional topics** → *Governmental regulations*; • **Applied computing** → *Law*.

CCS '19, November 11–15, 2019, London, United Kingdom

ACM ISBN 978-1-4503-6747-9/19/11...$15.00
https://doi.org/10.1145/3319535.3354219

ACM Reference Format:
Gaurav Panwar, Roopa Vishwanathan, Satyajayant Misra, Austin Bos. 2019. SAMPL: Scalable Auditability of Monitoring Processes using Public Ledgers. In *2019 ACM SIGSAC Conference on Computer and Communications Security (CCS '19), November 11–15, 2019, London, United Kingdom.* ACM, New York, NY, USA, 18 pages. https://doi.org/10.1145/3319535.3354219

1 INTRODUCTION

With increases in connected devices and electronic communications becoming the mainstay of human interactions, monitoring of human electronic activities have become pervasive both by companies trying to use the information for business advantage and governments trying to surveil citizens for national security and criminal activities [31]. Organized surveillance, particularly by state actors poses a serious challenge to an individual's privacy on account of the resources at disposal and its potential for overreaching use [11, 31]. Further, individual or representative entities do not have a mechanism to audit the surveillance, even after it's completion, to assess if their rights were violated.

To motivate the discussion, we use the well-known United States Surveillance law, namely Electronic Communications Privacy Act (ECPA), it's amendments, and it's corresponding processes as an example. Similar laws exist in other countries, e.g., the Investigatory Powers Act in the UK, and the Telecommunications (Interception and Access) Act in Australia. Several studies have shown that said processes, although technically auditable, tend to be opaque and seldom fully auditable, even when the audit is performed by powerful oversight bodies, such as the US Congress [31, 40].

In these monitoring processes, the active players include the law enforcement/intelligence gathering agency (L) that makes the surveillance request; the judge/court (J) that grants the requests; and the company (C) that provides the data corresponding to the request. The other actors include the individual (I) being surveilled and other users/agencies, e.g., American Civil Liberties Union (ACLU) [6] whose mission is to defend and safeguard individual privacy rights. The steps in the process generally start with the agency L requesting a court order from the judge J. If J approves the request, she creates a sealed court order, which can only be unsealed by L for the company C; the sealed order can be unsealed for the public after a pre-defined time (set during the issue of the order). The company C either accepts the request and provides the data or challenges the order on perceived violations. Once all parties agree, C sends the data requested. The agency L and company C can iteratively request and transmit data respectively several times, as needed, within the purview of the order.

Challenges and Motivation: The said monitoring processes present several issues that hinder accountability and public auditability, that are desirable for transparency: **1)** The fact that there exists a sealed order is not publicly notified. **2)** Further, as per studies [35], there is no systematic mechanism to unseal orders. In the absence of information, there is no way for the public to even know if there is any order, let alone request its unsealing when the sealing date expires. Note that an order not getting unsealed might not necessarily mean the judge issuing the order is malicious, rather, the judge might simply forget to unseal the order at the right time. **3)** An important missing piece in all accountability mechanisms today is that there is no way to make sure that exchanges happening between L and C, at the time of the surveillance, followed the letter and spirit of the sealed order (enabling an auditable trail). **4)** The scalability of the processes given the number of requests (around 16K to 33K, as discussed below) and the frequency of exchanges between/among the parties has not been explored.

Currently the only information that is publicly available is summarized information from the courts themselves or from annual aggregate reporting by companies [21, 27]. For instance, the FISA court rulings present the number of requests made under different sections, the number fully or partially granted, and the number denied. For example in 2018, 1204 requests were submitted for Sections U.S.C. 50 §1805 and §1804, with 868 granted, 308 modified, 41 partly denied, and 18 completely denied. However, this information usually tends to be high level aggregate statistics, and are not useful for public accountability. It does not equip individuals being surveilled with the means to determine if any of the players involved (law enforcement agencies, companies) reached beyond the ambit of the court's order, or if they were unfairly surveilled, e.g., wholesale or dragnet surveillance.

As a result, the exchanges and dealings between governments conducting surveillance, citizens being surveilled, and non-profit privacy advocates and organizations, are uneasy at best, and pugnacious at worst. This is evidenced in the steady stream of lawsuits challenging the constitutionality of various government surveillance programs, raising pertinent questions about the legality and ethics of the surveillance itself, and if citizens' privacy and constitutional rights were violated [34, 41, 42].

Google's transparency report [27] states the number of user data and account requests made over a six-month period and the proportion of requests under each category (such as subpoena and search warrants). Notable is the fact that the number of requests to Google have been rising steadily for the last five years, e.g., in the US, 16,407 user data requests for roughly 31,072 user accounts for year 2012, to 32,877 user data requests corresponding to roughly 68,456 user accounts in 2017.

For the first months of 2018 (the last reported data), there were 20,936 user requests for approximately 62,142 user accounts. Similar reports are also available from other companies, such as Microsoft and Facebook [19, 30]. According to our findings, frequently, the information presented is scarce and there are neither well-defined mechanisms to audit surveillance processes from the outset, nor to enable the surveilled individual the capability to assess post-completion of the surveillance whether the search violated their privacy rights, e.g., the right of citizens to be secure against unreasonable searches and seizures, per the US Constitution's Fourth Amendment.

Contributions: In this paper, we propose our framework, SAMPL that addresses the challenges mentioned above. Our novel contributions include: **i)** Design of SAMPL: a generic and scalable framework for accountability of monitoring processes. **ii)** Capability for auditing the compliance of the entities over the lifetime of the surveillance order, from the outset, using cryptographic techniques, such as zero knowledge proofs (ZKPs), and Pedersen commitments. We introduce an entity called *Enforcer* who serves as the conduit for interactions between law enforcement/intelligence gathering agencies and companies, and verifies their interactions to guarantee compliance. We qualitatively prove that auditability of the surveillance process when the court order is active is only possible if an entity like our proposed enforcer serves as the conduit for information and the process does not leak information about the surveillance to the public, just provides audit insights. **iii)** A case study of our system in the context of the US legal system. **iv)** Security analysis of the the proposed framework. **v)** Validation of the framework using a near-real world implementation to assess scalability.

Outline: In Section 2, we review related work. In Sections 3 and 4, we present the system model, and threat model and privacy/security properties, respectively. In Section 5, we present our construction for SAMPL; in Section 6 we discuss SAMPL in the context of the US legal system; and in Section 7, we present the security analysis for SAMPL. In Section 8, we present our implementation of the framework and our evaluations to demonstrate both feasibility and scalability. In Section 9, we discuss possible enhancements, extensions and generalizations of SAMPL. For better readability, we give the proof of security of SAMPL in the appendix.

2 RELATED WORK

Our related work falls into three broad categories: auditing and access control mechanisms, dragnet surveillance, and surveillance with accountability. We review each of these below.

Auditing and access control mechanisms: Goldwasser and Park [25] proposed cryptographic mechanisms involving ZKPs and commitments to provide auditability in the application of *secret laws*. For example, the U.S. Foreign Intelligence Surveillance Act (FISA) court operations are classified, and the court typically hears arguments only from government agencies [22]. While the focus of [25] was on providing the public *auditable* records that secret laws were correctly applied by courts, our focus is on verifying whether the interactions between the law enforcement agencies and companies, follow the letter and spirit of a court's order.

Bates *et al.* [8] proposed mechanisms to enable secure audits of wiretapping systems. Kroll *et al.* [28] designed a way for entities such as companies, law enforcement/intelligence-gathering agencies to prove using cryptographic techniques that they are authorized to access data such as phone records and email data. These works focus on providing auditability using encrypted audit logs that are not accessible to the general public, whereas our goal is to focus on public accountability. Kamara [29] proposed a mechanism for federal agencies to carry out warranted tapping on phones of users, which focuses on providing access-control, not public accountability.

Dragnet surveillance: Segal *et al.* [38, 39] focused on building mechanisms to avoid *contact chaining*, where a large number of users get pulled into a surveillance net, chiefly because they were associated with a legitimate target of surveillance. Their accountability mechanism ensures that government agencies can safely disclose statistics such as number of warrants per month and maximum number of individuals affected per warrant.

Table 1: Notations

Variable	Definition
λ	Security parameter
$J, L, C, E, I, \mathbb{U}, \mathbb{I}$	Judge, Law enforcement agency, Company, Enforcers Set, Individual, Set of Users, Set of Individuals
σ	Signature
T	I's total data records
$\mathsf{RI} = (VK_{\mathsf{RI}}, SK_{\mathsf{RI}})$	Real identity of individual I
$\mathsf{AI} = (VK_{\mathsf{AI}}, SK_{\mathsf{AI}})$	Anonymized identity of individual I
$\mathsf{PI} = (VK_{\mathsf{PI}_1}, SK_{\mathsf{PI}_1}), \ldots, (VK_{\mathsf{PI}_m}, SK_{\mathsf{PI}_m})$	Pseudonymous identities of individual I
K_{CI}	Key shared between company C and individual I
K_{JLC}	Key shared between J, L, C
K_{EJLC}	Key shared between E, J, L, C
$\mathcal{C}$	Ciphertext
$bSize$	Batch Size for a client
$bNum$	Batch number for a specific client message
π_{PI_i}	ZKP that PI_i is valid pseudonym of individual I
SO	Surveillance order
IO	Intermediate order
ι	time period for surveillance
Verify()	Verification function
ZKPVerify()	ZKP Verification function
Jdecide()	Judge decision function
Ldecide()	Law enforcement agency decision function
Cdecide()	Company decision function
OrderGen()	Judge order generating function
SR	Law enforcement agency's surveillance request
SRR	Company's surveillance request response
$\|\|$	Concatenation operator
BC()	Blockchain
BC.read()	Blockchain read function
BC.write()	Blockchain write function

Surveillance: Frankle *et al.* [24], proposed a system which deals with accountability in secret processes, which is most relevant to our work. There are two significant differences between [24] and our work: 1) [24] requires law enforcement agencies and companies to post cryptographic commitments and ZKPs to the blockchain at regular intervals. Moreover, in their system, honest parties are trusted to log information regularly, and honest parties are expected to *report* dishonest logging whenever they see it. There are two problems with this: firstly, government agencies and companies might be forgetful, and cannot be trusted to post information regularly to a public ledger. Secondly, companies might be loath to report possibly dishonest logging by law enforcement agencies when they see it, fearing retribution.[1]

We remove this requirement by introducing an independent auditor, called Enforcer (E), who can keep both, the company and the law enforcement agency in check. 2) In [24], ZKPs created by an agency and/or company are basically a proof that they are aware of the court's surveillance order. A ZKP is a proof of knowledge, not compliance; merely proving *knowledge* of the contents of a court's orders does not guarantee that the agency/company are *complying* with the court's orders. In our system, the Enforcer explicitly verifies that the data requested by the law enforcement agency, and given by the company are within the ambit of the court's surveillance order. This is done in a privacy-preserving manner such that the Enforcer does not actually get to know the user's data (e.g., emails), but is able to verify that the agency is not over-requesting data, and the company is not over-sharing data.

3 SYSTEM MODEL

Parties: In our system, there are six parties: the individual being surveilled I, company C that I has an account (e.g., e-mail) with, law enforcement/intelligence gathering agency L requesting the surveillance, Judge J who can potentially issue the surveillance order on I, and an Enforcer E, who enforces accountability of L and C's operations, by ensuring that L does not request more information about I than what is authorized by J, and C does not over-share information about I, more than what is authorized by J. Finally our system has a set of interested users, $\mathbb{U}$, made up of civil-rights and/or non-profit organizations (e.g., American Civil Liberties Union (ACLU)) whose mission is to protect and preserve individuals' privacy as defined by laws. We assume that all communication between J, L, C, E, I, and $\mathbb{U}$ takes place via secure and authenticated channels. They use each other's public and verification keys, respectively to encrypt and authenticate all communication between them.

We note that $I \subset \mathbb{I}$, where $\mathbb{I}$ is a set of individuals who have an account with C. Our table of notations is given in Table 1.

Identities of an individual I: In our system, an individual I has three identities associated with her:

A real identity, RI which may correspond to I's e-mail address that is being surveilled. RI is established between I and C when I signs up for service with C. In our system RI is represented by a verification/signing key-pair: $\mathsf{RI} = (VK_{\mathsf{RI}}, SK_{\mathsf{RI}})$. The company C stores VK_{RI} and VK_{RI} is known only to J, L, and C. In particular, RI will not be known to E. We assume that RI is stored safely by I, does not get compromised, and acts as the root-of-trust for all other keys involving I.

An anonymized identity, AI, which corresponds to a nickname associated with RI. When a user signs up for service with a company, they are asked to create the anonymized identity AI which is linked by C to their real identity RI. The user can create only one anonymous identity with a service provider (e.g., one nickname per

[1] A report issued by the US Department of Justice's OIG [2] says that they found company employees provided telephone records to the FBI in response to just verbal and e-mail requests, without legal process or even exigent letters, since they (company employees) believed the requests related to major FBI counterterrorism investigations.

e-mail address). We represent AI by a keypair: $\mathsf{AI} = (VK_{\mathsf{AI}}, SK_{\mathsf{AI}})$. We use anonymized identities to avoid having the enforcer know RI. The company C stores VK_{AI} which is known and revealed to E, J, L, and C during the surveillance period.

A pseudonymous identity, $\mathsf{PI}_i; i \in [1..m]$, represented by $\mathsf{PI} = (VK_{\mathsf{PI}_i}, SK_{\mathsf{PI}_i})$ which corresponds to I's pseudonym associated with AI. The pseudonymous identity can be chosen from a set of m identities, with the restriction that only one pseudonymous identity can be active at any given point of time, and a pseudonymous identity cannot be reused. Pseudonymous identities, as opposed to real and anonymized identities, are transient key-pairs. VK_{PI} is known and revealed to E, J, L, and C. The company stores all historical VK_{PI}s for future verification.

An individual storing data on company servers: *Although* SAMPL *enables the auditing of a broad range of data and application types, for illustration in this paper we use user emails. In Section 9, we generalize this requirement.* When an individual I signs up for service with a company C, it interactively creates a symmetric key K_{CI} to be shared between C and I. I uses K_{CI} to encrypt sensitive information, but keeps the date and time as plaintext and signs the whole message. K_{CI} can be updated periodically. C and I agree on two parameters, *bSize* and *bNum*, which denote *batch size* and *batch number*.

The batch size represents the intervals at which the user's messages are batched. The batch number indicates the batch a given message originates from. Let I's total data records, e.g., emails be denoted by T. Then $bNum = T/bSize$, *bSize* can be a static or dynamic parameter. In the static case, I sets up *bSize* at the time of service initiation with C, and doesn't change it; in the dynamic case, *bSize* can be changed by I as needed. SAMPL supports both these implementation choices.

I creates and encrypts each email with K_{CI} before sending it to C. At the end of each batch, C creates a Merkle tree with the hashes of all messages in the batch at the leaves. C sends the root hash of the Merkle tree to I. I verifies the root hash calculation, signs it if accepts, and sends it to C. All signatures contain a timestamp which has sign date and time. C then discards the Merkle tree and archives just the signed root hash, since C can create the Merkle tree on demand from the stored ciphertexts as needed.

Role of Enforcer, E: Each communication between L and C involves them independently passing the message to E for verification. Once E verifies that the message is not over-requesting or over-sharing data with respect to an approved court order, the message is passed on to the intended recipient (C or L). When surveillance data from C is approved by E and received by L, C sends the shared key, K_{CI} directly to L, who can can then decrypt the information and carry out the investigation.

We envision the enforcer to be a government watchdog or organization that oversees adherence to laws and rights by private companies and law enforcement agencies. Federal agencies have their own oversight entities, e.g., FBI is audited by the Department of Justice's Office of the Inspector General (OIG). Other federal agencies also have their corresponding auditing entities. These entities currently do auditing when needed, and hence the auditing always happens after the event. We propose that the OIG plays a *proactive* role in auditing such process, and enforce accountability from the beginning, rather than play a *reactive* role and issue review and audit reports after-the-fact, as it currently does.

Blockchain and its operations: The blockchain, BC, is used as an official record for verification of actions performed, we use it as an off-the-shelf enabling technology. When forwarding a request, each entity posts a signed hash of the request/response to the blockchain–a transaction–all messages posted on the BC are signed. The BC also serves as a platform to announce new cases to the public watch dogs and the general public without divulging investigation details. The miners ensure that only valid entities involved in an investigation can post transactions to the BC. We envision the implementation of SAMPL using a permissioned blockchain with read-only access given to public. For efficiency and fast convergence, proof-of-stake may be used as the distributed consensus mechanism. The infrastructure required for the BC may be maintained and managed by the judicial system to engender greater trust.

4 THREAT MODEL

We list trust assumptions on the parties in the system:

Judge J: The judge J is assumed to be honest, but forgetful, i.e., J might forget to unseal records at the right time. J is trusted to correctly generate an Surveillance Order (*SO*) and place it on the BC. Whenever *SO*'s seal expires, members of $\mathbb{U}$ can choose to contact J to make public the contents of *SO*. $\mathbb{U}$ can then verify details of case, including contacting I as needed.

Law enforcement agency L: L is assumed to be malicious, in that L will try to over-request data beyond what is authorized by the *SO* issued by J (we discuss some overreaches in the real world in Section 6.2). Once the *SO* is posted by J on the blockchain, L will contact E with a surveillance request (*SR*). *SR* will be checked and ratified by E based on the *SO* and prevalent policies.

Company C: C is assumed to be malicious, in that C can over-share data beyond what is sought by the *SR*, and authorized by J's *SO*. If C fails to respond to an *SR* with a surveillance request response (*SRR*), then there are policy measures that can be exercised by J to enforce compliance.

Enforcer E: E verifies each *SR* generated by L and also verifies each *SRR* generated by C, respectively. We assume E is honest. The enforcer only knows I's anonymized identity, AI and pseudonymous identity, PI. In particular, E is not privy to I's real identity RI. E also does not have access to the plaintext version of I's records stored with C (e.g., emails). When a certain threshold of failures on the part of L, C is reached (which can be a implementation specific system parameter), E can choose to contact J and post a message to BC exposing identity of the faulty party. The enforcer does not store information linking AI and PI after evaluating an *SRR*.

No collusion assumption: In our system, we assume L and C do not directly communicate with each other, and go through E for information exchanges. We believe if L and C break this protocol and interact directly, auditable data structures [26] may be used to verify the operations of C. However, out of band, unbridled data exchange is difficult to prevent when both parties are complicit. Nevertheless, for accountability, it is in L's and C's interest to act according to due process, and go through E, and not collude.

4.1 Privacy and security properties

SAMPL provides the following privacy and security properties:

Accountability for L and C: We ensure that a malicious L and/or C cannot over-request, or over-share data, respectively, beyond that authorized by the *SO*, as long as they do not bypass the entire system, and collude via side-channels. This applies to both: over-requesting/over-sharing of the surveilled user's data, or data belonging to users not listed in the *SO* (not under surveillance).
Forgetful J: Our system enables an independent set of users, $\mathbb{U}$ (e.g., non-profit organizations such as ACLU) who keep track of court-order unsealing dates, to contact the courts to unseal non-sensitive information, contact the individuals who were being surveilled, and help them with further courses of action.
Security against malicious I and C: We ensure that a malicious I cannot make C fail E's queries by creating fake ZKP for their real, anonymous and pseudonymous identities. Also, a malicious C cannot create fake data for I and frame I.

We now give the computational assumption for our system.

Definition 4.1. (DDH Problem [10]) We say that the DDH problem is hard relative to $\mathcal{G}$ if for all PPT algorithms A, there is a negligible function negl such that

$$|Pr[A(\mathbb{G}, q, g, g^x, g^y, g^z) = 1] \\ -Pr[A(\mathbb{G}, q, g, g^x, g^y, g^{xy}) = 1]| \leq \mathsf{negl}(\lambda)$$

where in each case the probabilities are taken over the experiment in which $\mathcal{G}(1^\lambda)$ outputs $(\mathbb{G}, q, g)$, and then uniform $x, y, z \in \mathbb{Z}_q$ are chosen.

5 DESCRIPTION OF SAMPL

As a pre-requisite to using SAMPL for surveillance, I and C interact to setup keys, associated ZKPs, and other operations as outlined in Section 5.1. Surveillance on user I's data is carried out with interactions between J, L, C, and E as described in Section 5.2. We note that SAMPL has 7 protocols and 4 algorithms. We adopt the convention that communication protocols are run between two or more entities, and algorithms are computations done by a single entity. We recall that per our system model we assume that all communication between entities takes place over secure and authenticated channels.

5.1 Pre-Requisite for SAMPL

Protocols 1 and 2 bootstrap the communication between C and I, and the corresponding data exchange. These protocols are required so that in case of surveillance request by L for I's data, E can verify the user data without gaining any knowledge about identity of I.

Protocol 1: This is run by an individual I the first time she sets up an email account with company C. In Line 1, I does two 3-round ZKPs with C to prove that: 1) VK_{AI} was produced by someone who has knowledge of SK_{RI}, and 2) VK_{PI_i} was generated by someone who has knowledge of SK_{AI} (if C accepts VK_{AI} as valid). At the end, C will receive from I a copy of $VK_{\mathsf{AI}}, VK_{\mathsf{PI}_i}$ and their associated ZKPs, $\pi_{\mathsf{AI}}, \pi_{\mathsf{PI}_i}$, and signed copies of the ZKPs: $\sigma_{\mathsf{AI}} = Sign_{SK_{\mathsf{RI}}}(\pi_{\mathsf{AI}}), \sigma_{\mathsf{PI}_i} = Sign_{SK_{\mathsf{AI}}}(\pi_{\mathsf{PI}_i})$, along with some public verification metadata, *zkpVerf*, which will be used by the Enforcer for verifying the ZKPs. The proofs are Chaum-Pedersen-style interactive ZKPs [16], which can be made non-interactive using the Fiat-Shamir transform [20]. *Since the ZKPs are essentially used as black-boxes, in order not to distract the reader with their details, we give the ZKPs and their description in Appendix A.*

Protocol 1: Setup run between C and I.

Input : Public parameters: Group $\mathbb{G}$, $q = |\mathbb{G}|$, $g, h \in \mathbb{G}$.
Output : I establishes RI, AI, PI_i, *bSize* and K_{CI} with C.
Parties : C and I.

1 User I sets up $(VK_{\mathsf{AI}}, SK_{\mathsf{AI}})$ and $(VK_{\mathsf{PI}_i}, SK_{\mathsf{PI}_i})$, and sends the ZKPs, their verification metadata, and signatures on the ZKPs: $(\pi_{\mathsf{AI}}, \pi_{\mathsf{PI}_i})$, *zkpVerf*, and $(\sigma_{\mathsf{AI}}, \sigma_{\mathsf{PI}_i})$, respectively, to C.
2 User I sets up a shared key with C, K_{CI}, used to encrypt I's data stored on C's servers.
3 C and I agree upon and setup a batch-size, $bSize \in \mathbb{Z}^+$.

Next, I and C setup a shared key K_{CI} using which I's emails stored on C's servers are encrypted. I and C also agree upon a batch-size *bSize*, which denotes the message-intervals at which I's emails will be batched, e.g., after every 100 emails. C will batch all of I's emails at *bSize* intervals and create a Merkle hash tree for the batch with the hashes of the emails at the leaves; I will verify and sign the root of the tree.

Protocol 2: Exchange of data between C and I for a given batch.

Input : Public parameters: *bSize*, $bNum \in [1..maxbNum]$.
Output : C stores I's emails along with verification hashes.
Parties : C and I.

1 Let $\mathbb{M}_{bNum}$ represent the set of all e-mail messages in *bNum*.
2 **for** *each* $M_x \in \mathbb{M}_{bNum}$, $x \in [1..bSize]$ **do**
3   I encrypts M_x: $\mathcal{C}_x \leftarrow K_{CI}(M_x)$, sends $\mathcal{C}_x$ to C.
4   C stores $\mathcal{C}_x$.
  end
5 `/* At the end of batch` *bNum* `of` *bSize* `messages: */`
  Let $\mathbb{C}_{bNum}$ represent the set of all ciphertexts in *bNum*.
6 **begin**
7   C generates hashes, $H_x = H(\mathcal{C}_x)$, for all the $\mathcal{C}_x$ received from I.
8   C forms a Merkle tree $\mathbf{M}_{bNum}$, with the H_xs at the leaves, and R_{bNum} as root of the Merkle tree.
9   C sends $\mathbf{M}_{bNum}$ and R_{bNum} to I.
10   I verifies that the root hash (R_{bNum}) of $\mathbf{M}_{bNum}$ is correctly computed:
      10.1 If verification fails, I notifies C to retry.
      10.2 Else, I signs R_{bNum}: $\sigma_{R_{bNum}} \leftarrow Sign_{SK_{\mathsf{PI}_i}}(R_{bNum})$, sends $\sigma_{R_{bNum}}$ to C and deletes all local copies of M_x.
11   C stores $\sigma_{R_{bNum}}$ along with previously stored $\mathcal{C}_x$'s for batch *bNum*.
  end

Protocol 2: Protocol 2 depicts I's emails being stored on C's servers. Before I and C execute this algorithm, they would have already run Protocol 1 to setup the symmetric key K_{CI}. I creates an email message M_x and encrypts it with K_{CI}, generating $\mathcal{C}_x$, before forwarding it to C (Lines 2,3,4). This already happens in OpenSSL, where the connections (and data transmitted) between two communicating entities are encrypted using pairwise symmetric session keys.

Protocol 3: J issuing SO and posting SO on BC.

Input : $VK_{\text{AI}}, VK_{\text{PI}}$ of user I, $\mathbb{G}$, $g, h \in \mathbb{G}$, $q = |\mathbb{G}|$.

Output: J issues SO, sets up keys K_{JLC}, K_{EJLC} and transmits them to relevant parties.

Parties : E, J, L, and C.

1 L issues a request to J: $SR = (VK_{\text{RI}}||evidence)$.

2 J validates L's request. If "accept" $\leftarrow \mathcal{J}decide(SR)$, J generates $IO = (VK_{\text{RI}}||evidence)$ and gives to L.

3 L gives IO to C; C validates the IO. If "accept" $\leftarrow Cdecide(IO)$, C sends to J and L, $(VK_{\text{AI}}||\sigma_{\text{AI}}||\pi_{\text{AI}})$, given to C by I in Protocol 1.

4 J validates C's response, checks if "true" $\leftarrow Verify(VK_{\text{RI}}, \sigma_{\text{AI}}, \pi_{\text{AI}})$, and if "true" $\leftarrow ZKPVerify(VK_{\text{RI}}||VK_{\text{AI}}||\pi_{\text{AI}}||zkpVerf)$, and does the following:

4.1 Pick $K_{JLC} \leftarrow \{0,1\}^{\lambda}$, send to L and C. Pick $K_{EJLC} \leftarrow \{0,1\}^{\lambda}$, send to E, L, C. J also picks $r_2, r_3 \leftarrow \mathbb{Z}_q$, $g, h \in \mathbb{G}$, and generates Pedersen commitments: $Com_1 = (g^{K_{JLC}} h^{r_2})$, $Com_2 = (g^{K_{EJLC}} h^{r_3})$.

4.2 J creates $P1 = (VK_{\text{RI}}||evidence)$. $P1$ is encrypted with K_{JLC} and hence is accessible only to J, L and C. $P1$ is transmitted to L and C for verification and signatures.

4.3 J verifies the received signatures of L and C on $P1$, and embeds the signatures of J, L, C on $P1$: σ_{JP1},σ_{LP1}, and σ_{CP1} in $P2$, to preserve identity of L and C.

4.4 $P2$ contains VK_{AI}, start/end dates $\iota = [t_s, t_e]$, among other information, is encrypted with K_{EJLC}, and sent to L and C for verification and signatures $\sigma_{LP2}, \sigma_{CP2}$. $\sigma_{JP2}, \sigma_{LP2}, \sigma_{CP2}$ are then appended to the SO as $P3$.

4.5 Generates $SO \leftarrow OrderGen(VK_{\text{AI}}||VK_{\text{RI}}||evidence)$, which has format as described below:

$$SO = \langle \textbf{metadata}||\sigma_{\textbf{metadata}}||C_{P1}||C_{P2}||C_{P3}\rangle, \text{ where}$$
$$C_{P1} = E_{JLC}(P1)$$
$$C_{P2} = E_{EJLC}(P2)$$
$$C_{P3} = E_{EJLC}(\sigma_{JP2}||\sigma_{LP2}||\sigma_{CP2})$$

At the end of the current batch $bNum$, let $\mathbb{C}_{bNum}$ represent the set of all ciphertexts in $bNum$. C calculates hashes for all $\mathcal{C}_x \in \mathbb{C}_{bNum}$ and uses them as leaves to create a Merkle hash tree $\mathbf{M}_{bNum}$ (Lines 7,8). C sends $\mathbf{M}_{bNum}$ and the root hash (R_{bNum}) of the Merkle tree to I (Line 9). I verifies that R_{bNum} calculation is correct for the current batch. I signs the verified R_{bNum} and sends $\sigma_{R_{bNum}}$ to C (Line 10.2). I can then delete all the data stored locally since it is available for future retrieval from C. C stores $\sigma_{R_{bNum}}$ and discards the Merkle tree for the batch (Line 11). This construction helps reduce the space overhead significantly. This process is repeated for all future batches. If I found R_{bNum} to be wrongly calculated, then I does not sign R_{bNum} and C is contacted to reconstruct the Merkle tree and try again (Line 10.1).

5.2 Surveillance

The communication model under SAMPL can be divided into four phases, which we depict in Figure 1, and we give a high level idea of the phases in what follows.

Phase 1: Figure 1: Steps 1-7, are described in Protocol 3, and represents the first phase of SAMPL. It describes collection of information by J to validate the need for an SO, create, and post it to BC. This allows members of $\mathbb{U}$ to verify public data in SO for accountability of L and C, and allows L to conduct surveillance on data for I.

Phase 2: Figure 1: Steps 8-11, are described in Algorithm 4 and Algorithm 5, and represent the second phase of SAMPL. In Algorithm 4, L creates the SR corresponding to the SO created in **Phase 1**, and in Algorithm 5 we enforce accountability for L by having E verify the SR before sending it to C.

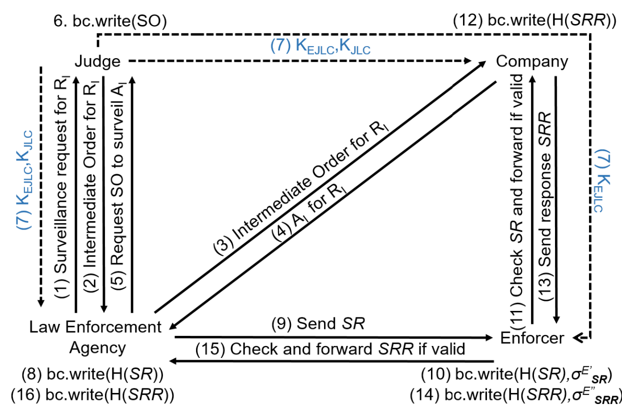


Figure 1: Workflow in SAMPL **(Dashed lines represent Key Exchange and solid lines represent regular communication).**

Phase 3: Figure 1: Steps 12-15, are described in Algorithm 6 and Algorithm 7, and represent the third phase of SAMPL. In Algorithm 6, C creates the SRR corresponding to the SR received in **Phase 2**, and in Algorithm 7 we enforce accountability for C by having E verify the SRR before sending it to L.

Phase 4: Figure 1: Step 16, is described in Protocol 8 and represents the fourth phase of SAMPL. In Protocol 8, L decrypts the user information and conducts the surveillance specified in the SO.

Protocol 3: Protocol 3 presents the interaction between J, L, C, and E, which culminates in J issuing a surveillance order SO, and setting up surveillance-related keys. In Line 1, L approaches J with evidence of suspicious behavior on the part of I which forms the surveillance request (SR). Here *evidence* represents the documented evidence supporting the SR. J has its own internal decision procedure, $\mathcal{J}decide$ using which it decides whether to accept or turn down the request (Line 2). If J decides to reject the request, L will have to return with an updated request $SR = (VK_{\text{RI}}||evidence')$, if it wants to persist with the request.

If the request SR is accepted, J generates an intermediate order IO, and gives it to L who forwards it to C. If C decides to comply (according to $Cdecide$), it retrieves VK_{AI} corresponding to VK_{RI}, sends VK_{AI} to L, along with π_{AI}, and σ_{AI} obtained in Protocol 1. L forwards this info to J (Line 3). If C decided to not comply with IO

(e.g., request violates statutory company policy), C would address the reasons to L prompting potential intervention from J, which is a judicial matter and out of scope of SAMPL. On receiving info from C, J independently verifies the ZKP associated with VK_{AI} and the signature on it (Line 4). If the verification fails, J notifies C and L, and exits. If the verification passes, J generates two symmetric keys: K_{JLC} meant to be shared between J, L, and C, and K_{EJLC} meant to be shared between E, J, L, and C. J then issues a surveillance order SO which is formatted as in Figure 2. The **metadata** may include case number, date of unsealing, and Pedersen commitments Com_1 and Com_2 to K_{JLC} and K_{EJLC}, respectively (Line 4.1), and any other information that can be made public about the case. The commitments are needed to hold J accountable. **Part 1** ($P1$) contains data meant to be shared between J, L, and C only, and includes VK_{RI} and the *evidence*. $P1$ is encrypted with K_{JLC} (Line 4.2), and the hash of the encrypted $P1$ is signed independently by J (σ_{JP1}), L (σ_{LP1}), and C (σ_{JP1}) (Line 4.3). These signatures are included inside **Part 2** ($P2$) along with VK_{AI}, start/end dates of surveillance (t_s, t_e) respectively[2]. $P2$ is encrypted with K_{EJLC}, before it is sent for verification and signing to J, L, and C which yield σ_{JP2}, σ_{LP2} and σ_{CP2}, respectively on successful verification. Before C signs hash of encrypted $P2$, it verifies that VK_{AI} contained in $P2$ corresponds to VK_{RI} contained in $P1$ that it had signed, i.e., σ_{CP1}.

These signatures are then verified by J, encrypted with K_{EJLC} and added to SO as part of **Part 3** ($P3$). The signatures are included in the encrypted text to preserve the identity of L and C from public until the SO is opened to public. Signatures on $\mathfrak{C}_{P1}$ and $\mathfrak{C}_{P2}$ are verified by E to hold J, L, and C accountable. The different kinds of SOs are discussed in Section 6.

Algorithm 4: This shows the surveillance request, SR created by L after J posts SO to the blockchain, BC. L creates an SR by creating a tuple with the start/end dates for the requested surveillance time interval, $\iota = [t_s, t_e]$ (Line 2). L includes the AI of the intended surveillance target, VK_{AI}. A reference to the original SO and the identity of C (whom the SR is intended for), is also included in the SR tuple. L then posts the hash of the SR on the BC and forwards SR to E for verification (Line 3,4).

Algorithm 4: L creating and posting SR on BC, and sending to E for verification.

Input : SO created on BC.
Output : Surveillance request SR created and sent to E.

1 **begin**
2 L creates a surveillance request: $SR = (\mathrm{SO}||\iota = [t_s, t_e]||VK_{\mathrm{AI}}||C)$.
3 L generates and posts $H(SR)$ to BC.
4 L sends SR to E, who handles it as described in Algorithm 5.
5 **end**

---

[2]An SO could possibly have multiple, non-contiguous dates/times of surveillance. This will not affect the system design.

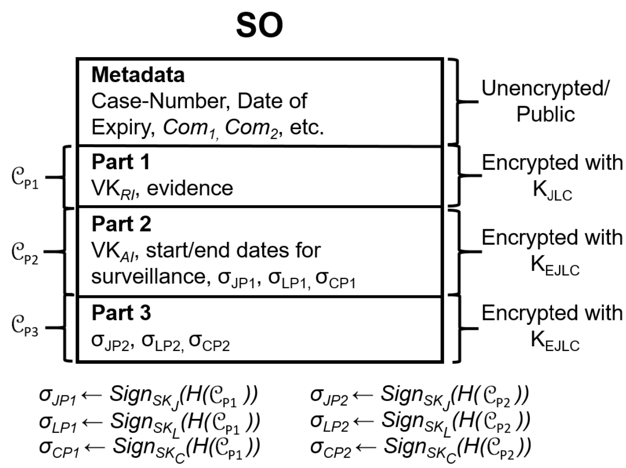


Figure 2: Structure of SO generated by J.

In SAMPL, the Enforcer uses the start/end times listed in the SO, and the pseudonymous identities of the email senders listed in the SO to check over-requesting by L and over-sharing by C.[3]

Algorithm 5: E verifying SR received from L.

Input : SR received from L.
Output : Accept or reject SR.

`/* Verify SR does not violate SO published on BC by J. */`

1 E retrieves K_{EJLC} sent by J in Protocol 3, does $P2 \leftarrow D_{EJLC}(C_{P2})$, posted on BC, and accepts SR as valid if:
2 **begin**
3 The VK_{AI} of $P2$ and SR match.
4 The time interval, $\iota = [t_s, t_e]$ contained in SR is within the timeline specified in $P2$.
5 **end**
6 If E accepts SR, a confirmation is posted on BC. Since all BC transactions are signed, we denote the corresponding transaction signature as σ^E_{SR}; and SR is forwarded to C.
7 If E rejects SR, it notifies agency L and judge J, and SR is not sent to C. It also stores evidence of the reason for rejection, which will be provided to J, L upon request.

Algorithm 5: Here E receives the SR from L and processes the SR. The verification includes checking that the time interval ι from SR is a sub-interval of the timeline contained in $P2$ of SO. After E verifies SR, it signs the hash, $H(SR)$, and posts the signature σ^E_{SR} on BC. Then SR is forwarded to the C listed as intended receiver in SR. If SR fails to verify with E, no message is posted on the BC, and SR is not forwarded to C. If fine-grained accountability is desired, the failure message can be posted to BC, identifying the reason and scope of the failure. We discuss this further in Section 9.2.

[3]This can be extended to the Enforcer checking pseudonyms of recipients too, filtering by subject of e-mails, etc., which gives finer auditing granularity. This will not affect the base system design, but will require more computations and verifications on the part of I, C, and E. We discuss such generalizations in Section 9.1.

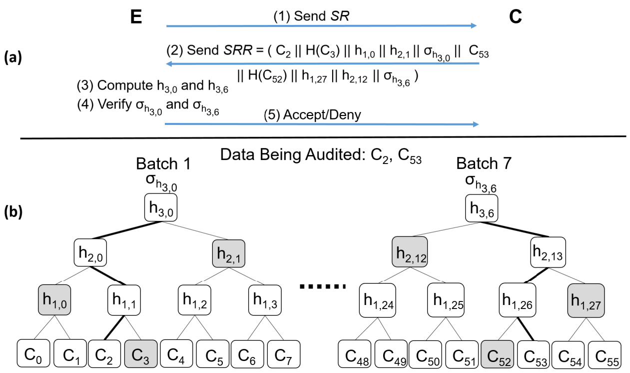


Figure 3: (a) Illustration of communication between C and E. (b) Illustration of relevant Merkle trees in SRR.

Algorithm 6 and Algorithm 7 cover interaction between E and C, as depicted in Figure 3, where C responds to an approved *SR* with an *SRR* containing requested user data. C selects user data that matches criteria outlined in *SR* (depicted by shaded items in Figure 3(b)) and adds the data to *SRR* before sending it to E for verification. E verifies the *SRR* and either forwards it to L if approved or contacts C if verification failed.

Algorithm 6: When C receives a verified *SR* from E, C verifies that ι and VK_{AI} listed in *SR* are the ones actually signed by C in $P2$ of *SO* (Line 2). C then creates an *SRR* in response to *SR*. C checks each message stored corresponding to VK_{AI} listed in the *SR*. If some message M_x in batch *bNum* matches the surveillance time ι in *SR*, then the encrypted message ($\mathcal{C}_x$), the sibling hashes for $H(\mathcal{C}_x)$ inthe Merkle tree for *bNum*, and σ_{bNum} are added to the *SRR* by C (Line 9). C also includes the ZKP for the VK_{PI_j} used to sign σ_{bNum} (Line 9). Once C has finished processing all messages for the VK_{AI} listed in *SR*, C adds identity of L to *SRR* and then posts a signed hash of *SRR* to BC (Line 13). *SRR* is then forwarded to E (Line 15).

For ease of exposition, we have presented *SRR* creation for one batch (*bNum*). If there exist multiple batches, $bNum_i$; $i \in [1..T/bSize]$, where T is the total number of I's messages stored on C, Line 7-11 of Algorithm 6 are repeated for all the batches. In Line 12, the *SRR* includes corresponding $\mathcal{C}$, sibling hashes, batch number, root of the Merkle hash tree, concatenated in order.

Algorithm 7: E receives the *SRR* from C, and parses its contents. E verifies the signature (σ^E_{SR}) on the corresponding *SR* (Line 1). For each $\mathcal{C}_x$ that appears in *SRR*, E checks that the message is dated within the time period $\iota = [t_s, t_e]$ from *SR* (Line 3). Then the root hash for $\mathcal{C}_x$, R_{bNum} is computed using the sibling hashes for $\mathcal{C}_x$ provided in *SRR*, and the signature $\sigma_{R_{bNum}}$ is verified (Line 4). The ZKP for VK_{PI_j} used in $\sigma_{R_{bNum}}$ is also verified by E (Line 5). If there are multiple batches, they are verified in succession at this time; as in Algorithm 6, we omit this part in the algorithm and this description for ease of exposition. After E verifies *SRR*, it signs $H(SRR)$, and a message containing the signature of E is posted on BC, and *SRR* is forwarded to L listed as intended receiver in *SRR* (Line 7). If *SRR* failed to verify with E, no message is posted in the BC and *SRR* is not forwarded to L. If fine-grained auditing is required, a failure message can be written to the BC.

Algorithm 6: C creating *SRR*, posting $H(SRR)$ on BC, and sending *SRR* to E for verification.

Input : *SR* received from E.
Output: *SRR* created and forwarded to E.
/* When C receives a validated SR from E, it does the following: */
1 **begin**
  /* SO verification */
2 C decrypts $\mathcal{C}_{P3}$ of *SO* contained in *SR*, verifies signatures $\sigma_{JP2}, \sigma_{LP2}$. C then decrypts $P2$ of *SO*, verifies $\sigma_{JP1}, \sigma_{LP1}$. It then checks if ($\iota$,$VK_{\mathsf{AI}}$) contained in *SR* corresponds to what it had signed in σ_{CP2}.
3 **end**
/* C create an SRR in response to the SR as follows */
4 **begin**
5 C retrieves and verifies signature σ^E_{SR} posted on BC.
6 Let $\mathbb{C}_{bNum}$ represent the set of all ciphertexts in *bNum*.
7 **for** *each* $\mathcal{C}_x \in \mathbb{C}_{bNum}$; $x \in [1..bSize]$ *for* VK_{AI} **do**
8 **if** $\mathcal{C}_x$ *was created during time-period* $\iota = [t_s, t_e]$ *from* *SR* **then**
9 Add $\mathcal{C}_x||siblingHashes(\mathcal{C}_x)||bNum||\sigma_{R_{bNum}}$ to *SRR*, Add the signed ZKP for VK_{PI_j} used to verify $\sigma_{R_{bNum}}$ ($j \in [1..m]$) to *SRR*: ($\sigma_{\mathsf{AI}}||\pi_{\mathsf{AI}}||\sigma_{\mathsf{PI}_j}||\pi_{\mathsf{PI}_j}||g||zkpVerf$), where *zkpVerf* is some metadata given to C by I for ZKP verification (details in Appendix A, Protocol 11).
10 **end**
11 **end**
12 C adds the identity of L to *SRR*. The final *SRR* is given below.

$$SRR = \langle SR||L||\mathcal{C}_x||siblingHashes(\mathcal{C}_x)|| bNum||\sigma_{R_{bNum}}||\sigma_{\mathsf{AI}}||\pi_{\mathsf{AI}}||\sigma_{\mathsf{PI}_j}|| \pi_{\mathsf{PI}_j}||g||zkpVerf\rangle$$

13 C generates and posts $H(SRR)$ on BC.
14 **end**
/* C sends SRR to E. */
15 C sends *SRR* to E, who processes it as described in Algorithm 7.

Protocol 8: Once L receives a validated *SRR* from E, it posts a signature on the hash of *SRR* to BC as acknowledgment. L then asks C to hand over K_{CI} to be able to decrypt I's encrypted emails in *SRR* and conduct surveillance.

Protocol 9: This is not a part of the regular workflow of SAMPL and is optionally executed by members of $\mathbb{U}$ on an as needed basis. It can be implemented using smart contracts. In Protocol 9, any member(s) of a set of watchdog organizations, $u \in \mathbb{U}$ (e.g., ACLU), who monitor the BC, can contact J whenever an *SO* expires and retrieve K_{JLC}, K_{EJLC}. Entity u decrypts the *SO* ($P1$ and $P2$), verifies signatures ($P3$), and can contact I who was surveilled. I, u can then investigate and verify the validity of reason for surveillance, if due

Algorithm 7: E verifying SRR received from C.

Input : SRR received from C.
Output: Accept or reject SRR.

E retrieves SR from SRR, and verifies signature σ^{E}_{SR} posted on BC.
for *each* $\mathcal{C}_x \in SRR; x \in [1..bSize]$ **do**
  E confirms that $\mathcal{C}_x$ is dated within time period ι from the SR.
  E computes $H(\mathcal{C}_x)$, runs $R_{bNum} \leftarrow rootCompute(\mathcal{C}_x || siblingHashes(\mathcal{C}_x) || bNum)$, and checks "true" $\stackrel{?}{=} Verify(VK_{\mathsf{PI}_j}, R_{bNum}, \sigma_{R_{bNum}})$.
  Finally, E verifies ZKP for VK_{PI_j} used to sign $\sigma_{R_{bNum}}$ with given $(\sigma_{\mathsf{AI}} || \pi_{\mathsf{AI}} || \sigma_{\mathsf{PI}_j} || \pi_{\mathsf{PI}_j} || g || zkpVerf)$.
end
If E accepts SRR, a confirmation is posted on BC. Since all BC transactions are signed, we denote the corresponding transaction signature as σ^{E}_{SRR}; and SRR is forwarded to L, who handles it as described in Protocol 8.
If E rejects SRR, it notifies J, C and SRR is not sent to L. It also stores evidence of the reason for rejection, which will be provided to J, C upon request.

Protocol 8: L on receiving validated SRR from E.

Input : Verified SRR received by L from E.
Output: Surveillance carried out by L.
Parties : L and C.

L receives SRR, and posts a signed hash of SRR to BC as acknowledgment of SRR received.
L gets K_{CI} from C to decrypt I's emails ($\mathcal{C}_x$'s contained in SRR), and carry out surveillance.

Protocol 9: Protocol run by members of $\mathbb{U}$.

Input : SO posted on BC.
Output: u checks adherence to protocol by parties involved in surveillance in relation to SO and follows up with J.
Parties : $u \in \mathbb{U}$ and J.

/* Whenever there is a message posted on BC by E: */
$u \in \mathbb{U}$ checks the signatures of the hashes posted.
/* Whenever an SO expires according to $\iota \in [t_s, t_e]$ posted on BC: */
u contacts J and retrieves K_{JLC}, K_{EJLC}. u decrypts $P1$, $P2$ and verifies $P3$ of the SO.

diligence was applied, and lawful procedures were followed during surveillance.

6 APPLICABILITY: CASE STUDY OF U.S. LEGAL SYSTEM

In this section, we discuss how our system can be instantiated and adapted in a real-world legal system to provide accountability. We consider the U.S. legal system as an example, and discuss our system within its constitutional and jurisdictional parameters. SAMPL can be modified to be applicable to legal systems in other countries.

6.1 Different Authorization Paths

The U.S. constitution provides several authorization paths for law enforcement agencies to obtain permission to conduct surveillance, some with judicial oversight, some without. We discuss them below.

Electronic Communications Privacy Act (ECPA): ECPA was created by the U.S. Congress in 1986 [17] to elucidate the boundaries of government surveillance on citizens, and clearly define the ways and means by which government surveillance can be conducted.[4] ECPA can be used by federal law enforcement agencies to obtain information about users' emails in transit, emails at rest, phone calls, location data, and more. ECPA provides law enforcement agencies two methods of accessing users' information: via warrant, or via subpoena. A subpoena is a court order demanding that someone or something be provided to assist in a case. For issuing a warrant, the law enforcement agency must show the issuing judge probable cause that a crime has been, or will be committed. Most warrants are unsealed when charges are filed against someone, and the defendant has the right to see the evidence collected against them before the trial.

Per ECPA statute 18 U.S.C. §2616 [3] and statute 18 U.S.C. §2703 [4], emails in transit, emails in storage on home computer, and unopened emails in remote storage stored for $\leq$ 180 days all need a warrant for law enforcement access. Opened emails in remote storage, and unopened emails stored for $>$ 180 days only need a subpoena for law enforcement access.

Our system can be deployed in a straightforward manner in both cases, as described in Section 5, where the SO written to the blockchain by J can be either a subpoena or a warrant. The SR and the furnished data are all routed through the Enforcer, E, who writes the data transfer success/failure to the blockchain BC for auditing (refer Section 5).

National Security Letter (NSL): The USA PATRIOT Act §505 [36, 37] empowered the Federal Bureau of Investigation (FBI) to issue a *National Security Letter* compelling companies to disclose information about their customers for a national security-related investigation. An NSL is typically issued to a company by a local FBI field office and does not require judicial oversight. It can be used to obtain meta-information about phone/email records, times, length of service, network addresses, how a customer paid for service; although the FBI cannot obtain actual content of phone/email records. An NSL can also be used by the FBI to obtain financial details, such as credit reports, and bank account details from banks, credit unions and insurance companies. Any recipient of an NSL is prohibited by law or "gagged" from disclosing their receipt of the NSL, which makes oversight difficult. Additionally, the U.S. government can seek judicial enforcement of an NSL in non-compliance situations, under ECPA statute 18 U.S.C. §2709.

[4] There have been calls to reform ECPA, with several amendments to the law being made over the years, such as the USA PATRIOT Act, among others. It has also faced criticism for being outdated and not inclusive of many modern methods of communication, since it was first codified in 1986. Nevertheless, as of this writing, ECPA with its amendments is the law in the U.S. relating to surveillance processes. A full discussion of proposals and avenues for future ECPA reform, and their possible consequences on our system is out of the scope of this paper.

Since NSL does not require a judge, there is no J to post the SO to BC. But L and C can still use SAMPL and hence E for auditability. L would create an SO for the NSL, post it on BC, and then create an SR, before sending it to E. E would then pass it on to C, after checking that the SO and the SR does not request content of emails (which is a legal restriction on NSLs). Note that E cannot check if an SO is for a genuine national security issue, since U.S. law expressly gives discretionary powers to the FBI while deciding to issue NSLs. But what E can help check is if the SO and SR adhere to legal guidelines, that is, the agency is only seeking meta-information. On receipt of the SR from E, C will construct and return an SRR to E, who will then verify it and send it to L.

Our pseudonymous identity scheme prevents E from learning the actual identities of the users whose details were requested. As discussed before, E writes the pass/fail result of the SR and SRR to the BC. The legal/political feasibility of writing encrypted NSLs to the BC is out of the scope of this paper.

Foreign Intelligence Surveillance Act (FISA): FISA was enacted in 1978 and amended in 2008 by the U.S. Congress for the purposes of surveillance related to foreign powers and persons [22]. Under FISA, a person who is believed to be a foreign power, or spying on behalf of a foreign power can be put under surveillance, even if they haven't engaged in any criminal activity. Over the years, the ambit of FISA has gradually expanded to include electronic surveillance, "roving wiretap" surveillance, pen-registers, and trap-and-trace devices, per 50 U.S.C. Ch. 36 [5]. Additionally, FISA permits *warrantless surveillance*[5] up until certain time periods, beyond which the agency conducting the surveillance needs to obtain a warrant from a special court called the FISA court [23]. Although the court maintains records of its proceedings, the FISA court's records are not available to the public.

Our system can be applied to the FISA ecosystem, which encompasses the court, and surveilling agencies which work with it, such as the NSA. The FISA ecosystem operates with little to no auditability (other than annual aggregate statistics published by the court). Using our system, the FISA court judges will issue and post an encrypted SO on the BC. The E can verify that the surveillance is not conducted in wholesale or an overarching manner by agencies, and only data that is pertinent to an ongoing investigation is revealed by companies. In particular, our system allows independent non-profit organizations (e.g., ACLU) to verify if due process has been followed during FISA-authorized surveillance, *even if the actual court orders are never made public*, without compromising national security.

6.2 Law Enforcement Agencies Overreach: Violations, "sneak peeks" and more

In the U.S. legal system, a government agency, e.g., FBI or NSA is, in most cases, required to present a warrant to a company for conducting surveillance on its customers, and conduct the surveillance within the confines of the warrant. Unfortunately, in practice, there are agency overreaches; we outline a few here. In 2018, the NSA's Office of Inspector General (OIG) in its first semi-annual unclassified report to the U.S. Congress described the investigations and activities of the NSA [32]. Among other findings, the OIG report found "several deficiencies that have the potential to impact the protection of U.S. persons privacy rights," in relation to FISA investigations conducted by the NSA.

A report by the Department of Justice (DoJ) OIG found that the FBI issued NSLs "contrary to statutory limitations," issued "improper requests under the statute referenced in the NSL", "obtained information beyond the time period referenced in the NSL," and various other illegal uses of NSLs [1]. A partially redacted 300-page report by the DoJ OIG [2] also found that the FBI acquired phone call information regarding "hot numbers" without legal process, made inaccurate statements to the FISA court, and improperly used FBI administrative subpoenas. The OIG report also finds that the FBI uses "exigent letters" and other informal requests for phone records that do not comply with legal requirements or FBI policies governing the acquisition of those records. The same report also found the FBI has a practice of conducting "sneak peeks" for telephone toll records in providers' databases without due process, a practice that violates the ECPA statute 18 U.S.C. §2702(a)(3).

All said, our system will help systematize a seemingly unpredictable process that would help law enforcement agencies and companies ensure that they follow the letter of the law with respect to issuing and responding to surveillance requests respectively.

7 SECURITY ANALYSIS

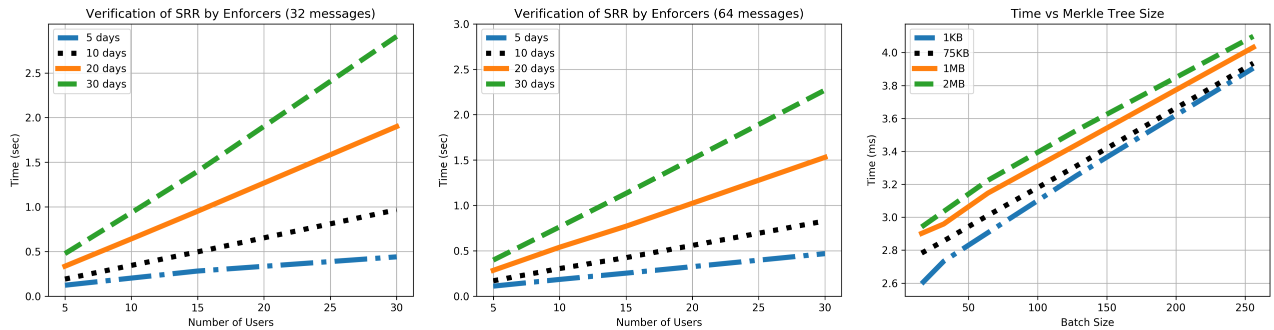
We prove the security of our constructions in the well-known Universal Composability (UC) framework [12]. The UC paradigm elegantly captures the conditions under which a given distributed protocol is secure, by comparing it to an ideal realization of the protocol. To this end, the UC framework defines two "worlds": the real-world, where the protocol, π to be proved secure runs in the presence of a real-world adversary, $\mathcal{A}$. The other is the ideal-world, where the entire protocol, ϕ is executed by an ideal, trusted functionality, in the presence of a simulator, $\mathcal{S}$, which models the ideal-world adversary. All users only talk to an ideal functionality via secure and authenticated channels, the ideal functionality takes input from users, performs some computations in a possibly interactive manner, and returns the output of the protocol. The goal then is to prove that no distinguishing algorithm, commonly called as "environment", $\mathcal{Z}$, can successfully distinguish between the execution (EXEC) of the two worlds.

7.1 Design of Ideal Functionalities

We define an ideal functionality, $\mathcal{F}_{\mathsf{Surveil}}$ which encompasses all our other functionalities and algorithms, and consists of four independent ideal functionalities, $\mathcal{F}_{\mathsf{Surveil}} = (\mathcal{F}_{\mathsf{zk}}^{\mathrm{SAMPL}}, \mathcal{F}_{\mathsf{init}}, \mathcal{F}_{\mathsf{create}}, \mathcal{F}_{\mathsf{BC}})$. Furthermore, we assume that $\mathcal{F}_{\mathsf{Surveil}}$ maintains internal state that is accessible at any time to $\mathcal{F}_{\mathsf{zk}}^{\mathrm{SAMPL}}, \mathcal{F}_{\mathsf{init}}, \mathcal{F}_{\mathsf{create}}, \mathcal{F}_{\mathsf{BC}}$. We describe the functionalities of $\mathcal{F}_{\mathsf{Surveil}}$, discuss some of their motivating design choices, and give the proof of the following theorem in Appendix B.

Theorem 7.1. *Let $\mathcal{F}_{\mathsf{Surveil}}$ be an ideal functionality for* SAMPL. *Let $\mathcal{A}$ be a probabilistic polynomial-time (PPT) adversary for* SAMPL, *and let $\mathcal{S}$ be an ideal-world PPT simulator for $\mathcal{F}_{\mathsf{Surveil}}$.* SAMPL *UC-realizes $\mathcal{F}_{\mathsf{Surveil}}$ for any PPT distinguishing environment $\mathcal{Z}$.*

[5]One such program has recently come to light [41].



(a) (b) (c)

Figure 4: (a) Verification time of SRR by E for different surveillance durations (legends) at batch size of 32 messages, (b) Verification time of SRR, batch size of 64 messages, and (c) Merkle tree computation time at C for different message sizes (legends) and batch sizes.

8 EXPERIMENTATION AND RESULTS

We evaluate the performance of SAMPL for scalability, and to benchmark the operations performed by different entities within SAMPL with varying system parameters and surveillance requirements.

8.1 Experimental Setup

Four Desktop class machines with Intel(R) Core(TM) i7-6700K CPUs and 8 GB RAM each were used to run our implementation of SAMPL. Each of the machines ran a single entity in SAMPL: J, E, L, and C, communicating over C Sockets. The entities were coded using the C programming language and compiled with gcc version 7.3.0 (Ubuntu 7.3.0-27 ubuntu1 18.04). Our code, along with test data generators, and a small test database is available online [33]. Random user data, for 500 users, was pre-generated and stored in an SQL database (we use email as the representative application) at C. User data was created for 120 days.

In our experiment, RI for a given user is tied to their real name and each user has an AI tied to their name in the database, where the AI is a key pair that is tied to the user's $\mathsf{PI}_i; i \in [1..m]$ using ZKPs. We simulated with only a single PI_i for each user's data, during the surveillance period. The cryptographic operations of signing and verifying user data, and ZKP related operations were prototyped using the Charm Cryptographic framework [7]. AES-256 in GCM mode was used for the symmetric key encryption involving K_{CI}, K_{JLC}, and K_{EJLC}. For emulating the blockchain in SAMPL, we used Ethereum [18]. Each entity ran its own Ethereum node and communicated with the local blockchain network.

8.2 Metrics and Parameters

Separate simulations were run for 5, 10, 15, and 30 users in the SO posted by J. The surveillance periods simulated were 5, 10, 20, and 50 days. These aforementioned values (number of users, days) were chosen to demonstrate scalability in the event of concurrency. We evaluate SAMPL using the following metrics:
(1) ZKP generation and verification time per user: The $Prime192v1$ Elliptic Curve was used as the prime order group $\mathbb{G}$ for ZKP as described in Protocol 11.
(2) Merkle root generation and signing per user: Simulations were run for batch-sizes with 16, 32, 64, 128, and 256, leaves in the tree with message sizes set to 1 KB, 75 KB, 1 MB, and 2 MB.
(3) Enforcer Verification Time: Measured for 5, 10, 15, and 30 users, batch sizes of 32 and 64 messages, and surveillance period of 5, 10, 20, and 50 days. The message size was set to 75 KB.

Verification of *SR* by E as depicted in Figure 1: Step 11, is not quantified in the results because it does not involve complex cryptographic operations. This step would incur a low computational cost regardless of the number of AIs and duration of surveillance in *SR*, as it only involves comparisons and range checks between *SR* and the corresponding *SO* on BC.

8.3 Results

Table 2 reflects the ZKP verification and generation times per user averaged over 100 runs. The generation time is calculated for the setup in Protocol 1 (only establishment of PI: ref. Protocol 11). The average ZKP generation time was 1.02 ms with a standard deviation of 0.236 ms. This time is expended when an I signs up for a new account with C or whenever I establishes a new PI with C. The verification time is calculated for an E verifying the user data inside *SRR* (calculated once per *SRR*). The verification time was found to be 1.066 ms with standard deviation of 0.096 ms.

Figure 4a shows the verification time of *SRR* by E for different number of Is in *SR*, and different surveillance periods, for a batch size of 32 messages. Figure 4b shows the *SRR* verification time, for a batch size of 64 messages. We observed a linear increase in computation time with an increase in the number of users. We note that the computation time includes the ZKP verifications, the Merkle tree generation and root signature verification (one per user), and doing the date range checks on the data.

Comparison of Figures 4a and 4b shows that the verification of *SRR* for batch size of 64 messages is faster by roughly 0.65 s. This difference is because for the same number of total messages, larger batch sizes will result in less Merkle tree roots and signature verification operations when compared to smaller batch sizes. In our simulations, *SRR* verification for 10 users for a surveillance period of 30 days involved processing 299 batches with the batch size of 32 messages, as opposed to 153 batches with 64 messages. Similarly, number of batches processed for 30 users over 30 days involved processing 898 batches with 32 messages and 460 batches with 64 messages.

Figure 4c shows the computation time at C at the end of each batch. Batch sizes of 16, 32, 64, 128, and 256 messages were simulated for messages sizes of 1 KB, 75 KB, 1 MB, and 2 MB, averaged over 50 runs. The larger message sizes represent emails with attachments. The computation time for the different message sizes converges as the batch size grows. This is because once the hashes of the messages are calculated for leaves of the Merkle tree, the rest of the operations on Merkle trees of given batch size are the same for messages of different sizes. For Merkle trees with larger messages initial hash computation of the leaves of the tree has to deal with larger data size.

To give a fine-grained analysis of components of *SRR* verification at E, we give a break down of the computation time in Table 3. For each step, it does follow that the amount of time taken is linear, as the number of users and/or surveillance period is increased, hence showing the scalability of our approach.

Table 2: Zero Knowledge Proof Timings

Operation	Mean	Standard Deviation
ZKP Generation	1.02 *ms*	0.236 *ms*
ZKP Verification	1.066 *ms*	0.096 *ms*

We note that the total time for operations performed on a given *SRR* depicted in Table 3 are lower than the computation time depicted in Figures 4a. This is due to the extra operations for look ups and other input-output operations performed by E on *SRR* during the verification.

9 DISCUSSION

In this section we discuss some generalizations and possible enhancements of SAMPL.

9.1 Generalization

SAMPL can apply to other types of surveillance criteria by modifying the way user records are stored by C. In case of email, the sender and receiver names and their IP addresses could be salted and hashed separately and stored along with other details such as date and time as the metadata. This information could be listed in the *SO* and subsequently verified by E without learning the actual values. This will enable search based on sender/receiver names and/or IP addresses. Searchable encryption [9] can be implemented to search based on specific keywords in the data records. Although this increases the types of surveillance auditable, it leaks more information to E.

SAMPL can also be extended to allow a user to delete historical records. The user would update the data record to a generic "deleted message," the Merkle root for the given historical batch would be recalculated with the new message, and the user would sign the updated Merkle root with the current SK_{PI}. Every time VK_{PI} gets updated by the user, C verifies the ZKP and also verifies the signatures on the Merkle root so that I cannot inject fake data/signatures to frame an honest C. For practicality, the management of users' VK_{PI}s can be handled by software like keystores, plugins, etc.

There can be instances where a single user is part of multiple surveillance requests. In that case, each *SO* has VK_{AI}, and E can link it to the corresponding VK_{PI} using the ZKP provided by C. Our framework does not provide unlinkability of the independent surveillances to a user's VK_{PI}. The problem of malicious enforcers leaking information about the VK_{AI}s is not addressed by us.

SR can also include requests for system logs showing activity of a certain user identified by VK_{AI}. If the logs contain VK_{RI}, to preserve privacy, C can replace it with VK_{AI}. If the logs contain VK_{PI}, then C furnishes the ZKP associated with VK_{PI}. Unlike data such as emails, users do not see the logs, hence do not sign them.

9.2 Enhancements and Adaptability

There are several design choices in our system that are implementation-specific. We list some below:

(1) Set of Enforcers: We can relax the assumption on the E from honest to being honest but curious. To provide unlinkability of users' PIs over multiple surveillances for a given time period, nonoverlapping striping of data across the set of Es, when sending *SR* or *SRR* could be used. Note that the sets of enforcers chosen by L and C need not be the same. This would increase the efficiency of verification of the system, as data for verification is not duplicated between different Es. As long as the number of *SO*s for a given AI does not exceed the number of enforcers in the system, the unlinkability assumption will hold (due to the non-overlapping striping).

(2) Internal decision procedures of J, L, and C: Certain actions are largely dependent on the specific jurisdiction, and are governed by the laws of the country where J, L, and C operate. What exactly J, L, and C do when any of their internal decision procedures return a "reject" in the course of operation is beyond the scope of SAMPL. For example, what happens when C decides to reject the *IO* when the *IO* violates statutory company policy in some way? Or what is the course of action for L to follow if J decides to reject its surveillance request?

(3) Handling Multiple Users: We described, prototyped, and analyzed SAMPL with an example of a single user being surveilled by L, this can easily be extended to multiple users ($I \in [1..\alpha]$) by modifying the *SO* to include a list of users' identities. We would then have $VK^1_{\mathsf{RI}}, \cdots, VK^{\alpha}_{\mathsf{RI}}$, $VK^1_{\mathsf{AI}}, \cdots, VK^{\alpha}_{\mathsf{AI}}$, and $VK^1_{\mathsf{PI}_i}, \cdots, VK^{\alpha}_{\mathsf{PI}_i}$. When multiple identities are surveilled, J needs to add a random string (salt) to each user's identity($VK^1_{\mathsf{RI}}, \cdots, VK^{\alpha}_{\mathsf{RI}}$) and hash it before putting it in $P1$ of *SO*. This randomization added to each identity will help protect the identities of each of the surveilled users from each other whenever the *SO* expires and is released to the individuals being surveilled. The random salts for all the VK_{RI}'s are shared with L and C.

(4) Hard reject/soft reject: Whenever an *SR* or *SRR* is rejected by E, perhaps due to clerical errors, administrative errors, or otherwise honest mistakes on the part of C or L, E just responds with a reject message and takes no further action (soft reject). E can assess how many times a certain party's request/response has been rejected. Once this number of rejections reaches a threshold, which can be a system parameter, E informs the party whose request/response was rejected, and the judge J, and stores a local copy of the reason

Table 3: ***SRR*** **Verification time (sec) break-down at E for $bNum = 32$ and message size of 75 KB.**

Surveillance Period (Days)	5				10				50			
Number of Users	5	10	15	30	5	10	15	30	5	10	15	30
ZKP Verification for PI_i (s)	0.039	0.0795	0.1207	0.2003	0.0750	0.1528	0.229	0.459	0.369	0.740	1.109	2.219
Merkle Root Generation (s)	0.140	0.259	0.382	0.619	0.246	0.475	0.705	1.390	1.1178	2.224	3.32	6.64
Merkle Sign Verification (s)	0.015	0.0304	0.046	0.0767	0.0286	0.0583	0.0875	0.1757	0.141	0.282	0.423	0.846

for the rejection (to provide to J upon request), and writes a "fail" message to the BC – a hard reject. Note that for fined grained information on errors/malicious behaviors, E can choose to post soft reject on BC.

(5) Auditable data structures: Auditable data structures [26] implemented on C's servers could also be used by E to verify that C is non-malicious and complying with court orders. This implementation would need careful system design with read/write counters on the data stores with E having access to the counters.

(6) Forward Security: If one of I's previously used SK_{PI} gets compromised and C gets access to it, C can fake I's historical data by modifying the Merkle trees for past batches and signing them with the compromised key. To guard against this, each time I chooses a new PI, a new Merkle tree is created between I and C whose leaves are the signed root hashes of the past batches. The root of this new hierarchical Merkle tree is signed with the new PI. This operation can be repeated for each new PIs to make it harder for a malicious C to frame I, since C would need to compromise multiple SK_{PI}s belonging to I.

10 CONCLUSION

In this paper, we present a practical mechanism for secure auditing of surveillance orders by an overseer called *Enforcer*, *E*. The E checks if law enforcement agencies and companies are over-requesting and over-sharing user data, respectively, beyond what is permitted by the surveillance order, in a privacy-preserving way, such that E does not know the real identities of the users getting surveilled, nor does it get to read the users' unencrypted data. Our system also has inbuilt checks and balances to require unsealing of surveillance orders at the appropriate times, thus enabling accounting of the surveillance operation being surveilled to verify that lawful procedures were followed, protecting users from government overreach, and helping law enforcement agencies and companies demonstrate that they followed the rule of law.

ACKNOWLEDGMENT

The authors thank Dr. Dennis Giever, Head, Department of Criminal Justice, New Mexico State University, and an anonymous, retired FBI agent for their valuable comments.

Research supported by US NSF awards #1800088; #1719342; #1345232, #1914635, EPSCoR Cooperative agreement OIA-1757207; and US Army Research Office grant #W911NF-07-2-0027. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the federal government.

REFERENCES

[1] Dept. of Justice Office of Inspector General: A review of the federal bureau of investigations's use of national security letters. https://oig.justice.gov/reports/2016/o1601b.pdf.
[2] Dept. of Justice Office of Inspector General: A review of the federal bureau of investigations's use of exigent letters and other informal requests for telephone records. https://oig.justice.gov/special/s1001r.pdf.
[3] Government Publishing Office. https://www.govinfo.gov/app/details/USCODE-2015-title29/USCODE-2015-title29-chap28-subchapI-sec2616.
[4] Government Publishing Office. https://www.govinfo.gov/app/details/USCODE-2011-title18/USCODE-2011-title18-partI-chap121-sec2703.
[5] Foreign Intelligence Surveillance. http://uscode.house.gov/view.xhtml?path=/prelim@title50/chapter36&edition=prelim.
[6] American civil liberties union. https://www.aclu.org.
[7] Akinyele, J. A., Garman, C., Miers, I., Pagano, M. W., Rushanan, M., Green, M., and Rubin, A. D. Charm: a framework for rapidly prototyping cryptosystems. *J. Cryptographic Engineering 3*, 2 (2013), 111–128.
[8] Bates, A. M., Butler, K. R. B., Sherr, M., Shields, C., Traynor, P., and Wallach, D. S. Accountable wiretapping -or- I know they can hear you now. In *19th Annual Network and Distributed System Security Symposium, NDSS* (2012).
[9] Bellare, M., Boldyreva, A., and O'Neill, A. Deterministic and efficiently searchable encryption. In *Annual International Cryptology Conference* (2007), Springer, pp. 535–552.
[10] Boneh, D. The decision diffie-hellman problem. In *Algorithmic Number Theory, Third International Symposium, ANTS, Proceedings* (1998), pp. 48–63.
[11] Bovard, J. *Terrorism and Tyranny: Trampling Freedom, Justice, and Peace to Rid the World of Evil.* Palgrave Macmillan, 2004.
[12] Canetti, R. Universally composable security: A new paradigm for cryptographic protocols. In *42nd Annual Symposium on Foundations of Computer Science, FOCS* (2001), pp. 136–145.
[13] Canetti, R. Universally composable signature, certification, and authentication. In *17th IEEE Computer Security Foundations Workshop, (CSFW-17* (2004).
[14] Canetti, R., Lindell, Y., Ostrovsky, R., and Sahai, A. Universally composable two-party and multi-party secure computation. In *Proceedings on 34th Annual ACM Symposium on Theory of Computing (STOC)* (2002), pp. 494–503.
[15] Chase, M., and Lysyanskaya, A. On signatures of knowledge. In *Annual International Cryptology Conference* (2006), Springer, pp. 78–96.
[16] Chaum, D., and Pedersen, T. P. Wallet databases with observers. In *Advances in Cryptology - CRYPTO* (1992), pp. 89–105.
[17] ECPA. https://it.ojp.gov/privacyliberty/authorities/statutes/1285.
[18] Ethereum project. https://www.ethereum.org.
[19] Facebook transparency report. https://transparency.facebook.com/government-data-requests/country/US.
[20] Fiat, A., and Shamir, A. How to prove yourself: Practical solutions to identification and signature problems. In *Advances in Cryptology - CRYPTO '86, Santa Barbara, California, USA, 1986, Proceedings* (1986), pp. 186–194.
[21] Fisa annual reports. https://www.uscourts.gov/statistics-reports/analysis-reports/directors-report-foreign-intelligence-surveillance-courts.
[22] Foreign Intelligence Surveillance Act. https://it.ojp.gov/privacyliberty/authorities/statutes/1286.
[23] Foreign Intelligence Surveillance Court. https://www.fisc.uscourts.gov.
[24] Frankle, J., Park, S., Shaar, D., Goldwasser, S., and Weitzner, D. J. Practical accountability of secret processes. In *27th USENIX Security Symposium, USENIX* (2018), pp. 657–674.
[25] Goldwasser, S., and Park, S. Public accountability vs. secret laws: Can they coexist?: A cryptographic proposal. In *Proceedings of the 2017 on Workshop on Privacy in the Electronic Society* (2017), WPES '17, pp. 99–110.
[26] Goodrich, M. T., Kornaropoulos, E. M., Mitzenmacher, M., and Tamassia, R. Auditable data structures. In *2017 IEEE European Symposium on Security and Privacy, EuroS&P* (2017), pp. 285–300.
[27] Google transparency report: Requests for user information. https://transparencyreport.google.com/user-data/overview?hl=en&user_requests_report_period=series:requests,accounts;authority:US;time:Y2018H1&lu=user_requests_report_period.
[28] J. Kroll and E. W. Felten and D. Boneh. Secure protocols for accountable warrant execution. http://www.cs.princeton.edu/~felten/warrant-paper.pdf, 2014.
[29] Kamara, S. Restructuring the NSA metadata program. In *Financial Cryptography and Data Security - FC 2014 Workshops, BITCOIN and WAHC* (2014), pp. 235–247.
[30] Microsoft: Law enforcements requests report. https://www.microsoft.com/en-us/corporate-responsibility/lerr.

[31] Moore, A. Privacy, security, and government surveillance: Wikileaks and the new accountability. *Public Affairs Quarterly 25*, 2 (2011), 141–156.
[32] NSA OIG semi-annual report to Congress. https://www.oversight.gov/sites/default/files/oig-sa-reports/OIG%20UNCLASS%20SAR%20OCT-MAR%202018.pdf.
[33] SAMPL. https://github.com/nsol-nmsu/SAMPL.
[34] ACLU vs. ODNI: FOIA lawsuit seeking records about government surveillance under the USA freedom act. https://tinyurl.com/y28u3c2g.
[35] Parsons, C., and Molnar, A. Government surveillance accountability: the failures of contemporary canadian interception reports. *Canadian journal of law and technology 16*, 1 (2018), 143–169.
[36] The USA PATRIOT Act: Preserving Life and Liberty. https://www.justice.gov/archive/ll/highlights.htm.
[37] The USA PATRIOT Act of 2001. https://www.sec.gov/about/offices/ocie/aml/patriotact2001.pdf.
[38] Segal, A., Feigenbaum, J., and Ford, B. Privacy-preserving lawful contact chaining: [preliminary report]. In *Proceedings of the 2016 ACM on Workshop on Privacy in the Electronic Society, WPES* (2016), pp. 185–188.
[39] Segal, A., Ford, B., and Feigenbaum, J. Catching bandits and only bandits: Privacy-preserving intersection warrants for lawful surveillance. In *4th USENIX Workshop on Free and Open Communications on the Internet, FOCI* (2014).
[40] Setty, S. Surveillance, secrecy, and the search for meaningful accountability. *Stanford Journal of International Law 51*, 1 (2015), 69–103.
[41] The NSA continues to violate Americans' internet privacy rights. https://www.aclu.org/blog/national-security/privacy-and-surveillance/nsa-continues-violate-americans-internet-privacy.
[42] Wikimedia vs. NSA: Challenge to upstream surveillance under the FISA amendments act. https://www.aclu.org/cases/wikimedia-v-nsa-challenge-upstream-surveillance-under-fisa-amendments-act.

A ZERO-KNOWLEDGE PROOFS BETWEEN I AND C

Protocol 10 is initiated by I when she needs to establish her real identity RI (corresponding to an email address and represented by keypair $(VK_{\mathsf{RI}}, SK_{\mathsf{RI}})$) and tie it to an anonymized identity AI (corresponding to a nickname for the email address and represented by keypair $(VK_{\mathsf{AI}}, SK_{\mathsf{AI}})$). I can choose to create a new AI if she needs to change the current AI in case SK_{AI} gets compromised.

The goal of Protocol 10 is for I to establish her $(VK_{\mathsf{RI}}, SK_{\mathsf{RI}})$, $(VK_{\mathsf{AI}}, SK_{\mathsf{AI}})$ keypairs, and prove in zero-knowledge to C that VK_{AI} could have been generated only by someone who had knowledge of SK_{RI}, and that the two key-pairs are related to each other by a DDH tuple. To this end, I and C do a Chaum-Pedersen-style interactive ZKP [16] (ZKP) for I to prove her anonymized identity, AI to C. The proof π_{AI} can be made non-interactive by applying the Fiat-Shamir transform [20]. If C chooses to accept the proof as valid, it asks I to send a signed copy of the transcript of the proof, σ_{AI}. C stores π_{AI} and σ_{AI}.

Protocol 11 is initiated by I when she needs to establish her pseudonymous identity (PI) keypair $(VK_{\mathsf{PI}_i}, SK_{\mathsf{PI}_i})$, where $i \in [1..m]$. I could have multiple PIs tied in to a single AI, but only one can be active at a given point in time. I creates a new PI_{i+1} if SK_{PI_i} gets compromised or after a certain time period, which could be a system parameter.

The goal of Protocol 11 is for I to establish her $(VK_{\mathsf{PI}_i}, SK_{\mathsf{PI}_i})$ keypairs, and prove in zero-knowledge to C that VK_{PI_i} could only have been generated by someone who had knowledge of SK_{AI}, and the two key-pairs are related to each other by a DDH tuple. To this end, I and C do a Chaum-Pedersen-style IZKP [16], similar to Protocol 10 for I to prove her current pseudonymous identity, VK_{PI_i} to C (made non-interactive by applying the Fiat-Shamir transform). If C chooses to accept the proof, PI_i, as valid, it asks I to send a signed copy σ_{PI_i} of the transcript of the proof. C stores π_{PI_i} and σ_{PI_i}. π_{PI_i} and σ_{PI_i} are used by C during surveillance to prove

Protocol 10: Setup of (RI, AI) keypairs.

Inputs : Public parameters: Group $\mathbb{G}$, $q = |\mathbb{G}|$, $g, h \in \mathbb{G}$.
ZKP Claim: VK_{AI} was generated by someone with knowledge of $SK_{\mathsf{AI}}, SK_{\mathsf{RI}}$.
Witness: $SK_{\mathsf{AI}}, SK_{\mathsf{RI}}$.
Output : Signed ZKP: $Sign_{SK_{\mathsf{RI}}}(\pi_{\mathsf{AI}})$
Parties : C and I

I picks $a, a' \leftarrow \mathbb{Z}_q$, sets $SK_{\mathsf{RI}} = a$, $SK_{\mathsf{AI}} = a'$, and $VK_{\mathsf{RI}} = g^a$, $VK_{\mathsf{AI}} = g^{a'}$.
begin
- I picks $\omega_1 = g^{a \cdot a'}$, and sends DDH tuple $(g, X = g^a, Y = g^{a'}, Z = g^{a \cdot a'})$ to C.
- C picks a challenge $s \leftarrow \mathbb{Z}_q$, and sends Com$(s)$ to I, where Com is a Pedersen commitment.
- I picks $r_1 \leftarrow \mathbb{Z}_q$, computes $y_1 = g^{r_1} \bmod q$, $y_2 = g^{a' \cdot r_1} \bmod q$. I sends y_1, y_2 to C.
- C sends s to I.
- I verifies Com, computes response $z = a \cdot s + r_1 \bmod q$, and sends (z, y_1, y_2) to C.
- C verifies if $g^z \stackrel{?}{=} (X^s \cdot y_1) \bmod q$, and if $Y^z \stackrel{?}{=} (Z^s \cdot y_2) \bmod q$. If checks verify, C accepts the response as valid, asks I to send signed transcript of proof, π_{AI}.
- I sends $\sigma_{\mathsf{AI}} = Sign_{SK_{\mathsf{RI}}}(\pi_{\mathsf{AI}} = H(g||VK_{\mathsf{RI}}||VK_{\mathsf{AI}}||\omega_1||y_1||y_2||s||z))$ to C.

end

that PI_i was generated by I. Although we have abstracted it out, a Pedersen commitment is of the form $g^v \cdot h^r (\bmod q)$, where $g, h \in \mathbb{G}$, $q = |\mathbb{G}|$, v is the value to be committed to, and r is the commitment randomness. Here $h = g^a \bmod q$, where $a \leftarrow \mathbb{Z}_q$ is chosen by the receiver of the commitment. We assume h is fairly chosen in a distributed manner by I, C.

Note that the ZKP and the signature on the ZKP can be replaced by a single signature proof of zero-knowledge of knowledge [15], but we do not discuss this optimization in this paper.

B UC FUNCTIONALITIES AND ANALYSIS

The notion of UC security is captured by the pair of definitions below:

Definition B.1. (UC-emulation [12]) Let π and ϕ be probabilistic polynomial-time (PPT) protocols. We say that π UC-emulates ϕ if for any PPT adversary $\mathcal{A}$ there exists a PPT adversary $\mathcal{S}$ such that for any balanced PPT environment $\mathcal{Z}$ we have

$$\mathrm{EXEC}_{\phi,\mathcal{S},\mathcal{Z}} \approx \mathrm{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}$$

Definition B.2. (UC-realization [12]) Let $\mathcal{F}$ be an ideal functionality and let π be a protocol. We say that π UC-realizes $\mathcal{F}$ if π UC-emulates the ideal protocol for $\mathcal{F}$.

We describe the functionalities of $\mathcal{F}_{\mathsf{Surveil}}$: $\mathcal{F}_{\mathsf{zk}}^{\mathsf{SAMPL}}$, $\mathcal{F}_{\mathsf{init}}$, $\mathcal{F}_{\mathsf{create}}$, $\mathcal{F}_{\mathsf{BC}}$. We assume that $\mathcal{F}_{\mathsf{Surveil}}$ maintains a table τ, with information about the individuals being surveilled, and the surveillance orders. A single row of the table would look like: $(VK_{\mathsf{RI}}, SO, \mathsf{soid})$ where

Protocol 11: Setup of (PI_i) keypair.

Inputs : Public parameters: Group $\mathbb{G}, |\mathbb{G}| = q, g, h \in \mathbb{G}, VK_{\mathsf{AI}}, SK_{\mathsf{AI}}$.
Claim : VK_{PI_i} was generated by someone with knowledge of $SK_{\mathsf{AI}}, SK_{\mathsf{PI}_i}$.
Witness : $SK_{\mathsf{AI}}, SK_{\mathsf{PI}_i}$.
Output: Signed ZKP: $Sign_{SK_{\mathsf{AI}}}(\pi_{\mathsf{PI}_i})$.
Parties : C and I

```
for i ∈ [1..m] do
    I picks a'' ← Z_q, sets SK_{PI_i} = a'', and VK_{PI_i} = g^{a''}.
    begin
        I parses SK_AI as a' and VK_AI as g^{a'}.
        I picks ω_2 = g^{a'·a''}. I sends DDH tuple
          (g, zkpVerf = (Y = g^{a'}, P = g^{a''}, Q = ω_2)) to C.
        C picks a challenge s_1 ← Z_q, and sends Com(s_1) to
          I, where Com is a Pedersen commitment.
        I picks r'_1 ← Z_q, computes y'_1 = g^{r'_1} mod q, and
          y'_2 = g^{a''·r'_1} mod q, and sends y'_1, y'_2 to C.
        C sends s_1 to I.
        I verifies Com, computes response:
          z_1 = a' · s_1 + r'_1 mod q, and sends (z_1, y'_1, y'_2) to C.
        C verifies if g^{z_1} =? (Y^{s_1} · y'_1) mod q, and if
          P^{z_1} =? (Q^{s_1} · y'_2) mod q. If checks verify, C accepts
          the response as valid, asks I to send signed
          transcript of proof, π_{PI_i}.
        I sends σ_{PI_i} = Sign_{SK_AI}(π_{PI_i} =
          H(g||VK_AI||VK_{PI_i}||ω_2||y'_1||y'_2||s_1||z_1)) to C.
    end
end
```

soid denotes the id number of the SO which is associated with VK_{RI}. We use $\perp$ to denote unresponsive parties, malformed replies to the ideal functionalities, and ideal functionalities returning fail messages to parties.

$\mathcal{F}_{\mathsf{zk}}^{\mathrm{SAMPL}}$: We define our ideal functionality for zero-knowledge proofs, $\mathcal{F}_{\mathsf{zk}}^{\mathrm{SAMPL}}$, based on the ideal zero knowledge functionality, $\mathcal{F}_{\mathsf{zk}}$ defined by Canetti *et al.* [14]. While [14] deals with generic relations, our $\mathcal{F}_{\mathsf{zk}}^{\mathrm{SAMPL}}$ is restricted only to discrete-log relations, and also involves the ideal functionality writing the claim to the shared table τ. $\mathcal{F}_{\mathsf{zk}}^{\mathrm{SAMPL}}$ is given in Figure 5, and the $\mathcal{F}_{\mathsf{zk}}$ functionality of [14] is given in Figure 6.

$\mathcal{F}_{\mathsf{zk}}^{\mathrm{SAMPL}}$ is parametrized by a prime-order cyclic group $\mathbb{G}, |\mathbb{G}| = q$, $g \in \mathbb{G}$, $a \in \mathbb{Z}_q$, and a session id, *sid*. The prover, I sends a claim to be proven, VK_{RI} to $\mathcal{F}_{\mathsf{zk}}$, and a witness a. $\mathcal{F}_{\mathsf{zk}}$ checks if $g^a = VK_{\mathsf{RI}}$, i.e., if the claim is correct and forwards VK_{RI} to the verifier C and the ideal-world adversary $\mathcal{S}$, and writes VK_{RI} into table τ.

$\mathcal{F}_{\mathsf{zk}}$ as given in Figure 6, is parametrized by a relation R, and a session id, *sid*. The prover, P sends a claim to be proven, x to $\mathcal{F}_{\mathsf{zk}}$, and a witness w. $\mathcal{F}_{\mathsf{zk}}$ checks if $R(x, w) = 1$, i.e., if the claim is correct and forwards x to the verifier V and the ideal-world adversary $\mathcal{S}$.

$\mathcal{F}_{\mathsf{init}}$: The $\mathcal{F}_{\mathsf{init}}$ ideal functionality described in Figure 7, interacts with J, L, and C, initiates the process for creating a SO, and posts the SO to the BC. L initiates contact with $\mathcal{F}_{\mathsf{init}}$ by sending a

Functionality $\mathcal{F}_{\mathsf{zk}}^{\mathrm{SAMPL}}$

$\mathcal{F}_{\mathsf{zk}}^{\mathrm{SAMPL}}$ proceeds as follows, running with prover I, verifier C, an adversary $\mathcal{S}$. Let $\mathbb{G}$ be a prime-order cyclic group, $g \in \mathbb{G}$, $|\mathbb{G}| = q$, and $a \in \mathbb{Z}_q$.

(1) Upon receiving $(VK_{\mathsf{RI}}, sid, a)$ from I, if $g^a = VK_{\mathsf{RI}}$, send (VK_{RI}, sid) to C and $\mathcal{S}$, else exit. Write (VK_{RI}) to table τ and exit.

Figure 5: Ideal functionality for ZKPs in SAMPL

Functionality $\mathcal{F}_{\mathsf{zk}}$

$\mathcal{F}_{\mathsf{zk}}$ proceeds as follows, running with a prover P, verifier V, and an adversary $\mathcal{S}$, and parametrized with a relation R:

(1) Upon receiving $(\mathsf{zk}-\mathsf{prover}, sid, x, w)$ from P, if $R(x, w) = 1$, send $(\mathsf{zk}-\mathsf{proof}, sid, x)$ to V and $\mathcal{S}$ and exit. Otherwise exit.

Figure 6: Ideal functionality for ZKPs [14]

$(\mathsf{create}-\mathsf{IO}, evidence, VK_{\mathsf{RI}})$ request tuple to $\mathcal{F}_{\mathsf{init}}$. $\mathcal{F}_{\mathsf{init}}$ forwards the request to J, who can accept or decline it. If J accepts, $\mathcal{F}_{\mathsf{init}}$ creates an intermediate order $IO = (VK_{\mathsf{RI}}, evidence)$ and forwards the IO to C. C can either accept or decline the IO. If either J or C declines the IO request, $\mathcal{F}_{\mathsf{init}}$ aborts the execution and exits. If C accepts, $\mathcal{F}_{\mathsf{init}}$ checks if VK_{RI} was deposited in the shared table τ by $\mathcal{F}_{\mathsf{zk}}^{\mathrm{SAMPL}}$.

If yes, it means VK_{RI} was verified by $\mathcal{F}_{\mathsf{zk}}^{\mathrm{SAMPL}}$. $\mathcal{F}_{\mathsf{init}}$ then generates a key $K \leftarrow \{0,1\}^{\lambda}$, generates a string regarding the surveillance order, $data \leftarrow \{0,1\}^{\lambda}$, which includes *evidence* provided by L, crimes committed by VK_{RI}, reason for sealing, etc. It also generates *metadata*, which includes the date the SO will be unsealed. $\mathcal{F}_{\mathsf{init}}$ writes (SO, soid) to the table τ, in the VK_{RI} row. $\mathcal{F}_{\mathsf{init}}$ then creates the SO tuple: $(metadata, \mathcal{C} = E_K(VK_{\mathsf{RI}}, data))$, sends $(K, \mathcal{C})$ to J, L, C, calls $\mathcal{F}_{\mathsf{BC}}$ and posts the SO on the BC. Finally, when the SO needs to be unsealed, $\mathcal{F}_{\mathsf{init}}$ proactively contacts I, whom VK_{RI} belongs to, and gives her the decrypted contents of the SO.

$\mathcal{F}_{\mathsf{create}}$: $\mathcal{F}_{\mathsf{create}}$ is given in Figure 8. $\mathcal{F}_{\mathsf{create}}$ creates a request SR and response SRR. L first contacts $\mathcal{F}_{\mathsf{create}}$ for creating an SR by sending VK_{RI}, upon which $\mathcal{F}_{\mathsf{create}}$ looks up table τ for an SO corresponding to VK_{RI}. If none has been entered by $\mathcal{F}_{\mathsf{init}}$, that means L is not authorized to surveil VK_{RI}, and $\mathcal{F}_{\mathsf{create}}$ returns $\perp$ to L. Else $\mathcal{F}_{\mathsf{create}}$ proceeds with generating SR and forwards SR to L and C. At this point C is expected to respond to SR with VK_{RI}'s emails, batch information, and Merkle tree information required to verify the emails are from the correct batches.

We represent all this information by a string, *records*. If C ignores the request, $\mathcal{F}_{\mathsf{create}}$ will write C's identity to BC, along with the associated SO (this means C is behaving maliciously). If C responds with the records, $\mathcal{F}_{\mathsf{create}}$ will first verify that the records belong to the surveillance time-period as given in the *metadata* part of the SO. If verification succeeds, $\mathcal{F}_{\mathsf{create}}$ will create the SRR, which will be sent to L and C. Finally $\mathcal{F}_{\mathsf{create}}$ posts the hash of SR, SRR to BC respectively.

$\mathcal{F}_{\mathsf{BC}}$: The blockchain functionality is given in Figure 9. $\mathcal{F}_{\mathsf{BC}}$ receives messages from $\mathcal{F}_{\mathsf{init}}$ and $\mathcal{F}_{\mathsf{create}}$. $\mathcal{F}_{\mathsf{BC}}$ writes tuples to the

Functionality $\mathcal{F}_{\text{init}}$

(1) L sends $\mathcal{F}_{\text{init}}$ a tuple requesting for an *IO*, (create − IO, *evidence*, VK_{RI}), which $\mathcal{F}_{\text{init}}$ forwards to J. J replies with either (accept, VK_{RI}) or $\perp$. If J responds with $\perp$, $\mathcal{F}_{\text{init}}$ returns $\perp$ to L and exits.
(2) If J accepts the *IO* request, $\mathcal{F}_{\text{init}}$ creates an intermediate order $IO = (VK_{\text{RI}}, \textit{evidence})$ and sends it to C. C can either send (accept) or $\perp$ to $\mathcal{F}_{\text{init}}$. If C sends (accept), $\mathcal{F}_{\text{init}}$ checks if VK_{RI} is present in table τ. If yes, $\mathcal{F}_{\text{init}}$ proceeds to the next step. If either C sends $\perp$, or VK_{RI} is not present in τ, $\mathcal{F}_{\text{init}}$ sends $\perp$ to J, L, C and exits.
(3) $\mathcal{F}_{\text{init}}$ picks a symmetric key, $K \leftarrow \{0,1\}^{\lambda}$, and generates *data* $\leftarrow \{0,1\}^{\lambda}$ creates a surveillance order tuple, $SO = (\textit{metadata}, (\mathcal{C} = E_K(VK_{\text{RI}}, \textit{data}))$, and picks an soid $\in \mathbb{Z}^+$. $\mathcal{F}_{\text{init}}$ writes (SO, soid) to τ in the VK_{RI} row. $\mathcal{F}_{\text{init}}$ sends $(K, \mathcal{C})$ to J, L, C. $\mathcal{F}_{\text{init}}$ calls $\mathcal{F}_{\text{BC}}$ and writes SO to the blockchain.
(4) At the time of unsealing of SO, $\mathcal{F}_{\text{init}}$ sends I a tuple $SO = (\textit{metadata}, VK_{\text{RI}}, \textit{data})$ and exits.

Figure 7: Ideal functionality for issuance of SO.

Functionality $\mathcal{F}_{\text{create}}$

(1) L sends a tuple (create − SR, VK_{RI}) to $\mathcal{F}_{\text{create}}$. $\mathcal{F}_{\text{create}}$ looks up the SO corresponding to VK_{RI} in τ. If none exists, $\mathcal{F}_{\text{create}}$ sends $\perp$ to L and exits. Else, $\mathcal{F}_{\text{create}}$ generates an $SR = (SO, VK_{\text{RI}})$ and forwards it to L and C.
(2) C replies to $\mathcal{F}_{\text{create}}$ with a tuple (VK_{RI}, *records* $\leftarrow \{0,1\}^{\lambda}$), where *records* $\leftarrow \{0,1\}^{\lambda}$ denote VK_{RI}'s emails, and verification metadata. If C replies with $\perp$, $\mathcal{F}_{\text{create}}$ will call $\mathcal{F}_{\text{BC}}$ and write (SO, C) to the BC and exit.
(3) In response to C's tuple, $\mathcal{F}_{\text{create}}$ verifies *records*, and creates an $SRR = (SO, \textit{records})$ tuple, and forwards to L and C.
(4) $\mathcal{F}_{\text{create}}$ calls $\mathcal{F}_{\text{BC}}$, posts $H(SR)$ and $H(SRR)$ to the BC and exits.

Figure 8: Ideal functionality for creating *SR*, *SRR*.

blockchain, and sends a copy of the new block, B, to parties J, L, C, I. This is done by sending (update, B). The party can either accept the update, or decline (unresponsive, disconnected, or non-cooperating parties). When a dormant party wishes to update itself, it can request a copy of the full blockchain by sending a read message to $\mathcal{F}_{\text{BC}}$.

B.1 Discussion and Analysis

We now briefly discuss the correctness of our ideal functionalities, some of our motivating design choices, including aspects that may seem unusual.

B.1.1 Correctness. The privacy properties our system aims to provide are accountability for L and C, protection against a forgetful J who might forget to unseal orders, and protection against a malicious I and C. The design of our ideal functionalities need to capture these properties.

Accountability is provided by the fact that $\mathcal{F}_{\text{create}}$ generates the *SR* and *SRR*, thus ensuring that no data is over-requested by L, or over-shared by C, both in terms of redundant data belonging to the same user, or other users' data. $\mathcal{F}_{\text{init}}$ creates the *SO* and guarantees that the *SO* will get unsealed by $\mathcal{F}_{\text{init}}$ before it exits, thus providing protection against forgetful J. Since $\mathcal{F}_{\text{zk}}^{\text{SAMPL}}$ checks the witness and generates the ZKP for each VK_{RI}, it ensures that a user cannot create a fake ZKP for VK_{RI} that passes verification, yet the corresponding SK_{RI} cannot be used for decrypting the user's email records. Protection against a malicious C which tries to include fake data in an *SRR* is provided by $\mathcal{F}_{\text{create}}$, which verifies C's returned user information before creating an *SRR*.

B.1.2 Peculiar design choices. (1) In $\mathcal{F}_{\text{init}}$, J, C can return $\perp$ to $\mathcal{F}_{\text{init}}$ in Step 1 and Step 2 respectively: This is to model the fact that in the real-world, J has the right to refuse a surveillance request by L, and C has the right to refuse or appeal an intermediate order by J.
(2) $\mathcal{F}_{\text{init}}$ creates an *SO*, and $\mathcal{F}_{\text{create}}$ generates the *SR* and *SRR* for *SO*, but only after being contacted by L (Step 1 of $\mathcal{F}_{\text{create}}$): This is because in the real-world, L might get an *SO* authorized by J, but may choose not to follow it up with any action, i.e., eventually not conduct any surveillance, e.g., because L needs to invest its limited resources in higher-priority matters, budget cuts after *SO* was issued, etc.
(3) $\mathcal{F}_{\text{create}}$ writes C's identity to the BC if C doesn't respond with I's email records in Step 2 of $\mathcal{F}_{\text{create}}$: We assume that I is an (email) customer of C, and C will have email records associated with I for the surveillance period. These emails are stored only with C. If C deliberately chooses not to respond to, or refuses an *SR*, *after* having accepted the *IO* that the *SR* is a follow up on ($\mathcal{F}_{\text{init}}$, Step 2), then that can only be construed as malicious behavior on the part of C. Hence $\mathcal{F}_{\text{create}}$ will expose malicious C's identity on the public BC.

Functionality $\mathcal{F}_{\text{BC}}$

(1) $\mathcal{F}_{\text{BC}}$ receives three kinds of write messages: $\mathcal{F}_{\text{init}}$ writes *SO*, $\mathcal{F}_{\text{create}}$ writes (SO, C) and $\mathcal{F}_{\text{create}}$ writes $(H(SR), H(SRR))$. $\mathcal{F}_{\text{BC}}$ writes the tuples to the blockchain and sends a copy of the newest block B to all parties, J, L, C, I by sending a tuple (update, B).
(2) Each party either replies with (agree, B) or $\perp$. In the former case, the party updates the local copy of the blockchain, and is synced with the global blockchain. However if the reply was $\perp$, the party now has an outdated copy of the blockchain.
(3) In the event that an outdated party wants to get synced with the blockchain, it sends a message (*read*) to $\mathcal{F}_{\text{BC}}$, $\mathcal{F}_{\text{BC}}$ replies with (update, B'), where B' is the copy of the entire blockchain.

Figure 9: Ideal functionality for blockchain

B.2 Proof

We now give the proof of Theorem 7.1.

Proof: Our goal is to describe a simulator $\mathcal{S}$ such that for any real-world $\mathcal{A}$ running with SAMPL, $\mathcal{Z}$ cannot distinguish $\mathcal{A}$ from an ideal-world $\mathcal{S}$ running with $\mathcal{F}_{\text{Surveil}}$. $\mathcal{S}$ runs $\mathcal{A}$ internally, simulates all inputs $\mathcal{A}$ is expecting, and gives $\mathcal{A}$'s outputs to $\mathcal{F}_{\text{Surveil}}$, who will complete the simulation in the ideal-world. $\mathcal{S}$ reflects the choices of $\mathcal{A}$ in the ideal-world, and reflects the protocol outcomes and aborts of the ideal-world in the real-world. If $\mathcal{A}$ cheats, $\mathcal{S}$ aborts.

That is, any attempt by $\mathcal{A}$ to cheat in the real-world will result in the protocol aborting in the both, the ideal and real worlds. Hence it follows that no PPT $\mathcal{Z}$ can distinguish between $\text{EXEC}_{\text{SAMPL},\mathcal{A},\mathcal{Z}}$ and $\text{EXEC}_{\mathcal{F}_{\text{Surveil}},\mathcal{S},\mathcal{Z}}$. We now consider a complete run of the protocol, starting from when I sets up her RI, AI, π keypairs with C and ending when L receives the validated *SRR* from a subset of E.

First, $\mathcal{S}$ needs to create keypairs (VK_{RI}, SK_{RI}), (VK_{AI}, SK_{AI}), (VK_{PI_j}, SK_{PI_j}), $j \in [1..m]$. $\mathcal{S}$ simulates a UC-secure digital signature scheme, $\mathcal{S}_{\text{sig}}$, that UC-realizes the ideal digital signature functionality, $\mathcal{F}_{\text{sig}}$ (see [13] for UC-secure signatures definitions), and creates the keypairs. The $VK_{\text{RI}}, VK_{\text{AI}}$ and VK_{PI_j} will be handed over to $\mathcal{A}$. If $\mathcal{A}$ wishes to corrupt I, $SK_{\text{RI}}, SK_{\text{AI}}$ and SK_{PI_j} will also be given to $\mathcal{A}$.

$\mathcal{S}$ will also have to generate the zero-knowledge proofs associated with VK_{AI} and VK_{PI_j}. $\mathcal{S}$ runs the steps of Protocol 10, computes $\pi_{\text{AI}} = (H(g||VK_{\text{RI}}||VK_{\text{AI}}||\omega_1||y_1||y_2||s||z))$ and generates σ_{AI} by calling $\mathcal{S}_{\text{sig}}$. $\mathcal{S}$ then gives π_{AI} and σ_{AI} to $\mathcal{A}$ along with the keys. $\mathcal{S}$ follows a similar procedure for generating the $\pi_{\text{PI}_j}, \sigma_{\text{PI}_j}$ of Protocol 11. In the ideal-world, $\mathcal{S}$ will call $\mathcal{F}_{\text{sig}}$ to generate VK_{RI}, and call $\mathcal{F}_{\text{zk}}^{\text{SAMPL}}$ for generating the ZKP corresponding toVK_{RI}. If $\mathcal{A}$ rejects the ZKPs or signatures, $\mathcal{S}$ aborts the execution.

$\mathcal{S}$ then needs to setup shared key K_{CI} of Protocol 1, and pass it to $\mathcal{A}$, if $\mathcal{A}$ has corrupted either C and/or I. $\mathcal{S}$ creates a key $K \leftarrow \{0,1\}^{\lambda}$ by calling $\mathcal{F}_{\text{init}}$, and passes it to $\mathcal{A}$. Finally, $\mathcal{S}$ generates a random batch-size *bSize* and gives to $\mathcal{A}$. This completes the simulation of the setup phase.

Next, $\mathcal{S}$ needs to pass on inputs to $\mathcal{A}$ during the *SO* creation phase, and simulate the corresponding actions in the ideal-world (actions of Protocol 3). If $\mathcal{A}$ has corrupted L, $\mathcal{A}$ will generate the *SR* = (VK_{RI}, *evidence*), else, $\mathcal{S}$ generates the *SR* = (VK_{RI}, *evidence*) and gives the *SR* to $\mathcal{A}$. We recollect from our adversary model that J is forgetful, but not malicious. In other words, $\mathcal{A}$ cannot corrupt J.

Once J (impersonated by $\mathcal{S}$) has received the *SR* from $\mathcal{A}$, J will validate it, and decide whether to accept it or not. Once J decides, it will give its output to $\mathcal{A}$. $\mathcal{A}$ will then pass on the *IO* to C, through corrupted L. C will decide whether to accept the *IO* or not. If $\mathcal{A}$ has corrupted C, then this communication is handled locally by $\mathcal{A}$, and need not be simulated. If C is honest, its action will be simulated by $\mathcal{S}$.

C responds to the *IO*, and generates an *SRR* = ($VK_{\text{AI}}||\pi_{\text{AI}}||\sigma_{\text{AI}}$), and sends *SRR* to J, L. In the ideal world, $\mathcal{S}$ calls $\mathcal{F}_{\text{init}}$, which creates an *IO* and sends to J, L, C. If $\mathcal{A}$ cheats, i.e., includes a wrong $\pi_{\text{AI}}, \sigma_{\text{AI}}$ inside the *SRR*, then $\mathcal{S}$ will send a corresponding malformed message to $\mathcal{F}_{\text{init}}$, which will then abort (Step 2 of Figure 7), and $\mathcal{S}$ aborts the execution of $\mathcal{A}$. $\mathcal{S}$ then generates the *SO* as a honest J would, and gives the final *SO* to $\mathcal{A}$. If either of C or L are corrupted by $\mathcal{A}$, or if a subset of E are corrupted by $\mathcal{A}$, $\mathcal{S}$ will send the K_{JLC} and/or K_{EJLC} to $\mathcal{A}$. We do not give details of the *SO* generation by $\mathcal{S}$, since it is straightforward (simulate $\mathcal{S}_{\text{sig}}$ for signatures, $\mathcal{F}_{\text{sig}}$ in the ideal-world, etc.). If at any step, the signatures in the *SO* sent by $\mathcal{A}$ do not verify, $\mathcal{S}$ aborts. In the ideal-world, $\mathcal{S}$ calls $\mathcal{F}_{\text{init}}$ who will in turn call $\mathcal{F}_{\text{BC}}$ and posts the *SO* to the blockchain.

The next step for $\mathcal{S}$ is to simulate the storage of I's emails on C (Protocol 2). There are three cases to consider:

(1) *Case 0*: If both I and C are corrupted by $\mathcal{A}$, this is handled locally by $\mathcal{A}$, and does not need to be simulated.
(2) *Case 1*: If C is corrupted, but I is not, $\mathcal{S}$ creates I's outputs, i.e., for each $M_x \in \mathbb{M}_{bNum}, x \in [1..bSize]$, $\mathcal{S}$ generates a $\mathcal{C}_x$. $\mathcal{A}$, playing the role of corrupted C will create a Merkle hash tree with the $H(\mathcal{C}_x)$ at the leaves, which will be checked by $\mathcal{S}$. $\mathcal{S}$ will verify the root-hash and will abort of there is any mismatch. Else, $\mathcal{S}$ will sign the root-hashes by simulating $\mathcal{S}_{\text{sig}}$. In the ideal world, $\mathcal{S}$ will get random strings signed by calling $\mathcal{F}_{\text{sig}}$.
(3) *Case 2*: If I is corrupted, but C is not, $\mathcal{A}$ does I's leaf encryptions, creates $\mathcal{C}_x$'s, etc., and gives to $\mathcal{S}$. $\mathcal{S}$ generates the corresponding root-hashes for the Merkle trees, and sends the root-hashes to $\mathcal{A}$ for signing. $\mathcal{A}$ is expected to sign the root-hashes. If $\mathcal{A}$ refuses to sign the root-hashes, $\mathcal{S}$ will abort.

Now, $\mathcal{S}$ needs to simulate the creation and verification of the *SR* (Algorithm 4, and Algorithm 5). For this, $\mathcal{S}$ will retrieve the *SO*, ι, etc., and construct a tuple *SR* = ($SO||\iota||VK_{\text{RI}}||C$) and forward it to a subset of E. If L is corrupted, $\mathcal{A}$ will construct the *SR* tuple. If $\mathcal{A}$'s *SR* tuple is malformed, $\mathcal{S}$ aborts. In the ideal world, $\mathcal{S}$ calls $\mathcal{F}_{\text{create}}$, who generates the *SR*. At this point, $\mathbb{S} \subseteq E$ needs to validate *SR*. Per our adversary model, $\mathcal{A}$ can corrupt a minority of members in $\mathbb{S}$. Here there are two cases to consider:

(1) *Case 0*: None of $\mathbb{S}$ are corrupted: $\mathcal{S}$ verifies *SR* (if *SR* was generated by $\mathcal{A}$ in the previous step), and checks it against the *SO* $\mathcal{S}$ had created. $\mathcal{S}$ simulates $\mathcal{S}_{\text{sig}}$ and creates the signature $\sigma_{SR}^{\mathbb{S}}$, and gives it to $\mathcal{A}$. In the ideal world, $\mathcal{S}$ calls $\mathcal{F}_{\text{sig}}$ and creates the signature.
(2) *Case 1*: A (minority) subset of $\mathbb{S}$ are corrupted by $\mathcal{A}$. For the minority, $\mathcal{A}$ will check the *SR*. If $\mathcal{A}$ rejects the *SR*, or refuses to produce a signature $\sigma_{SR}^{\mathbb{S}}$, for any reason, $\mathcal{S}$ aborts, and sends a malformed request to $\mathcal{F}_{\text{create}}$, which will abort the simulation in the ideal world. Communication among members of corrupted minority of $\mathbb{S}$ is controlled by $\mathcal{A}$ and need not be simulated. If $\mathcal{A}$ behaves properly, i.e., validates the *SR* and produces signature $\sigma_{SR}^{\mathbb{S}}$, $\mathcal{S}$ will simulate the honest majority, and the ideal world similar to Case 0.

The next step is for $\mathcal{S}$ to simulate C producing an *SRR*, and a subset of E verifying the *SRR*. $\mathcal{S}$ first retrieves the *SO* it created. Here again there are two broad cases:

(1) *Case 0*: If C is uncorrupted, $\mathcal{S}$ retrieves the $\mathcal{C}_x \in \mathbb{C}_{bn}; x \in [1..bSize]$, adds the $\mathcal{C}_x$'s, sibling hashes, etc. to the *SRR* tuple, the ZKP tuple it created before, calls $\mathcal{S}_{\text{sig}}$, signs the *SRR* tuple, and gives the $H(SRR)$, along with the signed *SRR* to $\mathcal{A}$. $\mathcal{A}$ then passes it on to $\mathbb{S} \subseteq E$, who will accept or reject it. If all members of $\mathbb{S}$ are honest, $\mathcal{S}$ will validate the signed *SRR* and

we are done. In the ideal world, $\mathcal{S}$ will call $\mathcal{F}_{\text{create}}$, $\mathcal{F}_{\text{zk}}^{\text{SAMPL}}$, and $\mathcal{F}_{\text{sig}}$ to create and sign the *SRR*, respectively.

(2) *Case 1*: If C is corrupted, $\mathcal{A}$ will create the *SRR*; the *SO* is given to $\mathcal{A}$. Firstly, $\mathcal{A}$ can return a verification fail on the *SO* created by $\mathcal{S}$. If this happens, $\mathcal{S}$ will abort the simulation. If $\mathcal{A}$ chooses to proceed with the simulation, $\mathcal{A}$ will create the Merkle hash trees with the $H(\mathcal{C}_x)$ at the leaves, sibling hashes, etc.. $\mathcal{A}$ will give the ZKPs, π_{AI}, π_{PI_j} and signatures on the ZKPs, σ_{AI}, σ_{PI_j} to $\mathcal{S}$. If any do not verify, $\mathcal{S}$ aborts. $\mathcal{A}$ will generate the final *SRR*, and $H(SRR)$. If the *SRR* is malformed, in the ideal-world, $\mathcal{S}$ will cause $\mathcal{F}_{\text{create}}$ to abort by having C not reply to an *SRR*. $\mathcal{F}_{\text{create}}$ will write malicious C's identity to the blockchain by calling $\mathcal{F}_{\text{BC}}$.

If a minority of $\mathbb{S} \subseteq E$ are corrupted, $\mathcal{A}$ can return a fail on the ZKP verification, upon which $\mathcal{S}$ aborts. If $\mathcal{A}$ rejects the *SRR*, or refuses to produce a signature $\sigma_{SRR}^{\mathbb{S}}$, $\mathcal{S}$ aborts. In the ideal world, $\mathcal{S}$ will corrupt C such that C does not repond to $\mathcal{F}_{\text{create}}$'s request for an *SRR*, upon which $\mathcal{F}_{\text{create}}$ will write C's identity to the blockchain by calling $\mathcal{F}_{\text{BC}}$, and will then abort. If $\mathcal{A}$ validates the *SRR* and produces signature $\sigma_{SRR}^{\mathbb{S}}$, $\mathcal{S}$ will simulate the honest majority. In the ideal world $\mathcal{S}$ will call $\mathcal{F}_{\text{sig}}$. Lastly, $\mathcal{S}$ will give K_{CI} to $\mathcal{A}$, if $\mathcal{A}$ had not already corrupted C and/or I, and obtained K_{CI} earlier. This concludes our proof. ∎