Abuse Resistant Law Enforcement Access Systems

Matthew Green[1], Gabriel Kaptchuk[2], and Gijs Van Laer[1]

[1]Johns Hopkins University, {mgreen, gijs.vanlaer}@jhu.edu
[2]Boston University, kaptchuk@bu.edu

Abstract

The increasing deployment of end-to-end encrypted communications services has ignited a debate between technology firms and law enforcement agencies over the need for lawful access to encrypted communications. Unfortunately, existing solutions to this problem suffer from serious technical risks, such as the possibility of operator abuse and theft of escrow key material. In this work we investigate the problem of constructing law enforcement access systems that mitigate the possibility of unauthorized surveillance. We first define a set of desirable properties for an abuse-resistant law enforcement access system (ARLEAS), and motivate each of these properties. We then formalize these definitions in the Universal Composability (UC) framework, and present two main constructions that realize this definition. The first construction enables *prospective* access, allowing surveillance only if encryption occurs after a warrant has been issued and activated. The second, more powerful construction, allows *retrospective* access to communications that occurred prior to a warrant's issuance. To illustrate the technical challenge of constructing the latter type of protocol, we conclude by investigating the minimal assumptions required to realize these systems.

1 Introduction

Communication systems are increasingly deploying end-to-end (E2E) encryption as a means to secure physical device storage and communications traffic. E2E encryption systems differ from traditional link encryption mechanisms in that keys are not available to service providers, but are instead held by endpoints: typically end-user devices such as phones or computers. This approach ensures that plaintext data cannot be accessed by providers and manufacturers, or by attackers who may compromise their systems. Widely-deployed examples include messaging protocols [Sig, Wha17, Appc], telephony [Appa], and device encryption [Appb, Goo], with some systems deployed to billions of users.

The adoption of E2E encryption in commercial services has provoked a backlash from the law enforcement and national security communities around the world, based on concerns that encryption will hamper agencies' investigative and surveillance capabilities [WMT15, FB14, Bar19]. The U.S. Federal Bureau of Investigation has mounted a high-profile policy campaign called "Going Dark" around these issues [Fed], and similar public outreach has been conducted by agencies in other countries [LR18]. These campaigns have resulted in legislative proposals in the United States [Pop16, Gra20, SLG20] that seek to discourage the deployment of "warrant-proof" end-to-end encryption, as well as adopted legislation in Australia that requires providers to guarantee access to plaintext in commercial communication systems [Tar18].

The various legislative proposals surrounding encryption have ignited a debate between technologists and policymakers. Technical experts have expressed concerns that these proposals, if implemented, will undermine the security offered by encryption systems [AAB+15, Nat16, Sin20], either by requiring unsafe changes or prohibiting the use of E2E encryption altogether. Law enforcement officials have, in turn, exhorted researchers to develop new solutions that resolve these challenges [Bar19]. However, even the basic technical requirements of such a system remain unspecified, complicating both the technical and policy debates.

Existing Proposals for Law Enforcement Access. A number of recent and historical technical proposals have been advanced to resolve the technical questions raised by the encryption policy debate [Den94, Sav18, BBB+18, WV18, Tai16, LR18, BR99]. With some exceptions, the bulk of these proposals are variations on the classical *key escrow* [DB96] paradigm. In key escrow systems, one or more trusted authorities retain key material that can be used to decrypt targeted communications or devices.

Technologists and policymakers have criticized key escrow systems [AAB+15, Enc19, Nat18], citing concerns that, without additional protection measures, these systems could be abused to covertly conduct mass surveillance of citizens. Such abuses could result from a misbehaving operator or a compromised escrow keystore. Two recent policy working group reports [Enc19, Nat18] provide evidence that, at least for the case of communications services, these concerns are shared by members of the policy and national security communities.[1] Reflecting this consensus, recent high-profile technical proposals have limited their consideration only to the special case of *device encryption*, where physical countermeasures (*e.g.,* physical possession of a device, tamper-resistant hardware) can mitigate the risk of mass surveillance [Sav18, BBB+18]. Unfortunately, expanding the same countermeasures to messaging or telephony software seems challenging.

Abuse of Surveillance Mechanisms. Escrow-based access proposals suffer from three primary security limitations. First, key escrow systems require the storage of valuable key material that can decrypt most communications in the system. This material must be accessible to satisfy law enforcement request, but must simultaneously be defended against sophisticated, nation-state supported attackers. Second, in the event that key material is surreptitiously exfiltrated from a keystore, it may be difficult or impossible to detect its subsequent misuse. This is because escrow systems designed to allow lawful access to encrypted data typically store *decryption keys*, which can be misused without producing any detectable artifact.[2] Finally, these access systems require a human operator to interface between the digital escrow technology and the non-digital legal system, which raises the possibility of misbehavior by operators. These limitations must be addressed before any law enforcement access system can be realistically considered, as they are not merely theoretical: wiretapping and surveillance systems have proven to be targets for both nation-state attacks and operator abuse [BL06, Nak13, Gor13].

Overcoming these challenges is further complicated by law enforcement's desire to access data that was encrypted before an investigation is initiated. For example, several recent investigations requested the unlocking of suspects' phones or message traffic in the wake of a crime or terrorist attack [LG16]. Satisfying these requests would require *retrospectively* changing the nature of the encryption scheme used: ciphertext must be strongly protected before an investigation begins, but they must become accessible to law enforcement after an investigation begins. Satisfying these contradictory requirements is extraordinarily challenging without storing key material that can access all past ciphertexts, since a ciphertext may be created *before* it is known if there will be a relevant investigation in the future.

Law enforcement access systems that do not fail open in the face of lost key material or malicious operators have been considered in the past, *e.g.,* [Bla96, BR99, WV18]. Bellare and Rivest [BR99] proposed a mechanism to build *probabilistic* law enforcement access, in order to mitigate the risk of mass surveillance. Wright and Varia [WV18] proposed cryptographic puzzles as a means to increase the financial cost of abuse. While these might be theoretically elegant solutions, such techniques have practical limitations that may hinder their adoption: law enforcement is unlikely to tolerate arbitrary barriers or prohibitive costs that might impede legitimate investigations. Moreover, these proposals do little to enable detection of key theft or to prevent more subtle forms of misuse.

Towards Abuse Resistant Law Enforcement Access. In this work, we explore if it is technically possible to limit abuse while giving law enforcement the capabilities they are truly seeking: quickly decrypting relevant ciphertexts during legally compliant investigations. To do this, we provide a new cryptographic definition for an *abuse resistant law enforcement access system.* This definition focuses on abuse resistance by weaving

[1] The Carnegie Institution report [Enc19] concludes that "In the case of data in motion, for example, our group could identify no approach to increasing law enforcement access that seemed reasonably promising to adequately balance all of the various concerns".

[2] This contrasts with the theft of *e.g.,* digital certificates or signing keys, where abuse may produce artifacts such as fraudulent certificates [Nig11] or malware artifacts that can be detected through Internet-wide surveillance.

accountability features throughout the access process. More concretely, our goal is to construct systems that realize the following three main features:

- **Global Surveillance Policies.** To prohibit abuse by authorized parties, access systems must enforce specific and *fine-grained* global policies that restrict the types of surveillance that may take place. These policies could, for example, encompass limitations on the number of messages decrypted, the total number of targets, and the types of data accessed. They can be agreed upon in advance and made publicly available. This approach ensures that global limits can be developed that meet law enforcement needs, while also protecting the population against unlimited surveillance.
- **Detection of Abuse.** We require that any unauthorized use of escrow key material can be detected, either by the public or by authorized auditing parties. Achieving this goal ensures that even fully-adversarial use of escrow key material (*e.g.,* following an undetected key exfiltration) can be detected, and the system's security can be renewed through rekeying.
- **Operability.** At the same time, escrow systems must remain *operable*, in the sense that honest law enforcement parties should be able to access messages sent through a compliant system. We aim to guarantee this feature by ensuring that it is easy to verify that a message has been correctly prepared.

We stress that the notion of abuse-resistance is different from *impossible to abuse*. Under our definitions abuse may still happen, but the features described above will allow the abuse to be quickly identified and system security renewed. The most critical aspect of our work is that we seek to enforce these features *through the use of cryptography*, rather than relying on correct implementation of key escrow hardware or software, or proper behavior by authorities.

Prospective vs. retrospective surveillance. We will divide the access systems we discuss into two separate categories: *prospective* and *retrospective*. When using a *prospective* system, law enforcement may only access information encrypted sent or received from suspects *after* those suspects have been explicitly selected as targets for surveillance: this is analogous to "placing an alligator clip on a wire" in an analog wiretap. A *retrospective* access system, as described above, allows investigators to decrypt past communications, even those from suspects who were not the target of surveillance when encryption took place. Retrospective access clearly offers legitimate investigators more capabilities, but may also present a greater risk of abuse. Indeed, achieving accountable access in the challenging setting of retrospective key escrow, where encryption may take place *prior* to any use of escrow decryption keys, is one of the most technically challenging aspects of this work.

Our contributions. More concretely, in this work we make the following contributions.

- **Formalizing security notions for abuse resistant law enforcement access systems.** We first provide a high-level discussion of the properties required to prevent abuse in a key escrow system, with a primary focus on the general data-in-motion setting: *i.e.,* we do not assume that targets possess trusted hardware. Based on this discussion, we formalize the roles and protocol interface of an Abuse-Resistant Law Enforcement Access System (ARLEAS): a message transmission framework that possesses law enforcement access capability with strong accountability guarantees. Finally, we provide an ideal functionality $\mathcal{F}_{\text{ARLEAS}}$ in Canetti's Universal Composability framework [Can01].
- **A prospective ARLEAS construction from lossy encryption or non-interactive secure computation.** We show how to realize ARLEAS that is restricted to the case of *prospective* access: this restricts the use of ARLEAS such that law enforcement must identify surveillance parameters before a target communication occurs. We first show that this can be constructed efficiently using lossy encryption and efficient simulation sound NIZKs, with the limitation that warrants must explicitly specify the *identities* of users being targeted for surveillance. We then show a generalization of this construction such that warrants can be *arbitrary predicates* to be evaluated over each message metadata; our construction of this generalization relies on non-interactive secure computation [IKO$^+$11].

– **A retrospective ARLEAS construction from proof-of-publication ledgers and extractable witness encryption.** We show how to realize ARLEAS that admits *retrospective* access, while still maintaining the auditability and detectability requirements of the system. The novel idea behind our construction is to use secure *proof-of-publication ledgers* to condition cryptographic escrow operations. The cryptographic applications of proof-of-publication ledgers have recently been explored (under slightly different names) in several works [CGJ+17,GG17,KGM19,Sca19]. Such ledgers may be realized using recent advances in consensus networking, a subject that is part of a significant amount of research.

– **Evaluating the difficulty of retrospective systems.** Finally, we investigate the *minimal* assumptions for realizing retrospective access in an accountable law enforcement access system. As a concrete result, we present a lower-bound proof that any protocol realizing retrospective ARLEAS implies the existence of an extractable witness encryption scheme for some language L which is related to the ledger functionality and policy functions of the system. While this proof does not imply that all retrospective ARLEAS realizations require extractable witness encryption for general languages (*i.e.,* it may be possible to construct languages that have trivial EWE realizations), it serves as a guidepost to illustrate the barriers that researchers may face in seeking to build accountable law enforcement access systems.

1.1 Towards Abuse Resistance

In this work we consider the problem of constructing secure message transmission protocols with abuse resistant law enforcement access, which can be seen as an extension of secure message transmission as formalized in the UC framework by Canetti [Can01, CKN03]. Before discussing our technical contributions, we present the parties that interact with such a system and discuss several of the security properties we require.

The ARLEAS Setting. An ARLEAS system is comprised of three types of parties:

1. **Users:** Users employ a secure message transmission protocol to exchange messages with other users. From the perspective of these users, this system acts like a normal messaging service, with the additional ability to view public audit log information about the use of warrants on information sent through the system.

2. **Law Enforcement:** Law enforcement parties are responsible for initiating surveillance and accessing encrypted messages. This involves determining the scope of a surveillance request, obtaining a digital warrant, publishing transparency information, and then accessing the resulting data.

3. **Judiciary:** The final class of parties act as a check on law enforcement, determining whether a surveillance request meets the necessary legal requirements. In our system, any surveillance request must be approved by a judge before it is activated on the system. In our model we assume a single judge per system, though in practice this functionality can be distributed.

At setup time an ARLEAS system is parameterized by three functions, which we refer to as the global policy function, $p(\cdot)$, the warrant transparency function, $t(\cdot)$, and the warrant scope check function, $\theta(\cdot)$.[3] The purpose of these functions will become clear as we discuss operation and desired properties below. Finally, our proposals assume the existence of a verifiable, public broadcast channel, such as an append-only ledger. While this ledger may be operated by a centralized party, in practice we expect that such systems would be highly-distributed, *e.g.* using blockchain or consensus network techniques.

ARLEAS Operation. To initiate a surveillance request, law enforcement must first identify a specific class of messages (*e.g.* by metadata or sender/receiver); it then requests a surveillance warrant w from a judge. The judge reviews the request and authorizes or rejects the request. If the judge produces an authorized warrant, law enforcement must take a final step to *activate* the warrant in order to initiate surveillance.

[3] We later introduce a fourth parameterizing function, but omit it here for the clarity of exposition.

This activation process is a novel element of an abuse resistant access scheme, and it is what allows for the detection of misbehavior. To enforce this, we require that activation of a warrant w results in the publication of some information that is viewable by all parties in the system. This information consists of two parts: (1) a proof that the warrant is *permissible* in accordance with the global policy function, *i.e.* $p(w) = 1$, and (2) some transparency data associated with the warrant. The amount and nature of the transparency data to be published is determined by the warrant transparency function $t(w)$. Once the warrant has been activated, and the relevant information has been made public, law enforcement will be able to access any message that is within the scope of the warrant, as defined by the warrant scope check function $\theta(w)$.

ARLEAS Properties. For a law enforcement access system to be considered *abuse resistant*, it must satisfy the following intuitive properties:

- **Messages Secure without a Warrant.** A system must provide strong cryptographic security against attackers who are not authorized to receive messages, and law enforcement can only access the message if a judge has issued an applicable warrant.
- **Global Surveillance Policies.** Even in cases where all escrow authorities (*i.e.* law enforcement and judges) collude, surveillance requests must always obey a set of global limits defined by the *global policy function* which was chosen during system setup.
- **Abuse Detectability.** To enable detection of abuse or theft of key material, we require that whenever a warrant w is activated, law enforcement must publish $t(w)$ to all parties in the system, where $t(\cdot)$ is a *warrant transparency function* defined at system setup. This publication must occur even in cases where all escrow authorities collude.
- **Target Anonymity.** To preserve the integrity of investigations, users should learn no information about the contents of a warrant beyond what is revealed by the transparency function.
- **Escrow Verifiability.** Escrow authorities must be assured that the access system will decrypt messages within the scope of an activated, valid warrant. Because senders can always behave dishonestly (*e.g.* using alternative encryption mechanisms or encode messages using steganography), this guarantee cannot be enforced for all possible sender behavior. Instead, we mandate a weaker property that we call *escrow verifiability*: this ensures that recipients and/or service providers can filter messages that will decrypt differently for receivers and law enforcement. This ensures that *compliant messages* will be accessible by escrow authorities under appropriate circumstances.

In Section 3 we formalize this intuition and present a concrete security definition for an ARLEAS.

1.2 Technical Overview

We now present an overview of the key technical contributions of this work. We will consider this in the context of secure message transmission systems, which can be generalized to the setting of encrypted storage. Our overview will begin with intuition for building prospective ARLEAS, and then we will proceed to retrospective ARLEAS.

Accountability From Ledgers. For an ARLEAS the most difficult properties to satisfy are accountability and detectability. Existing solutions attempt to achieve this property by combining auditors and key escrow custodians; in order to retrieve key material that facilitates decryption, law enforcement must engage with an auditor. This solution, however, does not account for dishonest authorities, and is therefore vulnerable to covert key exfiltration and collusion. In our construction, we turn to public ledgers — a primitive that can be realized using highly-decentralized and auditable systems — as a way to reduce these trust assumptions.

Ledgers have the property that any party can access their content. Importantly, they also have the property that any parties can be convinced that other parties have access to these contents. Thus, if auditing information is posted on a ledger, all parties are convinced that that information is truly public. We note that using ledgers in this way is fundamentally different from prior work addressing encrypted

communications; our ledger is a public functionality that does not need to have any escrow secrets. As such, if it is corrupted, there is no private state that can be exploited by an attacker.

Warm up: Prospective ARLEAS. To build to our main construction, we first consider the simpler problem of constructing a *prospective* access system, one that is capable of accessing messages that are sent subsequent to a warrant being activated. For our practical construction, we make a further simplifying assumption that law enforcement will target *specific parties* for surveillance. We then extend this paradigm to allow law enforcement to target messages using arbitrary predicates.

A key aspect of this construction is that we consider a relatively flexible setting where parties have network access, and can receive periodic communications from escrow system operators prior to transmitting messages. We employ a public ledger for transmission of these messages, which provides an immutable record as well as a consistent view of these communications. The goal in our approach is to ensure that escrow updates embed information about the specifics of surveillance warrants that are active, while ensuring that even corrupted escrow parties cannot abuse the system.

Escrow lossy encryption. The basic intuition of our approach is to construct a "dual-trapdoor" public-key encryption system [BCP03] that senders can use to encrypt messages to specific parties. This scheme is designed with two ciphertexts c_1 and c_2, such that c_1 can be decrypted by the intended recipient using a normal secret key, while c_2 can be decrypted by law enforcement only if the recipient is under active surveillance. A feature of this scheme is that for all recipients not the target of surveillance, c_2 should contain no information about the plaintext.

Lossy encryption [PW08] is a natural tool to use to encrypt c_2, as an injective key can be used to encrypt c_2 when the recipient is under surveillance (preserving the plaintext) and a lossy key can be used to encrypt c_2 when the recipient is not under surveillance (destroying any information about the plaintext). A naive solution would have law enforcement generate either an injective key or a lossy key information for each user, as only law enforcement knows which users are under surveillance. Instead, we realize a more efficient construction using a tag-based variant of lossy encryption [PW08,BHY09,HLOV11,Hof12] that we call *none-but-N* lossy-tag-based encryption, or LTE. In this scheme, key generation creates public parameters mpk with respect to the set of user identifiers (tags) $\mathcal{T}$ that are under active surveillance, along with a secret decryption key. The public parameters are proportional in size to the number of users under surveillance. When encrypting a message, a sender encrypts under both the recipient's public key and tag, along with mpk.

An LTE scheme must satisfy three main security properties. First, if the parameter generation process is run honestly with some set of user identifiers $\mathcal{T}$ and any (even biased) random coins r, then even an adversarial law enforcement should not be able to retrieve the message m if the receiver is not in $\mathcal{T}$. Second, to ensure that law enforcement access is possible, we require that adversarial encryptors cannot produce a ciphertext that appears correctly formatted but does not admit decryption. Finally, we require that mpk must at least computationally hide the set of users that are being targeted for surveillance, *i.e.* no efficient adversary with mpk should be able to recover any information about $\mathcal{T}$, beyond the size of its description. In Section 4.4 we discuss candidate constructions for LTE schemes based on the lossy encryption scheme in [BHY09].

Building prospective ARLEAS for identities from LTE. Given an appropriate lossy tag-based encryption scheme, the remainder of the ARLEAS construction proceeds as follows. The global parameters of the scheme are created at setup: these include a public verification key for the judge presiding over the system, as well as a global transparency function t and policy function p agreed on by system participants.

When a law enforcement agency wishes to add a user to those being surveilled, it creates a new warrant w embedding the tag of that user and adds it to the set of active warrants $\mathcal{W}$. Law enforcement then runs the LTE parameter generation algorithm on the set of tags $\mathcal{T}$ embedded in the warrants in $\mathcal{W}$, obtaining mpk and the corresponding secret key. Law enforcement next contacts the judge to obtain a signature over w, and proceeds to generate a NIZK π that the following statements hold: (1) the prover posses a signature from the judge for each warrant in $\mathcal{W}$, (2) the parameter mpk was correctly generated with respect to the user-set $\mathcal{T}$ specified in the warrants in $\mathcal{W}$, (3) each warrant $w \in \mathcal{W}$ is permissible according to the global

policy function p, and (4) the transparency information $\mathsf{info} \leftarrow \{t(w), \forall w \in \mathcal{W}\}$ was calculated honestly. Finally, it transmits $(mpk, \mathsf{info}, \pi)$ to a global ledger. Each participant in the system must ensure that this message was correctly published, and verify the proof π. If this proof verifies correctly, the participants will accept the new parameter mpk and use this value for all subsequent encryptions.

A critical security property of this system is that, even if law enforcement and judges collude (*e.g.,* if both parties become catastrophically compromised), users retain the assurance that issued warrant in violation of the global policy p cannot be used. Moreover, even in this event, the publication of a transparency record info ensures that every warrant activated in the system produces a detectable artifact that can be used to identify abuse.[4]

Prospective ARLEAS for Arbitrary Predicates. The solution presented above is inherently identity based, which restricts the types of warrants that a judge is able to issue. We now describe a version of this system that facilitates law enforcement access to messages if they possess a valid warrant embedding an *arbitrary predicate* such that this predicate, evaluated over the message metadata, is satisfied. Unlike the identity-based solution, generating key material for each possible situation cannot work; while the number of identities in the system may be bounded, there are an exponential number of predicate functions that law enforcement might want to embed into warrants. Instead, we rely on non-interactive secure computation (NISC) [IKO+11], a reusable, non-interactive version of two-party computation. NISC for an arbitrary function f allows a receiver to post an encryption of some secret x_1 such that all players can reveal $f(x_1, x_2)$ to the receiver with only one message, without revealing anything about x_2 beyond the output of the function. We leverage such a scheme to have senders reveal the message plaintext to law enforcement if and only if law enforcement's input to the NISC scheme contains a valid, pertinent warrant.

As before, law enforcement computes the transparency information for their warrant $\mathsf{info} \leftarrow t(w)$ along with the first message of the NISC scheme, embedding the warrant. Both of these are posted onto the ledger, along with a non-interactive zero-knowledge proof of correctness and compliance with the policy function. Whenever a sender sends a message m, they generate c_1 as normal and then generate c_2 which, using the NISC scheme, allows law enforcement to compute

$$f(w, (m, \mathsf{meta})) = m \wedge \theta(\mathsf{meta}, w),$$

where $\theta(\cdot, \cdot)$ evaluates if the warrant applies to this particular message (we will discuss $\theta(\cdot, \cdot)$ in more detail in Section 3). Notice that if $\theta(\mathsf{meta}, w) = 0$, then the output of the NISC evaluation is uncorrelated with the message. However, if $\theta(\mathsf{meta}, w) = 1$, meaning law enforcement has been issued a valid warrant, then the message is recovered. As before, users are assured that all activated warrants are acceptable according to the policy function and detectable artifacts must be generated before any messages can be decrypted.

From Prospective to Retrospective. The major limitation of the ARLEAS construction above is that it is fundamentally restricted to the case of *prospective* access. Abuse resistance derives from the fact that "activation" of a warrant results in a distribution of fresh encryption parameters to users, and each of these updates renders only a subset of communications accessible to law enforcement. A second drawback of the prospective protocol is that it requires routine communication between escrow authorities and the users of the system, which may not be possible in all settings.

Updating these ideas to provide *retrospective* access provides a stark illustration of the challenges that occur in this setting. In the retrospective setting, the space of targeted communications is unrestricted at the time that encryption takes place. By the time this information is known, both sender and recipient may have completed their interaction and gone offline. Using some traditional, key-based solution to this problem implies the existence of powerful master decryption keys that can access *every* ciphertext sent by users of the system. Unfortunately, granting such power to any party (or set of parties) in our system is untenable; if this key material is compromised, any message can be decrypted without leaving a detectable artifact. The technical challenge in the retrospective setting is to find an alternative means to enable decryption, such that decryption is only possible on the conditions that (1) a relevant warrant has been issued that is

[4]The flexible nature of the transparency function t ensures that these records can contain both publicly-visible records (*e.g.,* a quantized description of the user set size, as well as private information that can be encrypted to auditors.

compliant with the global policy function, (2) a detectable artifact has been made public. This mechanism must remain secure even when encryption occurs significantly before the warrant is contemplated.

Ledgers as a cryptographic primitive. A number of recent works [CGJ+17,GG17,KGM19,CGJ19,Sca19] have proposed to use public ledgers as a means to *condition* cryptographic operations on published events. This paradigm was initially used by Choudhuri *et al.* [CGJ+17] to achieve fairness in MPC computations, while independently a variant was proposed by Goyal and Goyal [GG17] to construct one-time programs without the need for trusted hardware. Conceptually, these functionalities all allow decryption or program execution to occur only *after* certain information has been made public. This model assumes the existence of a secure global ledger $\mathcal{L}$ that is capable of producing a publicly-verifiable proof π that a value has been made public on the ledger. In principle, this ledger represents an alternative form of "trusted party" that participates in the system. However, unlike the trusted parties proposed in past escrow proposals [Den94], ledgers do not store any decryption secrets. Moreover, recent advances in consensus protocols, and particularly the deployment of proof-of-work and proof-of-stake cryptocurrency systems. *e.g.,* [KRDO17, DGKR18, GKZ19, BBBF18], provide evidence that these ledgers can be operated safely at large scale.

Following the approach outlined by Choudhuri *et al.* [CGJ+17], we make use of the ledger to *conditionally encrypt* messages such that decryption is only possible following the verifiable publication of the transparency function evaluated over a warrant on the global ledger. For some forms of general purpose ledgers that we seek to use in our system, this can be accomplished using extractable witness encryption (EWE) [BCP14].[5] EWE schemes allow a sender to encrypt under a statement such that decryption is possible only if the decryptor knows of a witness ω that proves that the statement is in some language L, where L parameterizes the scheme. While candidate schemes for witness encryption are known for specific languages (*e.g.* hash proof systems [CS02, GOVW12]), EWE for general languages is unlikely to exist [GGHW14].

Building Retrospective ARLEAS from EWE. Our retrospective ARLEAS construction assumes the existence of a global ledger that produces verification proofs π_{publish} that a warrant has been published to a ledger. As mentioned before, we aim to condition law enforcement access on the issuance of a valid warrant and the publication of a detectable artifact. In a sense, we want to use this published detectable artifact as a key to decrypt relevant ciphertexts. Thus, in this construction, a sender encrypts each message under a statement with a witness that shows evidence that these conditions have been met. This language reasons over (1) the warrant transparency function, (2) a function determining the relevance of the warrant to ciphertext, (3) the global policy function, (4) the judge's warrant approval mechanism, and (5) the ledger's proof of publication function.

On the Requirement of EWE. We justify the use of EWE in our construction by showing that the existence of a secure protocol realizing retrospective ARLEAS implies the existence of a secure EWE scheme for a related language that is deeply linked to the ARLEAS protocol. Intuitively, the witness for this language should serve as proof that the protocol has been correctly executed; law enforcement should be able to learn information about a message if and only if the accountability and detectability mechanisms have been run. For the concrete instantiation of retrospective ARLEAS, we give in Section 6, this would include getting a valid proof of publication from the ledger. If the protocol is realized with a different accountability mechanism, the witness encryption language will reason over that functionality. No matter the details of the accountability mechanism, we note that it should be difficult for law enforcement to locally simulate the mechanism. If it were computationally feasible, then law enforcement would be able to circumvent the accountability mechanism with ease.

1.3 Contextualizing ARLEAS In The Encryption Debate

This work is motivated by the active global debate on whether to mandate law enforcement access to encrypted communication systems via key escrow. Reduced to its essentials, this debate incorporates two

[5] Using the weaker witness encryption primitive may be possible if the ledger produces *unique* proofs of publication.

broad sub-questions. First: can mandatory key escrow be deployed safely? Secondly, if the answer to the first question is positive: *should it be deployed?*

We do not seek to address the second question in this work. Many scholars in the policy and technical communities have made significant efforts in tackling this issue [Nat18, Bar19, AAB+15, Enc19] and we do not believe that this work can make a substantial additional contribution. We stress, therefore, that our goal in this work is not to propose techniques for real-world deployment. Numerous practical questions and technical optimizations would need to be considered before ARLEAS could be deployed in practice.

Instead, the purpose of this work is to provide data to help policymakers address the first question. We have observed a growing consensus among stakeholders that key escrow systems should provide strong guarantees of information security as a precondition for deployment. Some stakeholders in the law-enforcement and national security communities grant that key escrow systems *should not be deployed* unless they can mitigate the risk of mass-surveillance via system abuse or compromise.[6] Unfortunately, there is no agreement on the definition of safety, and the technical community remains divided on whether traditional key escrow security measures (such as the use of secure hardware, threshold cryptography and policy safeguards) will be sufficient. We believe that the research community can help to provide answer these questions, and a failure to do so will increase the risk of unsound policy.

Our contribution in this paper is therefore to take a first step towards this goal. We attempt to formalize a notion of abuse-resilient key escrow, and to determine whether it can be realized using modern cryptographic techniques. Our work is focused on *feasibility*. With this perspective in mind, we believe that our work makes at least three necessary contributions to the current policy debate:

Surface the notion of cryptographic abuse-resistance. We raise the question of whether key escrow can be made *abuse resistant* using modern cryptographic technologies, and investigate what such a notion would imply. A key aspect of this discussion is the question of detectability: by making abuse and key exfiltration publicly detectable, we can test law enforcement's belief that backdoor secrets can remain secure, and renew security by efficiently re-keying the system.

Separate the problems of prospective and retrospective surveillance. By emphasizing the technical distinctions between prospective and retrospective surveillance, we are able to highlight the design space in which it is realistic to discuss law enforcement access mechanisms. In particular, our technical results in this work illustrate the cryptographic implausibility of retrospective ARLEAS: this may indicate that retrospective surveillance systems are innately susceptible to abuse.

Shift focus to public policy. In defining and providing constructions for prospective and retrospective ARLEAS, we formalize the notion of a global policy function and a transparency function (see Section 3). By making these functions explicit, we hope to highlight the difficult policy issues that must be solved before deploying any access mechanism. As noted by Feigenbaum and Weitzner [FW18], there are limits what cryptography can contribute to this debate; legal and policy experts must do a better job reducing the gray area between rules and principles so that technical requirements can be better specified.

Finally, we note that the existence of a cryptographic construction for ARLEAS may not be sufficient to satisfy lawn enforcement needs. The mathematics for cryptographically strong encryption systems is already public and widespread, and determined criminals may simply implement their own secure messaging systems [Enc]. Alternatively, they may use steganography or pre-encrypt their messages with strong encryption to prevent "real" plaintext from being recovered by law enforcement while still allowing contacts to read messages [HPRV19]. These practical problems will likely limit the power of any ARLEAS and must be considered carefully by policy makers before pushing for deployment.

[6] For evidence of this consensus, see *e.g.,* the 2018 National Academies of Sciences Report [Nat18], which provides a framework for discussing such questions. See also a recent report by the Carnegie Endowment [Enc19] which chooses to focus only on the problem of escrow for physical devices rather than data in motion, providing the following explanation: "it is much harder to identify a potential solution to the problems identified regarding data in motion in a way that achieves a good balance" (p. 10).

2 Related work

The past decade has seen the start of academic work investigating the notion of accountability for government searches. Bates *et al.* [BBS+12] focus specifically on CALEA wiretaps and ensuring that auditors can ensure law enforcement compliance with court orders. In the direct aftermath of the Snowden leaks, Segal *et al.* [SFF14] explored how governments could accountably execute searches without resorting to dragnet surveillance. Liu *et al.* [LRC14] focus on making the number of searches more transparent, to allow democratic processes to balance social welfare and individual privacy. Kroll *et al.* [KFB14, KZW+] investigate different accountability mechanisms for key escrow systems, but stop short of addressing end-to-end encryption systems and the collusion problems we address in this work. Kamara [Kam14] investigates cryptographic means of restructuing the NSA's metadata program. Backes considered anonymous accountable access control [BCS05], while Goldwasser and Park [GP17] investigate similar notions with the limitation that policies themselves may be secret, due to national security concerns. Frankle *et al.* [FPS+18] make use of ledgers to get accountability for search procedures, but their solution cannot be extended to the end-to-end encryption setting. Wright and Varia [WV18] give a construction that uses cryptographic puzzles to impose a high cost for law enforcement to decrypt messages. Servan-Schreiber and Wheeler [SSW19] give a construction for accountability that randomly selects custodians that law enforcement must access to decrypt a message. Panwar *et al.* [PVMB19] attempt to integrate the accountability systems closely with ledgers, but do not use the ledgers to address access to encryption systems. Finally, Scafuro [Sca19] proposes a closely related concept of "break-glass encryption" and give a construction that relies on trusted hardware.

3 Definitions

Notation. Let λ be an adjustable security parameter and $\mathsf{negl}(\lambda)$ be a negligible function in λ. We use $\|$ to denote concatenation, $\overset{c}{\approx}$ to denote computational indistinguishability, and $\overset{s}{\approx}$ to denote statistical indistinguishability. We will write $x \leftarrow \mathsf{Algo}(\cdot)$ to say that x is a specific output of running the algorithm Algo on specific inputs and will write $x \in \mathsf{Algo}(\cdot)$ to indicate that x is an element in the output distribution of Algo, when run with honest random coins. We write $\mathsf{Algo}^{\mathsf{Par}}$ to say that the algorithm Algo is parameterized by the algorithm Par.

Defining ARLEAS. We now formally define the notion of an Abuse-Resistant Law Enforcement Access System (ARLEAS). An ARLEAS is a form of message transmission scheme that supports accountable access by law enforcement officials. To emphasize the core functionality, we base our security definitions on the UC Secure Message Transmission ($\mathcal{F}_{\mathsf{SMT}}$) notion originally introduced by Canetti [Can01]. Indeed, our systems can be viewed as an extension of a multi-message SMT functionality [CKN03], with added escrow capability.

Parties and system parameters. An ARLEAS is an interactive message transmission protocol run between several parties and network components:

- User P_i: Users are the primary consumer of the end-to-end encrypted service or application. These parties, which may be numerous, interact with the system by sending messages to other users.
- Judge P_J: The judge is responsible for determining the validity of a search and issuing search warrants to law enforcement. The judge interacts with the system by receiving warrant requests and choosing to deny or approve the request.
- $\mathsf{LawEnforcement}$ P_{LE}: Law enforcement is responsible for conducting searches pursuant to valid warrants authorized by a judge. Law enforcement interacts with the system by requesting warrants from the judge and collecting the plaintext messages relevant to their investigations.

A concrete ARLEAS system also assumes the existence of a communication network that parties can use to transmit encrypted messages to other users. To support law enforcement access, it must be possible for law enforcement to "tap" this network and receive encrypted communications between targeted users. For the

purposes of this exposition, we will assume that law enforcement agents have access to any communications transmitted over the network (*i.e.,* the network operates as a transparent channel.) In practice, a service provider would handle the transmissions of ciphertexts. This service provider would also be responsible for storing ciphertext and metadata, and providing this information to law enforcement. Our simplified model captures the worst case network security assumption, where the service provider cooperates with all law enforcement requests. Service providers would also be responsible for checking that messages sent by users are compliant with the law enforcement access protocol. We move this responsibility to the receiver for simplicity. We discuss the role of service providers more in Appendix D.

An ARLEAS system is additionally parameterized by four functions, which are selected during a trusted setup phase:

- $t(w)$: the deterministic *transparency function* takes as input a warrant w and outputs specific information about the warrant that can be published to the general public.
- $p(w)$: the deterministic *global policy* function takes as input a warrant w and outputs 1 if this warrant is allowed by the system.
- $\theta(w, \mathsf{meta})$: the deterministic *warrant scope check* takes as input a warrant w and per-message metadata meta. It outputs 1 if meta is in scope of w for surveillance.
- $v(\mathsf{meta}, aux)$: The deterministic *metadata verification functionality* takes as input metadata associated with some message meta and some auxiliary information aux and determines if the metadata is correct. This auxiliary information could contain the ciphertext, global timing information, or some authenticated side channel information.

We discuss concrete instantiations of these functions in Appendix D.

ARLEAS scheme. An ARLEAS scheme comprises a set of six possibly interactive protocols. We provide a complete API specification for these protocols in later sections:

- Setup. On input a security parameter, this trusted setup routine generates all necessary parameters and keys needed to run the full system.
- SendMessage. On input a message m, metadata meta, and a recipient identity, this protocol sends an encrypted message from one party to another.
- RequestWarrant. On input a description of the warrant request, this procedure allows law enforcement to produce a valid warrant.
- ActivateWarrant. Given a warrant w, this protocol allows law enforcement and a judge to confirm and activate a warrant.
- VerifyWarrantStatus. Given a warrant w, this protocol is used to verify that a warrant is valid and active.
- AccessMessage. in the retrospective case, this protocol is used by law enforcement to open a message.

UC ideal functionality. To define the properties of an ARLEAS system, we present a formal UC ideal functionality $\mathcal{F}_{\text{ARLEAS}}$ in Figure 1. Recalling that ARLEAS can be instantiated in one of two modes, supporting only prospective or retrospective surveillance, we present a single definition that supports a parameter, $\mathsf{mode} \in \{\mathsf{pro}, \mathsf{ret}\}$.

Ideal World. For any ideal-world adversary $\mathcal{S}$ with auxiliary input $z \in \{0,1\}^*$, input vector x, and security parameter λ, we denote the output of the ideal world experiment by $\mathbf{Ideal}_{\mathcal{S},\mathcal{F}_{\text{ARLEAS}}^{v,t,p,\theta,\mathsf{mode}}}(1^\lambda, x, z)$.

Real World. The real world protocol starts with the adversary $\mathcal{A}$ selecting a subset of the parties to compromise $\mathcal{P}^{\mathcal{A}} \subset \mathcal{P}$, where $\mathcal{P}^{\mathcal{A}} \subset \{\{P_i\}, \{P_{\text{LE}}\}, \{P_{\text{LE}}, P_J\}\}$, where we denote sender with P_i and receiver with

Functionality $\mathcal{F}_{\text{ARLEAS}}^{v,t,p,\theta,\mathsf{mode}}$

The ideal functionality is parameterized by $\mathsf{mode} \in \{\mathsf{pro}, \mathsf{ret}\}$, a metadata verification function $v : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}$, the transparency function $t(\cdot)$, the global policy function $p(\cdot)$, and the warrant scope check functionality $\theta(\cdot,\cdot)$. The three latter functions are as defined above. We denote the session identifier as sid to separate different runs of the same protocol. We have several parties:

- $P_1, \ldots, P_n$: participants in the system
- P_J: the generator of a warrant
- P_{LE}: Law enforcement that can read the message given a valid warrant

Send Message: Upon receiving a message $(\mathsf{SendMessage}, \mathsf{sid}, P_j, m, \mathsf{meta}, \mathsf{valid})$ where $\mathsf{valid} \in \{0,1\}$ from party P_i, it sends $(\mathsf{Sent}, \mathsf{sid}, \mathsf{meta})$ to the adversary. If (sid, c) is received from the adversary,

- If $\mathsf{valid} = 0$ or $v(\mathsf{meta}, aux) = 0$, send $(\mathsf{Sent}, \mathsf{sid}, \mathsf{meta}, c, m)$ to P_i and send $(\mathsf{Sent}, \mathsf{sid}, \mathsf{meta}, c, 0)$ to P_{LE}.
- If $\mathsf{valid} = 1$, $v(\mathsf{meta}, aux) = 1$, and there is no entry w in the active warrant table W_{active} send $(\mathsf{Sent}, \mathsf{sid}, \mathsf{meta}, c, m)$ to P_i and P_j, and send $(\mathsf{Sent}, \mathsf{sid}, \mathsf{meta}, c)$ to P_{LE}.
- If $\mathsf{valid} = 1$, $v(\mathsf{meta}, aux) = 1$, and there is an entry w in the active warrant table W_{active} send $(\mathsf{Sent}, \mathsf{sid}, \mathsf{meta}, c, m)$ to P_i, P_j, and P_{LE}.

Finally, store $(\mathsf{Sent}, \mathsf{sid}, \mathsf{meta}, c, m)$ in the message table M.

Request Warrant: Upon receiving a message $(\mathsf{RequestWarrant}, \mathsf{sid}, w)$ from P_{LE}, the ideal functionality first checks if $p(w) = 1$, responding with $\perp$ and aborting if not. Otherwise, the ideal functionality sends $(\mathsf{ApproveWarrant}, w)$ to P_J. If P_J responds with $(\mathsf{Disapprove})$, the trusted functionality sends $\perp$ to P_{LE}. If P_J responds with $(\mathsf{Approve})$, the trusted functionality sends $(\mathsf{Approve})$ to P_{LE}, and stores the entry w in the issued warrant table W_{issued}.

Activate Warrant: Upon receiving a message $(\mathsf{ActivateWarrant}, \mathsf{sid}, w)$ from P_{LE}, the ideal functionality checks to see if $w \in W_{\text{issued}}$, responding with $\perp$ and aborting if not. If $w \in W_{\text{issued}}$, the trusted functionality adds the entry w to the active warrant table W_{active}, computes $t(w)$, and sends $(\mathsf{NotifyWarrant}, t(w))$ to all parties and the adversary.

Verify Warrant Status: Upon receiving message $(\mathsf{VerifyWarrantStatus}, \mathsf{sid}, c, \mathsf{meta}, w)$ from P_{LE}, if $\mathsf{mode} = \mathsf{pro}$, the ideal functionality responds with $\perp$ and aborts. Otherwise, if $(\mathsf{Sent}, \mathsf{sid}, \mathsf{meta}, c, m) \in M$ and $w \in W_{\text{active}}$ such that $\theta(w, \mathsf{meta}) = 1$, the ideal functionality returns 1. Finally, if $\theta(w, \mathsf{meta}) = 0$ or $w \notin W_{\text{active}}$, it returns 0.

Access message: Upon receiving message $(\mathsf{AccessData}, \mathsf{sid}, c, \mathsf{meta}, w)$ from P_{LE}, if $\mathsf{mode} = \mathsf{pro}$, the ideal functionality responds with $\perp$ and aborts. Otherwise, if $(\mathsf{Sent}, \mathsf{sid}, \mathsf{meta}, c, m) \in M$ and $w \in W_{\text{active}}$ such that $\theta(w, \mathsf{meta}) = 1$, the ideal functionality returns m. Finally, if $\theta(w, \mathsf{meta}) = 0$ or $w \notin W_{\text{active}}$, it returns 0.

Figure 1: Ideal functionality for an Abuse Resistant Law Enforcement Access System.

Protocol $\mathbf{Real}_{\mathcal{A},\Pi}(1^\lambda, x, z)$

$\mathbf{Real}_{\mathcal{A},\Pi}(1^\lambda, x, z)$ is parameterized by the protocol $\Pi = (\mathsf{Setup}, \mathsf{SendMessage}, \mathsf{RequestWarrant}, \mathsf{ActivateWarrant}, \mathsf{VerifyWarrantStatus}, \mathsf{AccessMessage})$ and a variable $\mathsf{mode} \in \{\mathsf{pro}, \mathsf{ret}\}$.

1. When $\mathbf{Real}_{\mathcal{A},\Pi}(1^\lambda, x, z)$ is initialized, then all parties engage in the interactive protocol $\Pi.\mathsf{Setup}$
2. When P_i is activated with $(\mathsf{SendMessage}, \mathsf{sid}, P_j, m, 1)$, parties P_i, P_j, and P_{LE} engage in the interactive protocol $\Pi.\mathsf{SendMessage}$. P_{LE} learns some metadata meta about the message.
3. When P_i is activated with $(\mathsf{SendMessage}, \mathsf{sid}, P_j, m, 0)$, parties P_i, and P_{LE} engage in the interactive protocol $\Pi.\mathsf{SendMessage}$ (with P_j not getting output). P_{LE} learns some metadata meta about the message.
4. When P_{LE} is activated with $(\mathsf{RequestWarrant}, \mathsf{sid}, \hat{w})$, parties P_{LE} and P_J engage in the interactive protocol $\Pi.\mathsf{RequestWarrant}$.
5. When P_{LE} is activated with $(\mathsf{ActivateWarrant}, \mathsf{sid}, w)$, all parties engage in the interactive protocol $\Pi.\mathsf{ActivateWarrant}$.
6. When P_{LE} is activated with $(\mathsf{VerifyWarrantStatus}, \mathsf{sid}, c, \mathsf{meta}, w)$, if $\mathsf{mode} = \mathsf{pro}$, P_{LE} returns $\perp$. Otherwise, P_{LE} calls the non-interactive functionality $\Pi.\mathsf{VerifyWarrantStatus}(c, \mathsf{meta}, w)$
7. When P_{LE} is activated with $(\mathsf{AccessData}, \mathsf{sid}, c, \mathsf{meta}, w)$, if $\mathsf{mode} = \mathsf{pro}$, P_{LE} returns $\perp$. Otherwise, P_{LE} calls the non-interactive functionality $\Pi.\mathsf{AccessMessage}(c, \mathsf{meta}, w)$

Figure 2: The real world experiment for a protocol implementing $\mathcal{F}_{\text{ARLEAS}}^{v,t,p,\theta,\mathsf{mode}}$

P_j. We limit the subsets of parties that can be compromised to these cases, because any other combination is trivial to simulate or can be deducted from the other cases. For example, if both P_i and P_j would be corrupted, there is nothing stopping them from not using the system. Moreover, we also don't consider the case where P_J is the only corrupted party, this case is a more specific then when both P_{LE} and P_J are corrupted and P_J on its own doesn't have any additional information to achieve anything different. All parties engage in a real protocol execution Π, the adversary $\mathcal{A}$ sends all messages on behalf of the corrupted parties and can choose any polynomial time strategy.

In a real world protocol we assume that communication between a sender P_i and receiver P_j happens over a transparent channel, meaning all other parties are able to receive all communication. We make this choice to simplify the protocol and security proofs. In the real world, this can be modeled with a service provider relaying messages between P_i and P_j that always complies with law enforcement requests and hands over encrypted messages when presented with a valid warrant. Note that this makes our modeling the worst case scenario, and therefore captures more selective service providers. Additionally, in practice, this service provider would validate if messages are well-formed to make sure P_i and P_j follow the real protocol.

For any adversary $\mathcal{A}$ with auxiliary input $z \in \{0,1\}^*$, input vector x, and security parameter λ, we denote the output of Π by $\mathbf{Real}_{\mathcal{A},\Pi}(1^\lambda, x, z)$.

Definition 1 *A protocol Π is said to be a secure ARLEAS protocol computing $\mathcal{F}_{ARLEAS}^{v,t,p,\theta,\mathsf{mode}}$ if for every PPT real-world adversary $\mathcal{A}$, there exists an ideal-world PPT adversary $\mathcal{S}$ corrupting the same parties such that for every input x and auxiliary input z it holds that*

$$\mathbf{Ideal}_{\mathcal{S},\mathcal{F}_{ARLEAS}^{v,t,p,\theta,\mathsf{mode}}}(1^\lambda, x, z) \overset{c}{\approx} \mathbf{Real}_{\mathcal{A},\Pi}(1^\lambda, x, z)$$

4 Building Blocks

4.1 Proof-of-publication ledgers

Our work makes use of a public append-only ledger that can produce a publicly-verifiable *proof of publication*. This concept was formalized by Goyal *et al.* [GG17], Choudhuri *et al.* [CGJ+17], and Kaptchuk *et al.* [KGM19], but related ideas have also been previously used by Liu *et al.* to realize time-lock encryption [LJKW18]. Plausible candidates for such ledgers have been the subject of great interest, due to the deployment of blockchains and other consensus networks [Nak08]. Significant work has been done to formalize the notion of a public, append-only ledger [CGJ19, BGK+18, BMTZ17, GG17] and study its applications to cryptographic protocols [ADMM14, BK14, CGJ+17]. This work uses a simplified ledger interface formalized in [CGJ+17] that abstracts away details such as timing information and temporary inconsistent views that are modeled in [BMTZ17]. However, this simplified view captures the *eventual* functionality of the complex models, and is therefore equivalent for our purposes.

The ledger ideal functionality is provided in Figure 3. This functionality allows users to post arbitrary information to the ledger; this data is associated with a particular index on the ledger, with which any user can retrieve the original data as well as a proof of publication. For security, our functionality encodes a notion we refer to as *ledger unforgeability*, which requires that there exists an algorithm to verify a proof that a message has been posted to the ledger, and that adversaries cannot forge this proof.

Functionality $\mathcal{L}^{\mathsf{Verify}}$

GetCounter: Upon receiving $(\mathsf{GetCounter})$ from any party, return ℓ.

Post: Upon receiving (Post, x), the trusted party increments ℓ by 1, computes the proof of publication π_{publish} on $(\ell||x)$ such that $\mathsf{Verify}((\ell||x), \pi_{\mathsf{publish}}) = 1$. Add the entry $(\ell, x, \pi_{\mathsf{publish}})$ to the entry table T. Respond with $(\ell, x, \pi_{\mathsf{publish}})$

GetVal: Upon receiving (GetVal, ℓ), check if there is an entry $(\ell, x, \pi_{\mathsf{publish}})$ in the entry table T. If not, return $\perp$. Otherwise, return $(\ell, x, \pi_{\mathsf{publish}})$.

Figure 3: Ideal functionality for a proof-of-publication ledger, from [CGJ+17].

4.2 Authenticated Communication

We use a variant of Canetti's ideal functionality for authenticated communication, $\mathcal{F}_{AUTH}$, to abstract the notion of message authentication [Can01]. This is presented in Figure 4. Since we restrict our analysis to static corruption, we simplify this functionality to remove the adaptive corruption interface.[7]

4.3 Simulation Extractable Non-Interactive Zero Knowledge

In our protocols we require non-interactive zero knowledge proofs of knowledge that are simulation extractable. To preserve space, we refer the reader to the definitions of Sahai [Sah99] and De Santis *et al.* [DDO+01]. Rather than rely on UC functionalities, we employ a NIZK directly in our protocols.

Definition 2 (Simulation Extractable Non-Interactive Zero Knowledge) A non-interactive zero-knowledge proof of knowledge for a relation $\mathcal{R}$ is a set of algorithms $(\mathsf{ZKSetup}, \mathsf{ZKProve}, \mathsf{ZKVerify}, \mathsf{ZKSimulate})$ defined as follows:

[7] Note that this ideal functionality only handles a single message transfer, but to achieve multiple messages, we rely on universal composition and use multiple instances of the functionality.

Functionality $\mathcal{F}_{AUTH}$

Sending: Upon receiving an input $(\mathsf{Send}, \mathsf{sid}, P_j, m)$ from P_i, if it is *not* the first instance of $(\mathsf{Send}, \cdot, \cdot, \cdot)$, the ideal functionality does nothing. Otherwise, it sends $(\mathsf{Sent}, \mathsf{sid}, P_i, P_j, m)$ to the adversary. If the adversary responds with (OK), then the ideal functionality sends $(\mathsf{Sent}, \mathsf{sid}, P_i, P_j, m)$ and halts. If the adversary does not approve, the ideal functionality drops the message and halts.

Figure 4: The message authentication ideal functionality $\mathcal{F}_{AUTH}$ supporting static corruption, adapted from [Can01].

- $\mathsf{ZKSetup}(1^\lambda)$ returns the common reference string and simulation trapdoor $(\mathsf{CRS}_{ZK}, \tau)$.
- $\mathsf{ZKProve}(\mathsf{CRS}_{ZK}, x, \omega)$ takes in the common reference string CRS_{ZK}, a statement x and a witness ω and outputs a proof π.
- $\mathsf{ZKVerify}(\mathsf{CRS}_{ZK}, x, \pi)$ takes in the common reference string CRS_{ZK}, a statement x and a proof π, and outputs either 1 or 0.
- $\mathsf{ZKSimulate}(\mathsf{CRS}_{ZK}, \tau, x)$ takes in the common reference string CRS_{ZK}, a statement x and the simulation trapdoor τ and outputs a proof π.

We say that a non-interactive zero-knowledge argument of knowledge is simulation extractable if it satisfies the following properties:

- **Completeness:** If a prover has a valid witness, then they can always convince the verifier. More formally, for all relations $\mathcal{R}$ and all x, ω, if $\mathcal{R}(x, \omega) = 1$, then

$$\Pr\left[\mathsf{ZKVerify}(\mathsf{CRS}_{ZK}, x, \pi) = 1 \,\middle|\, (\mathsf{CRS}_{ZK}, \tau) \leftarrow \mathsf{ZKSetup}(1^\lambda), \pi \leftarrow \mathsf{ZKProve}(\mathsf{CRS}_{ZK}, x, \omega)\right] = 1$$

- **Perfect Zero knowledge:** A scheme has zero-knowledge if a proof leaks no information beyond that truth of the statement x. We formalize this by saying that an adversary with oracle access to a prover cannot tell if that prover runs the honest algorithm $\mathsf{ZKProve}$ or uses the trapdoor and $\mathsf{ZKSimulate}$.

$$\Pr\left[\mathcal{A}^{\mathsf{ZKProve}(\mathsf{CRS}_{ZK}, \cdot, \cdot)}(\mathsf{CRS}_{ZK}) = 1 | (\mathsf{CRS}_{ZK}, \cdot) \leftarrow \mathsf{ZKSetup}(1^\lambda)\right] \overset{s}{\approx} \Pr\left[\mathcal{A}^{\mathsf{ZKSimulate}(\mathsf{CRS}_{ZK}, \tau, \cdot)}(\mathsf{CRS}_{ZK}) = 1 | (\mathsf{CRS}_{ZK}, \tau) \leftarrow \mathsf{ZKSetup}(1^\lambda)\right]$$

- **Simulation Extractability:** There exists an extractor $\mathsf{Extract}$ such that

$$\Pr\left[\mathcal{R}(x, \omega) = 1 \,\middle|\, \begin{array}{c} (\mathsf{CRS}_{ZK}, \tau) \leftarrow \mathsf{ZKSetup}(1^\lambda), \\ (x, \pi) \leftarrow \mathcal{A}^{\mathsf{ZKSimulate}(\mathsf{CRS}_{ZK}, \tau, \cdot)}(\mathsf{CRS}_{ZK}), \\ \omega \leftarrow \mathsf{Extract}(\mathsf{CRS}_{ZK}, \tau, x, \pi) \end{array}\right] \geq 1 - \mathsf{negl}(\lambda)$$

It has been shown that realizing this primitive for languages outside BBP requires the common reference string model [Ore87, GO94, GK96]. We present the common reference string ideal functionality from [CF01] in Figure 5.

Functionality $\mathcal{F}_{CRS}^{\mathcal{D}}$

$\mathcal{F}_{CRS}^{\mathcal{D}}$ proceeds as follows, when parameterized by a distribution $\mathcal{D}$:

Common Reference String: When activated for the first time on input $(\mathsf{CRS}, \mathsf{sid})$, choose a value $d \xleftarrow{\$} \mathcal{D}$ and send back to the activating party. In each other activation return the value d to the activating party.

Figure 5: Ideal functionality for generating a Common Reference String, from [CF01].

4.4 Lossy Tag Encryption

Lossy encryption [BHY09] is an encryption application of lossy trapdoor functions, which were introduced by Peikert and Waters [PW08]. Intuitively, lossy encryption is a public key encryption scheme that has an algorithm to generate *lossy keys* that are computationally indistinguishable from normal keys. When encrypting with these lossy keys, the resulting ciphertext contains no information about the plaintext.

Definition 3 A lossy public-key encryption scheme is a tuple of algorithms $(\mathsf{KeyGen}, \mathsf{KeyGen}_{\mathsf{loss}}, \mathsf{Enc}, \mathsf{Dec})$ defined as

- $\mathsf{KeyGen}(1^\lambda)$ generates an injective keypair $(\mathsf{pk}, \mathsf{sk})$
- $\mathsf{KeyGen}_{\mathsf{loss}}(1^\lambda)$ generates a lossy keypair $(\mathsf{pk}, \cdot)$
- $\mathsf{Enc}(\mathsf{pk}, m)$ takes in a public key pk and a plaintext message m and outputs a ciphertext c
- $\mathsf{Dec}(\mathsf{sk}, c)$ takes in a secret key sk and a ciphertext c and either outputs $\perp$ or the message m

We require that the above algorithms satisfies the following properties:

- **Correctness on real keys**: For all messages m, it should hold that

$$\Pr\left[m = \mathsf{Dec}(\mathsf{sk}, \mathsf{Enc}(\mathsf{pk}, m)) \,\middle|\, (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda)\right] = 1$$

- **Lossiness on lossy keys**: for all $(\mathsf{pk}, \cdot) \leftarrow \mathsf{KeyGen}_{\mathsf{loss}}(1^\lambda)$ and m_0, m_1 such that $|m_0| = |m_1|$,

$$\mathsf{Enc}(\mathsf{pk}, tag, m_0) \overset{s}{\approx} \mathsf{Enc}(\mathsf{pk}, tag, m_1)$$

- **Indistinguishability of keys**: Finally, over all random coins, it should hold that

$$\mathsf{KeyGen}(1^\lambda) \overset{c}{\approx} \mathsf{KeyGen}_{\mathsf{loss}}(1^\lambda)$$

In this work, we require a specific generalization of lossy encryption we call lossy-tag encryption (LTE). Intuitively, this is an encryption scheme with a single public key in which encryption takes as input a "tag" in addition to the public key and plaintext. Encrypting under a tag from a specific subset will produce in an injective ciphertext, while the remaining tags will produce a lossy ciphertext. This notion is closely related to numerous previous works, including lossy encryption [BHY09], lossy trapdoor functions [PW08], identity-based lossy trapdoor functions [BKPW12] all-but-one functions [PW08], and all-but-n functions [HLOV11]. We define lossy-tag encryption formally as follows:

Definition 4 A lossy-tag encryption (LTE) scheme with respect to a tag space $\mathbb{T}$ consists of a tuple of algorithms $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ defined as follows:

- $\mathsf{KeyGen}(1^\lambda, \mathcal{T})$ takes in a set of tags $\mathcal{T} \subset \mathbb{T}$ of polynomial size in λ and outputs a public key mpk and a secret key msk.
- $\mathsf{Enc}(mpk, tag, m)$ encrypts the message m under the public key mpk and $tag \in \mathbb{T}$ to produce ciphertext c.
- $\mathsf{Dec}(msk, tag, c)$ takes in the secret key msk, $tag \in \mathbb{T}$ and a ciphertext c and either outputs a message m or $\perp$.

We require that the above algorithms satisfy the follow properties

- **Correctness on injective tags**:

$$\Pr\left[m = \mathsf{Dec}(msk, tag, \mathsf{Enc}(mpk, tag, m)) \;\middle|\; \begin{matrix} tag \in \mathcal{T} \\ (mpk, msk) \leftarrow \mathsf{KeyGen}(1^\lambda, \mathcal{T}) \end{matrix}\right] = 1$$

- **Lossiness on lossy tags**: for all messages m_0, m_1 and all sets $\mathcal{T}$, if $tag \notin \mathcal{T}$ and $(mpk, msk) \in \mathsf{KeyGen}(1^\lambda, \mathcal{T})$, then

$$\mathsf{Enc}(mpk, tag, m_0) \overset{s}{\approx} \mathsf{Enc}(mpk, tag, m_1)$$

- **Indistinguishability of tag sets**: for all sets $\mathcal{T}_0 \neq \mathcal{T}_1$ such that $|\mathcal{T}_0| = |\mathcal{T}_1|$,

$$\mathsf{KeyGen}(1^\lambda, \mathcal{T}_0) \overset{c}{\approx} \mathsf{KeyGen}(1^\lambda, \mathcal{T}_1)$$

A stronger version of this definition could remove the requirement that $|\mathcal{T}_0| = |\mathcal{T}_1|$ for indistinguishability of tag sets. For our constructions, we do not concern ourselves with this leakage.

Realizing lossy-tag encryption: When the size of $\mathbb{T}$ is polynomial in the security parameter, it is trivial to realize lossy-tag encryption from standard lossy encryption [BHY09] simply by generating one lossy keypair to represent each "tag". However, even for small sets $\mathbb{T}$ this may produce unreasonably large public keys. In this work, we present a direct instantiation of a lossy-tag encryption scheme based on DDH, such that the public parameters mpk that are linear in $|\mathcal{T}|$.

Let $\mathbb{G}$ be a cyclic group of order p. Define $\mathsf{pk} = (g, h, \tilde{g}, \tilde{h}) \in \mathbb{G}^4$, and $\mathsf{DoubleEncrypt}(\mathsf{pk}, m; r_1, r_2)$ to output $(g^{r_1}h^{r_2}, \tilde{g}^{r_1}\tilde{h}^{r_2} \cdot m)$. As noted in [BHY09], if pk is a DDH tuple then this encryption is injective, but if pk is a random tuple then the encryption is statistically lossy. We now present a construction Π_{LTE} for lossy-tag encryption as follows:

- $\mathsf{KeyGen}(params, \mathcal{T}) \to (mpk, msk)$. Sample $params = p, \mathcal{G}, g, h, \hat{g}, \hat{h}$ where $\mathcal{G}$ has order p and $g, h, \hat{g}, \hat{h}$ are generators of $\mathcal{G}$. Sample a random polynomial $A(x)$ in $\mathbb{Z}_p$ of degree $k = |\mathcal{T}|$ such that for each $s \in \mathcal{T}, A(x) = 0$. Then compute $B(x) = d_2 A(x)$ for some constant $d_2 \neq \frac{d_0}{d_1}$. Let α_i be the i^{th} coefficient of $A(x)$ and β_i be the i^{th} coefficient of $B(x)$. Use rejection sampling to sample random $\eta_0 \ldots \eta_k$ such then when η_i is interpreted as the i^{th} coefficient of a polynomial $E(x)$, $E(tag) \neq 0$ for all $tag \in \mathcal{T}$. Compute $mpk = ((g^{\eta_k}\hat{g}^{\alpha_k}, \ldots, g^{\eta_0}\hat{g}^{\alpha_0}), (h^{\eta_k}\hat{h}^{\beta_k}, \ldots, h^{\eta_0}\hat{h}^{\beta_0}))$. Compute $msk = (\eta_0, \eta_1, \ldots, \eta_k)$ and output mpk, msk.
- $\mathsf{Enc}(params, mpk, tag, m) \to c$.
  - Parse $((g^{\eta_k}\hat{g}^{\alpha_k}, \ldots, g^{\eta_0}\hat{g}^{\alpha_0}), (h^{\eta_k}\hat{h}^{\beta_k}, \ldots, h^{\eta_0}\hat{h}^{\beta_0})) \leftarrow mpk$
  - Compute the user public key

$$\mathsf{pk} = (g, h, \prod_{i=0}^{k}(g^{\eta_i}\hat{g}^{\alpha_i})^{tag^i}, \prod_{i=0}^{k}(h^{\eta_i}\hat{h}^{\beta_i})^{tag^i})$$

– Sample $r_1, r_2 \leftarrow \mathbb{Z}_p$
– Compute and return

$$c = (g^{r_1}h^{r_2}, \left(\prod_{i=0}^{k}(g^{\eta_i}\hat{g}^{\alpha_i})^{tag^i}\right)^{r_1} \cdot \left(\prod_{i=0}^{k}(h^{\eta_i}\hat{h}^{\beta_i})^{tag^i}\right)^{r_2} \cdot m)$$

- $\mathsf{Dec}(params, msk, tag, c) \rightarrow m$
  – Parse $(\eta_0, \eta_1, \ldots, \eta_k) \leftarrow msk$
  – Parse $(c_1, c_2) \leftarrow c$
  – Compute $y = \sum_{i=0}^{k} \eta_0 (tag)^k$
  – Compute and return $m = \frac{c_2}{c_1^y}$.

Proof. We now prove that Π_{LTE} above realizes the the functionality of lossy-tag encryption. To do so, we recall the construction of a lossy encryption scheme in Section 4.1 of [BHY09]. As mentioned above, they observe that ElGamal double encryption is injective when the public key has the structure (g, h, g^x, h^x), but is lossy when the public key has the structure (g, h, g^x, h^y), for $x \neq y$.

Correctness on injective tags. For injective tags, $A(x) = B(x) = 0$. Recall that the public key used during encryption is computed as $(g, h, \prod_{i=0}^{k}(g^{\eta_i}\hat{g}^{\alpha_i})^{tag^i}, \prod_{i=0}^{k}(h^{\eta_i}\hat{h}^{\beta_i})^{tag^i})$. Written another way, this is $(g, h, g^{E(tag)}\hat{g}^{A(tag)}, h^{E(tag)}\hat{h}^{B(tag)})$. Because $A(x) = B(x) = 0$, the public key is $(g, h, g^{E(tag)}, h^{E(tag)})$, where $E(tag)$ is non-zero. Note that this form is the same form as an injective key from [BHY09], so the resulting ciphertext is injective with corresponding private key $E(tag)$.

Lossiness on lossy tags. For lossy tags, $A(x) \neq B(x) \neq 0$. This can be observed because $B(x) = kA(x)$, and there are at most $|\mathcal{T}|$ zeros of a degree $|\mathcal{T}|$ polynomial, and all all these zeros were set to be the injective tags. The public key used for encryption is, as before, $(g, h, g^{E(tag)}\hat{g}^{A(tag)}, h^{E(tag)}\hat{h}^{B(tag)})$. Without loss of generality, the public key can then be written as

$$(g, h, g^{E(tag)+d_0A(tag)}, h^{E(tag)+d_1B(tag)}).$$

Note that because $B(\cdot)$ was sampled such that $d_2A(x) = B(x)$, and $\frac{d_0}{d_1} \neq d_2$, then $E(tag) + d_0A(tag) \neq E(tag) + d_1B(tag)$. Thus this public key is structured exactly like the lossy key from [BHY09], so the resulting ciphertext is lossy.

Indistinguishability of tag sets. Due to the key indistinguishability of [BHY09], it is clear that a lossy key and an injective key, when computed during encryption, are statistically indistinguishable. All that remains to argue is that the public parameters leak no information about the tag set besides its size (note that the size of mpk trivially leaks $|\mathcal{T}|$). Notice that it is sufficient to show that this property holds when two sets differ in only a single tag, as a straightforward hybrid argument in which a single tag is swapped in each hybrid can generalize the result. Next, notice that each element in mpk is formed like a Pendersen commitment [Ped92] to α_i or β_i. Thus, it is clear to see that if there exists an adversary that can distinguish between sets, it can be used to break the hiding property of Pendersen commitments.

4.5 Multi-sender Non-interactive Secure Computation

When instantiating our prospective protocol for arbitrary predicates in Section 5.1,we will require the use of Non-interactive Secure Computation (NISC) [IKO+11,AMPR14]. In NISC, a receiver can post an encryption embedding a secret x_1 such that senders with secret x_2 can reveal $f(x_1, x_2)$ to the receiver by sending only a single message. Realizing such a scheme (see [IKO+11]) is feasible in the CRS model [Can01, CLOS02] from two-round, UC-secure malicious oblivious transfer [DNO09, NO09], Yao's garbled circuits [Yao86, HK07], and generic non-interactive zero knowledge (see Section 4.3). The resulting protocols, however, are very

inefficient and require non-blackbox use of the underlying cryptographic primitives. While this is sufficient for our purposes, we note that depending on specific functionality required in an instantiation of ARLEAS, it may be possible to use more efficient constructions (*i.e.* depending on the size of the predicate circuit, etc.) Because the notation for NISC protocols varies, we fix it for this work below. We omit the ideal functionality of multi-sender NISC from [AMPR14], due to space constraints. Because we require non-blackbox use of the primitive, we will use it directly rather than as a hybrid.

Definition 5 (Multi-sender Non-interactive Secure Computation) *A garbling scheme for a functionality* $f : \{0,1\}^{\mathsf{input}_1} \times \{0,1\}^{\mathsf{input}_2} \to \{0,1\}^{\mathsf{output}}$ *is a tuple of PPT algorithms* $\Pi_{NISC} := (\mathsf{GenCRS}, \mathsf{NISC}_1, \mathsf{NISC}_2, \mathsf{Evaluate})$ *such that*

- $\mathsf{GenCRS}(1^\lambda, \mathsf{input}; r) \to (\mathsf{CRS}_{\mathsf{NISC}}, \tau_{\mathsf{NISC}})$*:* GenCRS *takes the security parameter* 1^n *and outputs a CRS, along with a simulation backdoor* τ_{NISC}*. When we explicitly need to specify the randomness, we will include it as* r *as here.*
- $\mathsf{NISC}_1(\mathsf{CRS}_{\mathsf{NISC}}, x_1; r) \to (\mathsf{nisc}_1^{public}, \mathsf{nisc}_1^{private})$*:* NISC_1 *takes in the CRS and an input* $x \in \{0,1\}^{\mathsf{input}_1}$ *and outputs the first message* NISC_1*. When we explicitly need to specify the randomness, we will include it as* r *as here.*
- $\mathsf{NISC}_2(\mathsf{CRS}_{\mathsf{NISC}}, f, x_2, \mathsf{nisc}_1^{public}; r) \to \mathsf{nisc}_2$*:* NISC_2 *takes in the CRS, a circuit* C*, an input* $x_2 \in \{0,1\}^{\mathsf{input}_2}$ *and the first garbled circuit message* nisc_1^{public}*. It outputs the second message* nisc_2*. When we explicitly need to specify the randomness, we will include it as* r *as here.*
- $\mathsf{Evaluate}(\mathsf{CRS}_{\mathsf{NISC}}, \mathsf{nisc}_2, \mathsf{nisc}_1^{private})$*:* $\mathsf{Evaluate}$ *takes as input the second GC message* nisc_2 *along with the private information* $\mathsf{nisc}_1^{private}$ *and outputs* $y \in \{0,1\}^{\mathsf{output}}$ *or the error symbol* $\perp$

We give an ideal world security definition for a multi-sender NISC in Figure 6.

Multi-sender Non-interactive Secure Computation.

Receiver Posting: Upon receiving a message (Input_1, x_1) from P_1, store x_1. Initialize and empty table $\mathbb{T}$ and ignore future Input_1 messages.

Sender Input: Upon receiving a message (Input_2, x_2) from P_i, store (P_i, x_2) in table $\mathbb{T}$.

Outputs: Upon receiving a message $(\mathsf{Outputs})$ from P_1, send $(\{(P_i, f(x_1, x_2)\}_{(P_i, x_2) \in \mathbb{T}})$

Figure 6: Ideal functionality for multi-sender NISC, from [AMPR14].

4.6 Witness Encryption and Extractable Witness Encryption

Our *retrospective* constructions require extractable witness encryption (EWE) [BCP14], a variant of witness encryption in which the existence of a distinguisher can be used to construct an extractor for the necessary witness [GLW14]. While EWE is a strong assumption, in later sections of this work we show that it is a minimal requirement for the existence of retrospective ARLEAS.

Definition 6 (Extractable Witness Encryption) *An extractable witness encryption* $\Pi_{EWE} = (\mathsf{Enc}, \mathsf{Dec})$ *for an NP language* L *associated with relation* R *consists of the following algorithms:*

- $\mathsf{Enc}(1^\lambda, x, m)$*: On input instance* x *and message* $m \in \{0,1\}$*, it outputs a ciphertext* c*.*

– $\mathsf{Dec}(c, w)$*: On input ciphertext* c *and witness* w*, it outputs* m'*.*

We require that the above primitive satisfy the following properties:

– **Correctness***: For every* $(x, w) \in R$*, for every* $m \in \{0,1\}$*,*

$$\Pr[m = \mathsf{Dec}(\mathsf{Enc}(1^\lambda, x, m), w)] = 1$$

– **Extractable Security***: For any PPT adversary* $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$*, if:*

$$\Pr\left[\mathcal{A}_1(1^\lambda, c, \mathsf{state}) = b \;\middle|\; \begin{array}{c} b \leftarrow \{0,1\} \\ (m_0, m_1, \mathsf{state}) \leftarrow \mathcal{A}_0(1^\lambda, x) \\ c \leftarrow \mathsf{Enc}(1^\lambda, x, m_b) \end{array}\right] \geq \frac{1}{2} + \mathsf{negl}(\lambda)$$

then there exists a PPT extractor Ext *such that for all auxiliary inputs* aux*:*

$$\Pr\left[w \leftarrow \mathsf{Ext}_{\mathcal{A}}(1^\lambda, x, aux) \text{ } s.t. \text{ } : (x, w) \in R\right] \geq \mathsf{negl}(\lambda)$$

4.7 Programmable Global Random Oracle Model

The security proof for our retrospective construction makes use of the programmable global random oracle model, introduced in [CDG+18]. The ideal functionality $\mathcal{G}_{\mathsf{pRO}}$ is illustrated in Figure 7.

Functionality $\mathcal{G}_{\mathsf{pRO}}$.

Initiate with an empty list $\mathsf{List}_{\mathcal{H}}$

Query: Upon receiving message $(\mathsf{HashQuery}, m)$ from party P, the ideal functionality proceeds as follows. Find h such that $(m, h) \in \mathsf{List}_{\mathcal{H}}$. If no such h exists, let $h \leftarrow\!\$\, \{0,1\}^\ell$ and store (m, h) in $\mathsf{List}_{\mathcal{H}}$. Return $(\mathsf{HashConfirm}, h)$ to P.

Program: Upon receiving message $(\mathsf{ProgramRO}, m, h)$ from adversary $\mathcal{A}$, the ideal functionality proceeds as follows. If $\exists h' \in \{0,1\}^\ell$ such that $(m, h') \in \mathsf{List}_{\mathcal{H}}$ and $h \neq h'$, then abort. Otherwise, add (m, h) to $\mathsf{List}_{\mathcal{H}}$ and output $(\mathsf{ProgramConfirm})$ to $\mathcal{A}$

Figure 7: Ideal functionality for the global programmable random oracle, from [CDG+18].

5 Prospective Solution

In this section we describe a prospective ARLEAS scheme, the first of which work with warrants that specify the target's identity and the second of which supports arbitrary predicates. Recall that the key feature of the prospective case is that warrants must be activated *before* targets perform encryption. A key implication of this setting is that new cryptographic material can be generated and distributed to users each time law enforcement updates the set of active warrants. The technical challenge, therefore, is to ensure that this material is distributed in such a way that the surveillance it permits is *accountable*, without revealing to targets any confidential information about which messages are being accessed.

The need for accountability restricts us from using many natural cryptographic tools. For example, Identity Based Encryption (IBE) systems provide a natural form of key escrow. Unfortunately, in a standard IBE scheme this key escrow is absolute: the master authority can decrypt any ciphertext in the system. To

enable limited surveillance, we require a system in which only a subset of communications will be targeted at any time epoch, and no additional information about *non-targeted* plaintexts will be revealed to the authorities. The first scheme relies on the Π_{LTE} scheme presented in Section 4 and ensures that messages sent to recipients under surveillance contain a copy of the ciphertext that can be decrypted by law enforcement, while messages send to recipients not under surveillance contain only a lossy ciphertext.

For generality, our main second construction supports targeting by allowing warrants to specify an arbitrary predicate over the *metadata* of a transmitted messages. In practice, we realize this functionality through the use of public ledgers and non-interactive secure computation techniques.

5.1 UC-Realizing $\mathcal{F}_{\text{ARLEAS}}^{v,t,p,\theta,\textsf{pro}}$ for Identity-Based Predicates

Our first construction only permits warrants that specify the identity of individuals to surveil and leverages the Π_{LTE} scheme presented in Section 4. With this encryption scheme, encrypting to some public keys create information theoretic lossy ciphertexts, while using other public keys results in injective ciphertexts. We use this scheme to allow law enforcement to decrypt messages exactly when they have activated a warrant corresponding to the recipient's identity. When sending a new message, a user encrypts directly to the recipient as normal and creates a second ciphertext using the Π_{LTE} scheme. The key material used for the second ciphertext comes public parameters generated by law enforcement that are posted onto the public ledger.

Our construction makes use of a CCA secure encryption system Π_{Enc}, a SUF-CMA secure signature scheme Π_{Sign}, a lossy-tag encryption scheme Π_{LTE} (presented in Section 4). We now present a protocol $\pi_{\text{PRO}}^{v,t,p,\theta}$ in the $\mathcal{L}^{\textsf{Verify}}$, $\mathcal{F}_{CRS}^{\Pi_{\text{NIZK}}.\textsf{ZKSetup}}$, $\mathcal{F}_{AUTH}$ hybrid model. Our scheme consists of the following interactive protocols:

$\pi_{\textbf{PRO}}^{v,t,p,\theta}.\textsf{Setup}$**:**

- All users send ($\textsf{CRS}$) to $\mathcal{F}_{CRS}^{\Pi_{\text{NIZK}}.\textsf{ZKSetup}}$ to retrieve the common reference string for the NIZK scheme.
- Each user P_j computes $(\textsf{pk}_j, \textsf{sk}_j) \leftarrow \Pi_{\text{Enc}}.\textsf{KeyGen}(1^\lambda)$ and selects a unique tag tag_j and sends $(\textsf{pk}_j, tag_j)$ to P_{LE} and to each other user P_i via $\mathcal{F}_{AUTH}$.
- The judge P_J computes $(\textsf{pk}_{\text{sign}}, \textsf{sk}_{\text{sign}}) \leftarrow \Pi_{\text{Sign}}.\textsf{KeyGen}(1^\lambda)$ and send $\textsf{pk}_{\text{sign}}$ to all other users via $\mathcal{F}_{AUTH}$.
- Law enforcement P_{LE} runs $\pi_{\text{PRO}}^{v,t,p,\theta}.\textsf{ActivateWarrant}$ with as input an empty set $\emptyset$ as the valid warrants.

$\pi_{\textbf{PRO}}^{v,t,p,\theta}.\textsf{SendMessage}$:

- The sender P_i computes the ciphertext $(c_1, c_2, \pi, \textsf{meta})$ as follows, and sends it to P_j and P_{LE} via $\mathcal{F}_{AUTH}$:
  - Send ($\textsf{GetCounter}$) to $\mathcal{L}^{\textsf{Verify}}$ and receive the current counter ℓ. Then query $\mathcal{L}^{\textsf{Verify}}$ on $(\textsf{GetVal}, \ell)$ to receive the latest posting $(\ell, x, \pi_{\textsf{publish}})$. Parse x as $(mpk, \pi, \textsf{info})$. If $\Pi_{\text{NIZK}}.\textsf{ZKVerify}(mpk, \textsf{info}, \pi) = 0$ or $\mathcal{L}^{\textsf{Verify}}.\textsf{Verify}(\ell \| (mpk, \pi, \textsf{info}), \pi_{\textsf{publish}}) = 0$ return $\perp$ and halt.
  - $c_1 \leftarrow \Pi_{\text{Enc}}.\textsf{Enc}(\textsf{pk}_j, m; r_1)$, where $r_1 \xleftarrow{\$} \{0,1\}^\lambda$
  - $c_2 \leftarrow \Pi_{\text{LTE}}.\textsf{Enc}(mpk, tag_j, m; r_2)$ where $r_2 \xleftarrow{\$} \{0,1\}^\lambda$
  - Use $\Pi_{\text{NIZK}}.\textsf{ZKProve}$ to compute π such that

$$\pi \leftarrow NIZK\left\{(m, r_1, r_2) : \begin{array}{c} c_1 = \Pi_{\text{Enc}}.\textsf{Enc}(\textsf{pk}_j, m; r_1) \wedge \\ c_2 = \Pi_{\text{LTE}}.\textsf{Enc}(mpk, tag_j, m; r_2) \end{array}\right\}$$

  - Create $\textsf{meta} \leftarrow tag_j$

- Upon receiving c from P_i, P_j calls $\pi_{\text{PRO}}^{v,t,p,\theta}.\mathsf{VerifyMessage}$ on c. If the output is 1, then recover the message as $m \leftarrow \Pi_{\text{Enc}}.\mathsf{Dec}(\mathsf{sk}_j, c_2)$
- Upon receiving c from P_i, P_{LE} calls $\pi_{\text{PRO}}^{v,t,p,\theta}.\mathsf{VerifyMessage}$ on c. If the output is 1, then recover the message as $m \leftarrow \Pi_{\text{LTE}}.\mathsf{Dec}(msk, tag_j)$

$\pi_{\mathbf{PRO}}^{v,t,p,\theta}.\mathsf{VerifyMessage}$:

- Any party parses $(c_1, c_2, \pi, \mathsf{meta}) \leftarrow c$ and verifies that π is correct and computes $v(\mathsf{meta}, aux)$, aborting if the output is 0. Otherwise, output 1.

$\pi_{\mathbf{PRO}}^{v,t,p,\theta}.\mathsf{RequestWarrant}$**:**

- P_{LE} sends $(\mathsf{RequestWarrant}, tag)$ to P_J via $\mathcal{F}_{AUTH}$. P_J then either decides to send $(\mathsf{Disapprove})$ to P_{LE} and halt or executes the following:
  - Verify that $p(tag) = 1$. If not send $(\mathsf{Disapprove})$ to P_{LE} and abort.
  - $\sigma \leftarrow \Pi_{\text{Sign}}.\mathsf{Sign}(\mathsf{sk}_{\text{sign}}, tag)$
  - Send the signed warrant (tag, σ) to P_{LE} via $\mathcal{F}_{AUTH}$.

$\pi_{\mathbf{PRO}}^{v,t,p,\theta}.\mathsf{ActivateWarrant}$**:**

- P_{LE} adds the new warrant w to the set of valid warrants $\mathcal{W}$. It then extracts the set of tags $\mathcal{T} \leftarrow \{tag | (tag, \sigma) \in \mathcal{W}\}$
- Compute $(mpk, msk) \leftarrow \Pi_{\text{LTE}}.\mathsf{KeyGen}(1^\lambda, \mathcal{T}; r)$ for $r \xleftarrow{\$} \{0,1\}^\lambda$
- Compute $\mathsf{info} \leftarrow \{t(tag, \sigma) | (tag, \sigma) \in \mathcal{W}\}$
- Use $\Pi_{\text{NIZK}}.\mathsf{ZKProve}$ to compute π such that

$$\pi \leftarrow NIZK\{(\mathcal{W}, \mathcal{T}, msk, r) : \mathsf{info} = \{t(tag, \sigma) | (tag, \sigma) \in \mathcal{W}\} \wedge \mathcal{T} \leftarrow \{tag | (tag, \sigma) \in \mathcal{W}\} \wedge (mpk, msk) \in \Pi_{\text{LTE}}.\mathsf{KeyGen}(1^\lambda, \mathcal{T}; r) \wedge \forall (tag, \sigma) \in \mathcal{W}, \Pi_{\text{Sign}}.\mathsf{Verify}(\mathsf{pk}_{\text{sign}}, tag, \sigma) = p(tag) = 1\}$$

- Send $(\mathsf{Post}, (mpk, \pi, \mathsf{info}))$ to $\mathcal{L}^{\mathsf{Verify}}$ and receive $(\ell, x, \pi_{\mathsf{publish}})$.

Theorem 1 *Assuming a CCA secure public key encryption scheme Π_{Enc}, a SUF-CMA secure signature scheme Π_{Sign}, a NIZK scheme Π_{NIZK}, and a lossy-tag encryption scheme Π_{LTE}, $\pi_{PRO}^{v,t,p,\theta}$ UC-realizes $\mathcal{F}_{ARLEAS}^{v,t,p,\theta,pro}$ initialized in prospective mode in the $\mathcal{L}^{\mathsf{Verify}}$, $\mathcal{F}_{CRS}^{\Pi_{NIZK}.\mathsf{ZKSetup}}$, $\mathcal{F}_{AUTH}-$hybrid model for* meta *that contains receiver identity $\theta(w, \mathsf{meta}) = (w == \mathsf{meta})$.*

Security Proof. The proof of security can be found in Appendix A

5.2 UC-Realizing $\mathcal{F}_{\mathbf{ARLEAS}}^{v,t,p,\theta,\mathbf{pro}}$ for Arbitrary Predicates

The prospective solution we have presented is limited in the flexibility of warrants; unlike the ideal functionality, warrants were limited to specifying a single individual. It is clear that it would be better to support *arbitrary* warrants whose applicability to some meta could be checked with the warrant scope functionality θ.

First, we note that directly extending the existing solution's intuitions is insufficient, as creating a one-to-one correspondence between tags and meta would clearly result in exponentially sized public parameters.

We require a mechanism that evaluates the warrant scope predicate $\theta(w, \mathsf{meta})$ and outputs the message only if the result is 1. Clearly this can be accomplished using extractable witness encryption, and indeed we will use extractable witness encryption to accomplish a similar goal in Section 6. However, we are able to leverage the *prospective* nature of this case to realize prospective ARLEAS from non-interactive secure computation (NISC) [IKO+11]. Recall that a NISC scheme for some function f allows a receiver to post an encryption of some secret x_1 such that all players can reveal $f(x_1, x_2)$ to the receiver with only one message, without revealing anything about x_2 beyond the output of the function. For the following construction, we require an NISC scheme for the function I_k, defined as

$$I_k((w_1, w_2, \ldots, w_k), (m, \mathsf{meta})) = m \wedge (\theta(\mathsf{meta}, w_1) \vee \ldots \vee \theta(\mathsf{meta}, w_k)).$$

This function evaluates the warrant scope check functionality on the metadata over k different warrants. If any of them evaluate to true, the message is output. Otherwise, I_k outputs 0. Note that the number of warrants is an explicit parameter of the function and its circuit representation.

Law enforcement begins by posting the first message of the NISC scheme, embedding as input their k warrants, along with the transparency information and proof of correctness. As before, senders send a ciphertext (c_1, c_2, π). c_1 remains a normal public key ciphertext for the recipient. c_2 is modified (from the previous construction) to be the second message of the NISC scheme. This message must embed the inputs (m, meta). Most known realizations of NISC rely on garbled circuits, with the second message containing the garbling of the intended function and hardcoding the sender's inputs. As such, we need to ensure that the sender computes the second message of the NISC scheme with respect to the correct functionality; this can be handled by requiring malicious security from the underlying NISC scheme or by including a correctness in the NIZK π. As we require a proof of consistency (*i.e. c_1 and c_2 embed the same message*), we already require non-blackbox use of the NISC scheme.

Upon receiving the resulting ciphertext, law enforcement can attempt to decrypt by evaluating the NISC ciphertext. By the security of the NISC scheme, law enforcement will only learn information about the plaintext if they have a relevant warrant and posted the required transparency information, accomplishing our goal.

We now proceed to give a formal description of this protocol.

$\pi^{v,t,p,\theta}_{\mathbf{PRO}}.\mathsf{Setup}$**:**

- All users send (CRS) to $\mathcal{F}_{CRS}^{\Pi_{\mathrm{NIZK}}.\mathsf{ZKSetup}}$ to retrieve the common reference string for the NIZK scheme and all users send (CRS) to $\mathcal{F}_{CRS}^{\Pi_{\mathrm{NISC}}.\mathsf{GenCRS}}$ to retrieve the common reference string for the NISC scheme $\mathsf{CRS}_{\mathsf{NISC}}$.
- Each user P_j computes $(\mathsf{pk}_j, \mathsf{sk}_j) \leftarrow \Pi_{\mathrm{Enc}}.\mathsf{KeyGen}(1^\lambda)$ and sends pk_j to P_{LE} and to each P_i via $\mathcal{F}_{AUTH}$.
- The judge P_J computes $(\mathsf{pk}_{\mathrm{sign}}, \mathsf{sk}_{\mathrm{sign}}) \leftarrow \Pi_{\mathrm{Sign}}.\mathsf{KeyGen}(1^\lambda)$ and send $\mathsf{pk}_{\mathrm{sign}}$ to all other users via $\mathcal{F}_{AUTH}$.
- Law enforcement P_{LE} runs $\pi^{v,t,p,\theta}_{\mathrm{PRO}}.\mathsf{ActivateWarrant}$ with an empty set $\emptyset$ as the valid warrants.

$\pi^{v,t,p,\theta}_{\mathbf{PRO}}.\mathsf{SendMessage}$:

- The sender P_i computes the ciphertext $(c_1, c_2, \pi, \mathsf{meta})$ as follows, and sends it to P_j and P_{LE} via $\mathcal{F}_{AUTH}$:
  - Send $(\mathsf{GetCounter})$ to $\mathcal{L}^{\mathsf{Verify}}$ and receive the current counter ℓ. Then query $\mathcal{L}^{\mathsf{Verify}}$ on (GetVal, ℓ) to receive the latest posting $(\ell, x, \pi_{\mathsf{publish}})$. Parse x as $(\mathsf{nisc}_1^{\mathrm{public}}, \pi, \mathsf{info})$. If $\Pi_{\mathrm{NIZK}}.\mathsf{ZKVerify}(\mathsf{nisc}_1^{\mathrm{public}}, \mathsf{info}, \pi) = 0$ or $\mathcal{L}^{\mathsf{Verify}}.\mathsf{Verify}(\ell \| (\mathsf{nisc}_1^{\mathrm{public}}, \pi, \mathsf{info}), \pi_{\mathsf{publish}}) = 0$ return $\perp$ and halt.
  - $c_1 \leftarrow \Pi_{\mathrm{Enc}}.\mathsf{Enc}(\mathsf{pk}_j, m; r_1)$, where $r_1 \xleftarrow{\$} \{0,1\}^\lambda$
  - Create meta

– $\mathsf{nisc}_2 \leftarrow \Pi_{\text{NISC}}.\mathsf{NISC}_2(\mathsf{CRS}_{\mathsf{NISC}}, I_{|\mathsf{info}|}, (m, \mathsf{meta}), \mathsf{nisc}_1^{\text{public}}; r_2)$, where $r_2 \xleftarrow{\$} \{0,1\}^\lambda$

– $c_2 \leftarrow \mathsf{nisc}_2$

– Use $\Pi_{\text{NIZK}}.\mathsf{ZKProve}$ to compute π such that

$$\pi \leftarrow NIZK\left\{(m, r_1, r_2) : \begin{array}{c} c_1 = \Pi_{\text{Enc}}.\mathsf{Enc}(\mathsf{pk}_j, m; r_1) \wedge \\ c_2 = \Pi_{\text{NISC}}.\mathsf{NISC}_2(\mathsf{CRS}_{\mathsf{NISC}}, I_{|\mathsf{info}|}, (m, \mathsf{meta}), \mathsf{nisc}_1^{\text{public}}; r_2) \end{array}\right\}$$

- Upon receiving c from P_i, P_j calls $\pi_{\text{PRO}}^{v,t,p,\theta}.\mathsf{VerifyMessage}$ on c. If the output is 1, then recover the message as $m \leftarrow \Pi_{\text{Enc}}.\mathsf{Dec}(\mathsf{sk}_j, c_2)$
- Upon receiving c from P_i, P_{LE} calls $\pi_{\text{PRO}}^{v,t,p,\theta}.\mathsf{VerifyMessage}$ on c. If the output is 1, then recover the message as $m \leftarrow \Pi_{\text{NISC}}.\mathsf{Evaluate}(\mathsf{CRS}_{\mathsf{NISC}}, \mathsf{nisc}_2, \mathsf{nisc}_1^{\text{private}})$

$\pi_{\mathbf{PRO}}^{v,t,p,\theta}.\mathsf{VerifyMessage}$:

- Any party parses $(c_1, c_2, \pi, \mathsf{meta}) \leftarrow c$ and verifies that π is correct and computes $v(\mathsf{meta}, aux)$, aborting if the output is 0. Otherwise, output 1.

$\pi_{\mathbf{PRO}}^{v,t,p,\theta}.\mathsf{RequestWarrant}$**:**

- P_{LE} sends $(\mathsf{RequestWarrant}, \hat{w})$ to $P_J$via $\mathcal{F}_{AUTH}$. P_J then either decides to send $(\mathsf{Disapprove})$ to P_{LE} and halt or executes the following:
  – Verify that $p(\hat{w}) = 1$. If not send $(\mathsf{Disapprove})$ to P_{LE} and abort.
  – $\sigma \leftarrow \Pi_{\text{Sign}}.\mathsf{Sign}(\mathsf{sk}_{\text{sign}}, \hat{w})$
  – Send the signed warrant $w = (\hat{w}, \sigma)$ to P_{LE} via $\mathcal{F}_{AUTH}$.

$\pi_{\mathbf{PRO}}^{v,t,p,\theta}.\mathsf{ActivateWarrant}$**:**

- P_{LE} adds the new warrant w to the set of valid warrants $\mathcal{W}$. Let $w^* = w_1 \| \ldots \| w_{|\mathcal{W}|}$ for $w_i = (\hat{w}_i, \sigma_i) \in \mathcal{W}$.
- $(\mathsf{nisc}_1^{\text{public}}, \mathsf{nisc}_1^{\text{private}}) \leftarrow \Pi_{\text{NISC}}.\mathsf{NISC}_1(\mathsf{CRS}_{\mathsf{NISC}}, w^*; r)$ and record $\mathsf{nisc}_1^{\text{private}}$
- Compute $\mathsf{info} \leftarrow \{t(w) | w \in \mathcal{W}\}$
- Use $\Pi_{\text{NIZK}}.\mathsf{ZKProve}$ to compute π such that

$$\pi \leftarrow NIZK\left\{(\mathcal{W}, \mathsf{nisc}_1^{\text{private}}, r) : \begin{array}{c} \mathsf{info} = \{t(w) | w \in \mathcal{W}\} \wedge \\ (\mathsf{nisc}_1^{\text{public}}, \mathsf{nisc}_1^{\text{private}}) \leftarrow \Pi_{\text{NISC}}.\mathsf{NISC}_1(\mathsf{CRS}_{\mathsf{NISC}}, w^*; r) \wedge \\ \forall (\hat{w}, \sigma) \in \mathcal{W}, \Pi_{\text{Sign}}.\mathsf{Verify}(\mathsf{pk}_{\text{sign}}, \hat{w}, \sigma) = p(\hat{w}) = 1 \end{array}\right\}$$

- Send $(\mathsf{Post}, (\mathsf{nisc}_1^{\text{public}}, \pi, \mathsf{info}))$ to $\mathcal{L}^{\mathsf{Verify}}$ and receive $(\ell, x, \pi_{\mathsf{publish}})$.

Theorem 2 *Assuming a CCA secure public key encryption scheme Π_{Enc}, a SUF-CMA secure signature scheme Π_{Sign}, a NIZK scheme Π_{NIZK}, and an NISC scheme Π_{NISC}, $\pi_{PRO}^{v,t,p,\theta}$ UC-realizes $\mathcal{F}_{ARLEAS}^{v,t,p,\theta,pro}$ initialized in prospective mode in the $\mathcal{L}^{\mathsf{Verify}}$, $\mathcal{F}_{CRS}^{\Pi_{NIZK}.\mathsf{ZKSetup}}$, $\mathcal{F}_{CRS}^{\Pi_{NISC}.\mathsf{GenCRS}}$, $\mathcal{F}_{AUTH}-$hybrid model.*

Security Proof. The proof of security can be found in Appendix B

6 Retrospective Solution

In the previous section we proposed a protocol to realize ARLEAS under the restriction that access would be prospective only. That protocol requires that law enforcement must activate a warrant and post the resulting parameters on the ledger before any targeted communication occurs. In this section we address the retrospective case. The key difference in this protocol is that law enforcement may activate a warrant at any stage of the protocol, even after a target communication has occurred.

In this setting we assume law enforcement has a way of getting messages that were sent in the past. As described before, we take the simplifying assumption that messages automatically get send to law enforcement. In practice, either a service provider can forward them, after checking the warrant. One can try to avoid surveillance by using expiring messages, but service providers can be forced to keep encrypted messages for a certain period of time. Or law enforcement can actively record messages in transit.

Our construction makes use of an extractable witness encryption scheme Π_{EWE} (see Definition 6) to encrypt the law enforcement ciphertext c_2. This scheme is parameterized by a language L_{EWE} that is defined with respect to the transparency function $t(\cdot)$, the policy function $p(\cdot)$, the targeting function $\theta(\cdot,\cdot)$, the warrant signing key $\mathsf{pk}_{\text{sign}}$, and the ledger verification function $\mathcal{L}.\mathsf{Verify}$, as follows:

$$L_{\text{EWE}} = \left\{ \mathsf{meta} \;\middle|\; \exists w, (t, \mathsf{info}, \pi_{\mathsf{publish}}) \text{ s.t. } \begin{array}{l} w = (\hat{w}, \sigma), \mathcal{L}.\mathsf{Verify}((\ell \| \mathsf{info}), \pi_{\mathsf{publish}}) = 1, \\ \mathsf{info} = t(w), \Pi_{\text{Sign}}.\mathsf{Verify}(\mathsf{pk}_{\text{sign}}, \hat{w}, \sigma) = 1, \\ p(\hat{w}) = 1, \theta(\hat{w}, \mathsf{meta}) = 1 \end{array} \right\}$$

Intuitively, these ciphertexts can only be decrypted by law enforcement once they have performed all the accountability tasks required by the ARLEAS.
Our construction also makes use of a simulation-extractable NIZK scheme Π_{NIZK} satisfying Definition 2. We will prove statements in the following languages:

$$L^1_{\text{NIZK}} = \left\{ (c_1, c_2, \mathsf{pk}, \mathsf{meta}) \;\middle|\; \exists (r, r_1, r_2) \text{ s.t. } \begin{array}{l} c_1 = \Pi_{\text{Enc}}.\mathsf{Enc}(\mathsf{pk}, r; r_1) \wedge \\ c_2 = \Pi_{\text{EWE}}.\mathsf{Enc}(\mathsf{meta}, r; r_2) \end{array} \right\}$$

$$L^2_{\text{NIZK}} = \left\{ (\mathsf{info}, \mathsf{pk}_{\text{sign}}) \;\middle|\; \exists (\hat{w}, \sigma) \text{ s.t. } \begin{array}{l} \Pi_{\text{Sign}}.\mathsf{Verify}(\mathsf{pk}_{\text{sign}}, \hat{w}, \sigma) = 1 \wedge \\ x \leftarrow t(w) \end{array} \right\}$$

We will describe our protocol in a hybrid model that makes use of several functionalities. These include $\mathcal{L}$, $\mathcal{F}^{\mathcal{D}}_{CRS}$, $\mathcal{G}_{\mathsf{pRO}}$ and $\mathcal{F}_{AUTH}$.

6.1 UC-Realizing $\mathcal{F}^{v,t,p,\theta,\mathbf{ret}}_{\mathbf{ARLEAS}}$

We now provide a description of the retrospective ARLEAS protocol $\pi^{v,t,p,\theta}_{\text{RET}}$.

$\pi^{v,t,p,\theta}_{\mathbf{RET}}.\mathsf{Setup}$**:**

- All users send (CRS) to $\mathcal{F}^{\Pi_{\text{NIZK}}.\mathsf{ZKSetup}}_{CRS}$ to retrieve the common reference string for the NIZK scheme.
- P_J computes $(\mathsf{pk}_{\text{sign}}, \mathsf{sk}_{\text{sign}}) \leftarrow \Pi_{\text{Sign}}.\mathsf{KeyGen}(1^\lambda)$ and sends $\mathsf{pk}_{\text{sign}}$ to all other users via $\mathcal{F}_{AUTH}$.

$\pi^{v,t,p,\theta}_{\mathbf{RET}}.\mathsf{SendMessage}$**:**

- The sender P_i computes the ciphertext $(c_1, c_2, c_3, \pi, \mathsf{meta})$ as follows, and sends it to P_j and P_{LE} via $\mathcal{F}_{AUTH}$:
  - Sample $r \leftarrow \{0,1\}^\lambda$
  - Query the random oracle to obtain the hashes:
    $(\mathsf{HashConfirm}, r_1) \leftarrow \mathcal{G}_{\mathsf{pRO}}(\mathsf{HashQuery}, (\text{“ENC”} \| r \| m))$,
    $(\mathsf{HashConfirm}, r_2) \leftarrow \mathcal{G}_{\mathsf{pRO}}(\mathsf{HashQuery}, (\text{“WE”} \| r \| m))$, and

$(\mathsf{HashConfirm}, r_3) \leftarrow \mathcal{G}_{\mathsf{pRO}}(\mathsf{HashQuery}, (\text{“RP”} \| r))$

- $c_1 \leftarrow \Pi_{\text{Enc}}.\mathsf{Enc}(\mathsf{pk}, r; r_1)$, $c_2 \leftarrow \Pi_{\text{EWE}}.\mathsf{Enc}(\mathsf{meta}, r; r_2)$, and $c_3 \leftarrow m \oplus r_3$
- Use $\Pi_{\text{NIZK}}.\mathsf{ZKProve}$ to compute

$$\pi \leftarrow \mathit{NIZK}\{(r, r_1, r_2) : c_1 = \Pi_{\text{Enc}}.\mathsf{Enc}(\mathsf{pk}_j, r; r_1) \wedge c_2 = \Pi_{\text{EWE}}.\mathsf{Enc}(\mathsf{meta}, r; r_2)\}$$

- Upon receiving (send, c), P_j performs the following steps:
  - Call $\pi_{\text{RET}}^{v,t,p,\theta}.\mathsf{VerifyMessage}$ on c, aborting if the output is 0;
  - Compute $r' \leftarrow \Pi_{\text{Enc}}.\mathsf{Dec}(\mathsf{sk}_j, c_1)$
  - $(\mathsf{HashConfirm}, r_3) \leftarrow \mathcal{G}_{\mathsf{pRO}}(\mathsf{HashQuery}, (\text{“RP”} \| r'))$
  - Compute $m' \leftarrow c_3 \oplus r_3$
  - $(\mathsf{HashConfirm}, r_1) \leftarrow \mathcal{G}_{\mathsf{pRO}}(\mathsf{HashQuery}, (\text{“ENC”} \| r' \| m'))$
  - $(\mathsf{HashConfirm}, r_2) \leftarrow \mathcal{G}_{\mathsf{pRO}}(\mathsf{HashQuery}, (\text{“WE”} \| r' \| m'))$
  - Then to verify that the message has not been mauled, P_j recomputes $c_1' \leftarrow \Pi_{\text{Enc}}.\mathsf{Enc}(\mathsf{pk}_j, r'; r_1)$ and $c_2' \leftarrow \Pi_{\text{EWE}}.\mathsf{Enc}(\mathsf{meta}, r'; r_2)$. If $c_1 \neq c_1'$ or $c_2 \neq c_2'$, return $\perp$. Otherwise, return m'.
- Upon receiving (send, c), P_{LE} calls $\pi_{\text{RET}}^{v,t,p,\theta}.\mathsf{VerifyMessage}$ on c, aborting if the output is 0, and then calls $\pi_{\text{RET}}^{v,t,p,\theta}.\mathsf{AccessMessage}$ on c.

$\pi_{\mathbf{RET}}^{v,t,p,\theta}.\mathsf{VerifyMessage}$:

- Any party parses $(c_1, c_2, c_3, \pi, \mathsf{meta}) \leftarrow c$ and verifies that π is correct and computes $v(\mathsf{meta}, \mathit{aux})$, aborting if the output is 0. Otherwise, output 1.

$\pi_{\mathbf{RET}}^{v,t,p,\theta}.\mathsf{RequestWarrant}$**:**

- P_{LE} sends $(\mathsf{RequestWarrant}, \hat{w})$ to P_J via $\mathcal{F}_{AUTH}$. P_J then either decides to send $(\mathsf{Disapprove})$ to P_{LE} and halt or executes the following:
  - Verify that $p(\hat{w}) = 1$. If not send $(\mathsf{Disapprove})$ to P_{LE} and abort.
  - $\sigma \leftarrow \Pi_{\text{Sign}}.\mathsf{Sign}(\mathit{wsk}, \hat{w})$
  - Send the signed warrant $w = (\hat{w}, \sigma)$ to P_{LE} via $\mathcal{F}_{AUTH}$.

$\pi_{\mathbf{RET}}^{v,t,p,\theta}.\mathsf{ActivateWarrant}$**:**

- P_{LE} computes $\mathsf{info} \leftarrow t(w)$; uses $\Pi_{\text{NIZK}}.\mathsf{ZKProve}$ to compute

$$\pi \leftarrow \mathit{NIZK}\{(w) : w = (\hat{w}, \sigma), \Pi_{\text{Sign}}.\mathsf{Verify}(\mathsf{pk}_{\text{sign}}, \hat{w}, \sigma) = 1 \wedge \mathsf{info} \leftarrow t(w)\};$$

  and sends $(\mathsf{Post}, (\mathsf{info}, \pi))$ to $\mathcal{L}^{\mathsf{Verify}}$. It receives and returns $(\ell, \mathsf{info}, \pi_{\mathsf{publish}})$.

$\pi_{\mathbf{RET}}^{v,t,p,\theta}.\mathsf{VerifyWarrantStatus}$**:**

- P_{LE} calls $\Pi_{\text{EWE}}.\mathsf{Dec}(c_2, \mathsf{meta}, (\hat{w}, \sigma), (\ell, \mathsf{info}, \pi_{\mathsf{publish}}))$. If the output is $\perp$, return 0. Otherwise, return 1.

$\pi_{\mathbf{RET}}^{v,t,p,\theta}.\mathsf{AccessMessage}$**:**

- P_{LE} computes $r' \leftarrow \Pi_{\text{EWE}}.\mathsf{Dec}(c_2, \mathsf{meta}, (\hat{w}, \sigma), (\ell, \mathsf{info}, \pi_{\mathsf{publish}}))$.

- $(\mathsf{HashConfirm}, r_3) \leftarrow \mathcal{G}_{\mathsf{pRO}}(\mathsf{HashQuery}, (\text{“RP”} \| r'))$
- Recovers $m' \leftarrow c_3 \oplus r_3$.
- $(\mathsf{HashConfirm}, r_1) \leftarrow \mathcal{G}_{\mathsf{pRO}}(\mathsf{HashQuery}, (\text{“ENC”} \| r' \| m'))$
- $(\mathsf{HashConfirm}, r_2) \leftarrow \mathcal{G}_{\mathsf{pRO}}(\mathsf{HashQuery}, (\text{“WE”} \| r' \| m'))$
- Recomputes $c_1' \leftarrow \Pi_{\text{Enc}}.\mathsf{Enc}(\mathsf{pk}_j, r'; r_1)$ and $c_2' \leftarrow \Pi_{\text{EWE}}.\mathsf{Enc}(\mathsf{meta}, r'; r_2)$. If $c_1' = c_1$ and $c_2' = c_2$, P_{LE} returns m' and $\perp$ otherwise.

Theorem 3 *Assuming a CCA-secure public key encryption scheme Π_{Enc}, an extractable witness encryption scheme for L_{EWE}, a SUF-CMA secure signature scheme Π_{Sign}, and a simulation-extractable NIZK scheme Π_{NIZK}, $\pi_{RET}^{v,t,p,\theta}$ UC-realizes $\mathcal{F}_{ARLEAS}^{v,t,p,\theta,\mathsf{ret}}$ in the $\mathcal{L}^{\mathsf{Verify}}, \mathcal{F}_{CRS}^{\Pi_{NIZK}.\mathsf{ZKSetup}}, \mathcal{G}_{\mathsf{pRO}}-$hybrid model.*

Security Proof. The proof of security can be found in Appendix C

7 On the Need for Extractable Witness Encryption

The retrospective solution we present in Section 6 relies on extractable witness encryption. Intuitively, this strong assumption is required in our construction because a user must encrypt in a way that decryption is only possible under certain circumstances. Because the description of these circumstances can be phrased as an NP relation, witness encryption represents a “natural” primitive for realizing it. However, thus far we have not shown that the use of extractable witness encryption is *strictly* necessary. Given the strength (and implausibility [GGHW14]) of the primitive, it is important to justify its use. We do this by showing that any protocol Π_{A} that UC-realizes $\mathcal{F}_{\text{ARLEAS}}^{v,t,p,\theta,\mathsf{ret}}$ implies the existence of extractable witness encryption for a related language. Notice that this does not mean the existence of a particular ARLEAS instantiation implies the existence of generic extractable witness encryption scheme, but rather a specific, non-trivial scheme.

Before proceeding to formally define this related language, we give some intuition about its form. We wish to argue that a protocol Π_{A} *acts like* an extractable witness encryption scheme in the specific case where an adversary has corrupted the escrow authorities P_{LE} and P_J (along with an arbitrary number of unrelated users). Recall that in order to learn any information about a message sent in Π_{A}, the following conditions must be met: specifically, law enforcement must correctly run the protocol for $\Pi_{\text{A}}.\mathsf{RequestWarrant}$ and $\Pi_{\text{A}}.\mathsf{ActivateWarrant}$ such that if $\Pi_{\text{A}}.\mathsf{VerifyWarrantStatus}$ were to be called, it would output 1.[8] For the protocol we presented in Section 6, this corresponds to obtaining a correct proof of publication from the ledger. Importantly, it must be impossible for law enforcement and judges to generate this information independently; if it were possible, it would be easy for these parties to circumvent the accountability mechanism.

We give a formal definition of this language L below. We denote the view of a user P_i as $\mathcal{V}_{P_i}$, where this view is a collection of the views of running all algorithms that appear. We abuse notation slightly and denote the protocol transcript resulting from a sender P_S sending a message m to P_R as $\Pi_{\text{A}}.\mathsf{SendMessage}(\cdot, P_S, P_R, m)$

$$L = \left\{ (\mathsf{meta}, \mathsf{sid}) \;\middle|\; \exists \left(w, c, \left\{ \begin{matrix} \mathcal{V}_{P_{\text{LE}}}, \mathcal{V}_{P_J}, \\ \mathcal{V}_{P_0}, \ldots, \mathcal{V}_{P_n} \end{matrix} \right\} \right) \text{ s.t. } \begin{matrix} c, \mathsf{meta} \leftarrow \Pi_{\text{A}}.\mathsf{SendMessage}(\mathsf{sid}, P_S, P_R, m), \\ (\mathsf{Approve}) \leftarrow \Pi_{\text{A}}.\mathsf{RequestWarrant}(\mathsf{sid}, w), \\ (\mathsf{NotifyWarrant}, t(w)) \leftarrow \Pi_{\text{A}}.\mathsf{ActivateWarrant}(\mathsf{sid}, w), \\ 1 \leftarrow \Pi_{\text{A}}.\mathsf{VerifyWarrantStatus}(\mathsf{sid}, w, \mathsf{meta}, c) \end{matrix} \right\}$$

In this language, the statement comprises some specified metadata and a valid instance of the protocol Π_{A} from the perspectives of the parties P_{LE}, P_J, and the users P_i without the sender and receiver. This setup specifies all the relevant components of the protocol (including the ledger functionality, in the case

[8] As specified in the ideal functionality, during verification it will be checked that a warrant was properly requested and activated.

of the protocol presented in Section 6). The witness is a valid transcript starting with that setup, that includes the sending party sending a message with the appropriate metadata and concludes with a call to Π_{A}.VerifyWarrantStatus that returns 1. Note that if VerifyWarrantStatus returns 1, then in the real protocol, AccessMessage would return the relevant plaintext. Unlike other common witness encryption languages, we note that all correctly sampled statements are trivially in the language and have multiple witnesses. Therefore, we need the strong notion of extractable witness encryption. As we will discuss, finding a witness for the statement remains a difficult task.

Consider the implications if it were computationally feasible for an adversary to generate a witness for an honestly sampled statement for L. This would imply that an adversary corrupting P_{LE} and P_J interacting with the real protocol has a correct witness, which includes a call to ActivateWarrant, this implies our accountability property. Such a protocol could never succeed in meeting our original goals; law enforcement would always be able to simulate the steps required for proper accountability. An accountability mechanism that can be locally simulated cannot guarantee that all parties can monitor the mechanism, undermining the purpose of the protocol.

To formalize this intuition, we begin by describing an extractable witness encryption scheme Π_{EWE} for language L given access to an ARLEAS protocol Π_{A}.

- $\mathsf{Enc}(x, m)$ parses $(\mathsf{meta}, \mathsf{sid})$ from x and calls $\Pi_{\mathrm{A}}.\mathsf{SendMessage}(\mathsf{sid}, m, P_S, P_R)$ such that it outputs meta, c. It then returns the views $\{\mathcal{V}_{P_{\mathrm{LE}}}, \mathcal{V}_{P_J}, \mathcal{V}_{P_0}, \ldots, \mathcal{V}_{P_n}\}$ resulting from that run, excluding the private information associated with sending the message.

- $\mathsf{Dec}(c, \omega)$ parses $c, w, \mathsf{meta}, \mathsf{sid}$ from c and ω, calls $m \leftarrow \Pi_{\mathrm{A}}.\mathsf{AccessMessage}(\mathsf{sid}, w, \mathsf{meta}, c)$ and returns the result.

It is easy to see that this construction satisfies the correctness property of extractable witness encryption. Notice that a valid witness needs to contain inputs to VerifyWarrantStatus such that it outputs 1. Because VerifyWarrantStatus is defined to return 1 exactly when AccessMessage will return a message, the above decryption algorithm will return a message only with a valid witness.

We introduce the metadata in the statement in order to fix a witness to a particular statement. Note that our protocol generates an encryption as running part of the protocol, actually generating part of the witness. If metadata is not included in the statement, then *any* witness for a particular setup can be used to decrypt *any* ciphertext generated by the encryption oracle under the same statement. While this is not inherently problematic for extractable witness encryption, it no longer corresponds neatly to ARLEAS. Recall that warrants in ARLEAS specify the metadata for which they are relevant through the warrant scope check functionality $\theta(\cdot, \cdot)$ and this property must be enforced in the language. We now proceed to show that the above scheme Π_{EWE} satisfies extractable security if Π_{A} UC-realizes $\mathcal{F}_{\mathrm{ARLEAS}}^{v,t,p,\theta,\mathsf{ret}}$.

Theorem 4 *Given a protocol Π_A that UC-realizes $\mathcal{F}_{ARLEAS}^{v,t,p,\theta,\mathsf{ret}}$, Π_{EWE} is a secure extractable witness encryption scheme for the language L.*

Proof 1 *Given an adversary $\mathcal{A}$ with non-negligible advantage in the extractable witness encryption game for language L, either*

1. *We construct an extractor $\mathsf{Ext}_{\mathcal{A}}(1^{\lambda}, x, aux)$ by verifying if the adversary $\mathcal{A}$ ran $\Pi_A.\mathsf{RequestWarrant}(\mathsf{sid}, w)$ and $\Pi_A.\mathsf{ActivateWarrant}(\mathsf{sid}, w)$ such that $\Pi_A.\mathsf{VerifyWarrantStatus}(\mathsf{sid}, w, \mathsf{meta}, c) = 1$. If this was the case, the extractor would have all information to form a witness that it can output;*

2. *else, if such extractor does not exist, we construct a distinguisher $\mathcal{Z}$ that distinguishes between Π_A and ARLEAS ideal functionality. $\mathcal{Z}$ proceeds as follows*

   (a) *When $\mathcal{A}$ asks to sample a statement, $\mathcal{Z}$ instantiates Π_A with parties $\{P_{LE}, P_J, P_0, \ldots P_n, P_S, P_R\}$ on honest random coins. $\mathcal{Z}$ then generates some arbitrary metadata* meta *associated with a message that P_S could send in the future. and returns* meta, sid *to $\mathcal{A}$.*

(b) *When $\mathcal{A}$ sends the challenge plaintexts m_0, m_1 (such that $|m_0| = |m_1|$) on statement x, $\mathcal{Z}$ then flips a coin $b \xleftarrow{\$} \{0,1\}$, $\mathcal{Z}$ has P_S call*

$$\Pi_A.\mathsf{SendMessage}(\mathsf{sid}, m_b, P_S, P_R)$$

such that it outputs c, meta. $\mathcal{Z}$ then returns the updated views of P_{LE}, P_J and the N other users to $\mathcal{A}$.

(c) *When $\mathcal{A}$ outputs the guess b' and halts, $\mathcal{Z}$ outputs $b' == b$, where 1 indicates the real world and 0 indicates the ideal world.*

Note that in the ideal functionality, the joint views of law enforcement and the judge contain no information about the plaintext, because the ciphertext is chosen by the ideal world adversary without access to the plaintext. As such, if the adversary is able to distinguish between messages with non-negligible probability, $\mathcal{Z}$ must be interacting with the real world protocol.

Implications For Practical Retrospective ARLEAS. The relationship between retrospective ARLEAS and extractable witness encryption is an indication of the difficulty of realizing retrospective ARLEAS in practice. In very specific cases, it may be possible to phrase certain existing encryption schemes as witness encryption schemes, for example some IBE schemes. General purpose extractable witness encryption, on the other hand, is considered implausible [GGHW14]. The extractable witness encryption language we have described above must reason over the ledger authentication language and the various functionalities that parameterize an retrospective ARLEAS system. As such, the difficulty of realizing a practical retrospective ARLEAS will hinge on the complexity of the ledger and the parameterizing functionalities. If they are centralized and simple, it may be possible to instantiate an retrospective ARLEAS using the protocol we provided in Section 6 and known encryption techniques. However, the security provided by a centralized ledger is not significant, as a compromised central authority could circumvent the accountability properties of the system. Thus, we believe that this result indicates that instantiating an retrospective ARLEAS with meaningful security is impractical with known techniques.

8 Acknowledgments

The first author funded in part from the National Science Foundation under awards CNS-1653110 and CNS-1801479, a Google Security & Privacy Award. The second author is supported by the National Science Foundation under Grant #2030859 to the Computing Research Association for the CIFellows Project. The second author would like to thank Eliana Pfeffer for her help understanding the legal context of this work. Additionally, this material is based upon work supported by DARPA under Agreements No. HR00112020021 and Agreements No. HR001120C0084. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the United States Government or DARPA.

References

[AAB+15] Harold Abelson, Ross Anderson, Steven M. Bellovin, Josh Benaloh, Matt Blaze, Whitfield Diffie, John Gilmore, Matthew Green, Susan Landau, Peter G. Neumann, Ronald L. Rivest, Jeffrey I. Schiller, Bruce Schneier, Michael A. Specter, and Daniel J. Weitzner. Keys under doormats: mandating insecurity by requiring government access to all data and communications. *Journal of Cybersecurity*, 1(1):69–79, 11 2015.

[ADMM14] Marcin Andrychowicz, Stefan Dziembowski, Daniel Malinowski, and Lukasz Mazurek. Secure multiparty computations on bitcoin. In *2014 IEEE Symposium on Security and Privacy*, pages 443–458. IEEE Computer Society Press, May 2014.

[AMPR14] Arash Afshar, Payman Mohassel, Benny Pinkas, and Ben Riva. Non-interactive secure computation based on cut-and-choose. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 387–404. Springer, Heidelberg, May 2014.

[Appa] Apple. Facetime. Available at `https://apps.apple.com/us/app/facetime/id1110145091`.

[Appb] Apple. icloud security overview. Available at `https://support.apple.com/en-us/HT202303`.

[Appc] Apple. imessage. Available at `https://support.apple.com/explore/messages`.

[Bar19] William Barr. Attorney General William P. Barr Delivers Keynote Address at the International Conference on Cyber Security. Available at `https://www.justice.gov/opa/speech/attorney-general-william-p-barr-delivers-keynote-address-international-conference-cyber`, July 2019.

[BBB+18] Steven M. Bellovin, Matt Blaze, Dan Boneh, Susan Landau, and Ronald R. Rivest. Analysis of the CLEAR protocol per the National Academies' framework. Technical Report CUCS-003-18, Columbia University, May 2018.

[BBBF18] Dan Boneh, Joseph Bonneau, Benedikt Bünz, and Ben Fisch. Verifiable delay functions. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part I*, volume 10991 of *LNCS*, pages 757–788. Springer, Heidelberg, August 2018.

[BBS+12] Adam M. Bates, Kevin R. B. Butler, Micah Sherr, Clay Shields, Patrick Traynor, and Dan S. Wallach. Accountable wiretapping -or- I know they can hear you now. In *NDSS 2012*. The Internet Society, February 2012.

[BCP03] Emmanuel Bresson, Dario Catalano, and David Pointcheval. A simple public-key cryptosystem with a double trapdoor decryption mechanism and its applications. In Chi-Sung Laih, editor, *ASIACRYPT 2003*, volume 2894 of *LNCS*, pages 37–54. Springer, Heidelberg, November / December 2003.

[BCP14] Elette Boyle, Kai-Min Chung, and Rafael Pass. On extractability obfuscation. In Yehuda Lindell, editor, *TCC 2014*, volume 8349 of *LNCS*, pages 52–73. Springer, Heidelberg, February 2014.

[BCS05] Michael Backes, Jan Camenisch, and Dieter Sommer. Anonymous yet accountable access control. In *Proceedings of the 2005 ACM Workshop on Privacy in the Electronic Society*, WPES '05, pages 40–46, New York, NY, USA, 2005. Association for Computing Machinery.

[BGK+18] Christian Badertscher, Peter Gazi, Aggelos Kiayias, Alexander Russell, and Vassilis Zikas. Ouroboros genesis: Composable proof-of-stake blockchains with dynamic availability. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018*, pages 913–930. ACM Press, October 2018.

[BHY09] Mihir Bellare, Dennis Hofheinz, and Scott Yilek. Possibility and impossibility results for encryption and commitment secure under selective opening. In Antoine Joux, editor, *EUROCRYPT 2009*, volume 5479 of *LNCS*, pages 1–35. Springer, Heidelberg, April 2009.

[BK14] Iddo Bentov and Ranjit Kumaresan. How to use bitcoin to design fair protocols. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part II*, volume 8617 of *LNCS*, pages 421–439. Springer, Heidelberg, August 2014.

[BKPW12] Mihir Bellare, Eike Kiltz, Chris Peikert, and Brent Waters. Identity-based (lossy) trapdoor functions and applications. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 228–245. Springer, Heidelberg, April 2012.

[BL06] Cassell Bryan-Low. Vodafone, Ericsson Get Hung Up In Greece's Phone-Tap Scandal. *The Wall Street Journal*, June 2006.

[Bla96] Matt Blaze. Oblivious key escrow. In Ross Anderson, editor, *Information Hiding*, pages 335–343, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg.

[BMTZ17] Christian Badertscher, Ueli Maurer, Daniel Tschudi, and Vassilis Zikas. Bitcoin as a transaction ledger: A composable treatment. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part I*, volume 10401 of *LNCS*, pages 324–356. Springer, Heidelberg, August 2017.

[BR99] Mihir Bellare and Ronald L. Rivest. Translucent cryptography - an alternative to key escrow, and its implementation via fractional oblivious transfer. *Journal of Cryptology*, 12(2):117–139, March 1999.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145. IEEE Computer Society Press, October 2001.

[CDG+18] Jan Camenisch, Manu Drijvers, Tommaso Gagliardoni, Anja Lehmann, and Gregory Neven. The wonderful world of global random oracles. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part I*, volume 10820 of *LNCS*, pages 280–312. Springer, Heidelberg, April / May 2018.

[CF01] Ran Canetti and Marc Fischlin. Universally composable commitments. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 19–40. Springer, Heidelberg, August 2001.

[CGJ+17] Arka Rai Choudhuri, Matthew Green, Abhishek Jain, Gabriel Kaptchuk, and Ian Miers. Fairness in an unfair world: Fair multiparty computation from public bulletin boards. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 719–728. ACM Press, October / November 2017.

[CGJ19] Arka Rai Choudhuri, Vipul Goyal, and Abhishek Jain. Founding secure computation on blockchains. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part II*, volume 11477 of *LNCS*, pages 351–380. Springer, Heidelberg, May 2019.

[CKN03] Ran Canetti, Hugo Krawczyk, and Jesper Buus Nielsen. Relaxing chosen-ciphertext security. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 565–582. Springer, Heidelberg, August 2003.

[CLOS02] Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. Universally composable two-party and multi-party secure computation. In *34th ACM STOC*, pages 494–503. ACM Press, May 2002.

[CS02] Ronald Cramer and Victor Shoup. Universal hash proofs and a paradigm for adaptive chosen ciphertext secure public-key encryption. In Lars R. Knudsen, editor, *EUROCRYPT 2002*, volume 2332 of *LNCS*, pages 45–64. Springer, Heidelberg, April / May 2002.

[DB96] Dorothy E Denning and Dennis K Branstad. A taxonomy for key escrow encryption systems. *Communications of the ACM*, 39(3):34–40, 1996.

[DDO+01] Alfredo De Santis, Giovanni Di Crescenzo, Rafail Ostrovsky, Giuseppe Persiano, and Amit Sahai. Robust non-interactive zero knowledge. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 566–598. Springer, Heidelberg, August 2001.

[Den94] Dorothy E. Denning. The US key escrow encryption technology. *Computer Communications*, 17(7):453–457, 1994.

[DGKR18] Bernardo David, Peter Gazi, Aggelos Kiayias, and Alexander Russell. Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part II*, volume 10821 of *LNCS*, pages 66–98. Springer, Heidelberg, April / May 2018.

[DMNS06] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. In Shai Halevi and Tal Rabin, editors, *TCC 2006*, volume 3876 of *LNCS*, pages 265–284. Springer, Heidelberg, March 2006.

[DNO09] Ivan Damgård, Jesper Buus Nielsen, and Claudio Orlandi. Essentially optimal universally composable oblivious transfer. In Pil Joong Lee and Jung Hee Cheon, editors, *ICISC 08*, volume 5461 of *LNCS*, pages 318–335. Springer, Heidelberg, December 2009.

[Dwo08] Cynthia Dwork. Differential privacy: A survey of results. In *International conference on theory and applications of models of computation*, pages 1–19. Springer, 2008.

[Enc] EncroChat. Encrochat network. `http://encrochat.network/`.

[Enc19] Encryption Working Group. Moving the Encryption Policy Conversation Forward. Technical report, Carnegie Endowment for International Peace, 2019.

[FB14] Lorenzo Franceschi-Bicchierai. FBI Director: Encryption Will Lead to a 'Very Dark Place'. *Mashable*, October 2014.

[Fed] Federal Bureau of Investigation. Going Dark. Available at `https://www.fbi.gov/services/operational-technology/going-dark`.

[FPS+18] Jonathan Frankle, Sunoo Park, Daniel Shaar, Shafi Goldwasser, and Daniel J. Weitzner. Practical accountability of secret processes. In William Enck and Adrienne Porter Felt, editors, *USENIX Security 2018*, pages 657–674. USENIX Association, August 2018.

[FW18] Joan Feigenbaum and Daniel J Weitzner. On the incommensurability of laws and technical mechanisms: Or, what cryptography can't do. In *Cambridge International Workshop on Security Protocols*, pages 266–279. Springer, 2018.

[GG17] Rishab Goyal and Vipul Goyal. Overcoming cryptographic impossibility results using blockchains. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017, Part I*, volume 10677 of *LNCS*, pages 529–561. Springer, Heidelberg, November 2017.

[GGHW14] Sanjam Garg, Craig Gentry, Shai Halevi, and Daniel Wichs. On the implausibility of differing-inputs obfuscation and extractable witness encryption with auxiliary input. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part I*, volume 8616 of *LNCS*, pages 518–535. Springer, Heidelberg, August 2014.

[GK96] Oded Goldreich and Hugo Krawczyk. On the composition of zero-knowledge proof systems. *SIAM J. Comput.*, 25(1):169–192, February 1996.

[GKZ19] Peter Gazi, Aggelos Kiayias, and Dionysis Zindros. Proof-of-stake sidechains. In *2019 IEEE Symposium on Security and Privacy*, pages 139–156. IEEE Computer Society Press, May 2019.

[GLW14] Craig Gentry, Allison B. Lewko, and Brent Waters. Witness encryption from instance independent assumptions. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part I*, volume 8616 of *LNCS*, pages 426–443. Springer, Heidelberg, August 2014.

[GO94] Oded Goldreich and Yair Oren. Definitions and properties of zero-knowledge proof systems. *Journal of Cryptology*, 7(1):1–32, December 1994.

[Goo] Google. Encrypt your data - pixel phone help. Available at `https://support.google.com/pixelphone/answer/2844831?hl=en`.

[Gor13] Siobhan Gorman. NSA Officers Spy on Love Interests. *The Wall Street Journal*, August 2013.

[GOVW12] Sanjam Garg, Rafail Ostrovsky, Ivan Visconti, and Akshay Wadia. Resettable statistical zero knowledge. In Ronald Cramer, editor, *TCC 2012*, volume 7194 of *LNCS*, pages 494–511. Springer, Heidelberg, March 2012.

[GP17] Shafi Goldwasser and Sunoo Park. Public accountability vs. secret laws: Can they coexist? a cryptographic proposal. In *Proceedings of the 2017 on Workshop on Privacy in the Electronic Society*, WPES '17, pages 99–110, New York, NY, USA, 2017. Association for Computing Machinery.

[Gra20] Sen. Lindsey Graham. Eliminating abusive and rampant neglect of interactive technologies act of 2020. `https://www.congress.gov/bill/116th-congress/senate-bill/3398/text`, March 2020.

[HK07] Omer Horvitz and Jonathan Katz. Universally-composable two-party computation in two rounds. In Alfred Menezes, editor, *CRYPTO 2007*, volume 4622 of *LNCS*, pages 111–129. Springer, Heidelberg, August 2007.

[HLOV11] Brett Hemenway, Benoît Libert, Rafail Ostrovsky, and Damien Vergnaud. Lossy encryption: Constructions from general assumptions and efficient selective opening chosen ciphertext security. In Dong Hoon Lee and Xiaoyun Wang, editors, *ASIACRYPT 2011*, volume 7073 of *LNCS*, pages 70–88. Springer, Heidelberg, December 2011.

[Hof12] Dennis Hofheinz. All-but-many lossy trapdoor functions. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 209–227. Springer, Heidelberg, April 2012.

[HPRV19] Thibaut Horel, Sunoo Park, Silas Richelson, and Vinod Vaikuntanathan. How to subvert backdoored encryption: Security against adversaries that decrypt all ciphertexts. In Avrim Blum, editor, *ITCS 2019*, volume 124, pages 42:1–42:20. LIPIcs, January 2019.

[IKO+11] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, Manoj Prabhakaran, and Amit Sahai. Efficient non-interactive secure computation. In Kenneth G. Paterson, editor, *EUROCRYPT 2011*, volume 6632 of *LNCS*, pages 406–425. Springer, Heidelberg, May 2011.

[Kam14] Seny Kamara. Restructuring the NSA metadata program. In Rainer Böhme, Michael Brenner, Tyler Moore, and Matthew Smith, editors, *FC 2014 Workshops*, volume 8438 of *LNCS*, pages 235–247. Springer, Heidelberg, March 2014.

[KFB14] J Kroll, E Felten, and Dan Boneh. Secure protocols for accountable warrant execution. 2014.

[KGM19] Gabriel Kaptchuk, Matthew Green, and Ian Miers. Giving state to the stateless: Augmenting trustworthy computation with ledgers. In *NDSS 2019*. The Internet Society, February 2019.

[KRDO17] Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. Ouroboros: A provably secure proof-of-stake blockchain protocol. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part I*, volume 10401 of *LNCS*, pages 357–388. Springer, Heidelberg, August 2017.

[KZW+] Joshua A Kroll, Joe Zimmerman, David J Wu, Valeria Nikolaenko, Edward W Felten, and Dan Boneh. Accountable cryptographic access control.

[LG16] Eric Lichtblau and Joseph Goldstein. Apple Faces U.S. Demand to Unlock 9 More iPhones. *The New York Times*, February 2016.

[LJKW18] Jia Liu, Tibor Jager, Saqib A. Kakvi, and Bogdan Warinschi. How to build time-lock encryption. *Designs, Codes and Cryptography*, 86(11):2549–2586, Nov 2018.

[LR18] Ian Levy and Crispin Robinson. Principles for a more informed exceptional access debate. *Lawfare*, Thursday 2018.

[LRC14] J. Liu, M. D. Ryan, and L. Chen. Balancing societal security and individual privacy: Accountable escrow system. In *2014 IEEE 27th Computer Security Foundations Symposium*, pages 427–440, July 2014.

[Nak08] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008. 2008.

[Nak13] Ellen Nakashima. Chinese hackers who hacked Google gained access to sensitive data, U.S. officials say. *The Washington Post*, May 2013.

[Nat16] National Academies of Sciences, Engineering, and Medicine. *Exploring Encryption and Potential Mechanisms for Authorized Government Access to Plaintext*. The National Academies Press, 2016.

[Nat18] National Academies of Sciences, Engineering, and Medicine. *Decrypting the Encryption Debate: A Framework for Decision Makers*. The National Academies Press, Washington, DC, 2018.

[Nig11] Johnathan Nightingale. Fraudulent *.google.com Certificate, August 2011.

[NO09] Jesper Buus Nielsen and Claudio Orlandi. LEGO for two-party secure computation. In Omer Reingold, editor, *TCC 2009*, volume 5444 of *LNCS*, pages 368–386. Springer, Heidelberg, March 2009.

[Ore87] Yair Oren. On the cunning power of cheating verifiers: Some observations about zero knowledge proofs (extended abstract). In *28th FOCS*, pages 462–471. IEEE Computer Society Press, October 1987.

[Ped92] Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In Joan Feigenbaum, editor, *CRYPTO'91*, volume 576 of *LNCS*, pages 129–140. Springer, Heidelberg, August 1992.

[Pop16] Cody M. Poplin. Burr-feinstein encryption legislation officially released. *Lawfare*, April 2016.

[PVMB19] Gaurav Panwar, Roopa Vishwanathan, Satyajayant Misra, and Austin Bos. SAMPL: Scalable auditability of monitoring processes using public ledgers. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 2249–2266. ACM Press, November 2019.

[PW08] Chris Peikert and Brent Waters. Lossy trapdoor functions and their applications. In Richard E. Ladner and Cynthia Dwork, editors, *40th ACM STOC*, pages 187–196. ACM Press, May 2008.

[Sah99] Amit Sahai. Non-malleable non-interactive zero knowledge and adaptive chosen-ciphertext security. In *40th FOCS*, pages 543–553. IEEE Computer Society Press, October 1999.

[Sav18] Stefan Savage. Lawful device access without mass surveillance risk: A technical design discussion. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, pages 1761–1774, New York, NY, USA, 2018. Association for Computing Machinery.

[Sca19] Alessandra Scafuro. Break-glass encryption. In Dongdai Lin and Kazue Sako, editors, *PKC 2019, Part II*, volume 11443 of *LNCS*, pages 34–62. Springer, Heidelberg, April 2019.

[SFF14] Aaron Segal, Bryan Ford, and Joan Feigenbaum. 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 14)*, San Diego, CA, August 2014. USENIX Association.

[Sig] Signal. Signal secure messaging system.

[Sin20] Manish Sing. Over two dozen encryption experts call on India to rethink changes to its intermediary liability rules. *TechCrunch*, February 2020.

[SLG20] Sen. Marsha Blackburn Sen. Lindsey Graham, Sen. Tom Cotton. Lawful access to 5 encrypted data act. `https://www.judiciary.senate.gov/press/rep/releases/graham-cotton-blackburn-introduce-balanced-solution-to-bolster-national-security-end-use-of-warrant-proof-encryption-that-shields-criminal-activity`, June 2020.

[SSW19] Sacha Servan-Schreiber and Archer Wheeler. Judge, jury & encryptioner: Exceptional access with a fixed social cost, 2019.

[Tai16] Matt Tait. An approach to James Comey's technical challenge. *Lawfare*, April 2016.

[Tar18] Jamie Tarabay. Australian Government Passes Contentious Encryption Law. *The New York Times*, December 2018.

[Wha17] WhatsApp. WhatsApp Encryption Overview. Available at `https://scontent.whatsapp.net/v/t61/68135620_760356657751682_6212997528851833559_n.pdf/WhatsApp-Security-Whitepaper.pdf`, December 2017.

[WMT15] Nicholas Watt, Rowena Mason, and Ian Traynor. David Cameron pledges anti-terror law for internet after Paris attacks. *The Guardian*, January 2015.

[WV18] Charles Wright and Mayank Varia. Crypto crumple zones: Enabling limited access without mass surveillance. In *2018 IEEE European Symposium on Security and Privacy (EuroS P)*, pages 288–306, April 2018.

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *27th FOCS*, pages 162–167. IEEE Computer Society Press, October 1986.

A Proof of Theorem 1

Proof 2 *We prove that the above construction securely realizes* $\mathcal{F}_{ARLEAS}^{v,t,p,\theta,pro}$ *by showing that there does not exist a distinguisher* $\mathcal{Z}$ *that can distinguish between an interaction with the ideal functionality and a simulator* $\mathcal{S}$ *and the real protocol* $\pi_{PRO}^{v,t,p,\theta}$*. We define the interaction with the real protocol as follows: The experiment is initialized with* N *users* $P_1, \ldots P_N$*, law enforcement* P_{LE} *and a judge* P_J*. The adversary* $\mathcal{A}$ *chooses a subset of these users to corrupt. Then, users run* $\pi_{PRO}^{v,t,p,\theta}$.Setup*, with* $\mathcal{A}$ *controlling the actions of the corrupted parties. Honest users then, according to their arbitrary strategy, run* $\pi_{PRO}^{v,t,p,\theta}$.SendMessage *to exchange messages with other users. Law enforcement interacts with the judge to get warrants via* $\pi_{PRO}^{v,t,p,\theta}$.RequestWarrant *and uses* $\pi_{PRO}^{v,t,p,\theta}$.ActivateWarrant *to start surveilling a user. Honest parties follow an arbitrary strategy, but follow the protocol and corrupted parties are controlled by the adversary.*

We start our proof by first considering the case where a single user P_i *is corrupted.*

P_i is corrupted.

We begin by showing that $\pi_{PRO}^{v,t,p,\theta}$ *UC-realizes the* $\mathcal{F}_{ARLEAS}^{v,t,p,\theta,pro}$ *when a user* P_i *is compromised. We construct the simulator* $\mathcal{S}$ *as follows:*

1. *$\mathcal{S}$ generates the common reference string for the NIZK scheme directly, and stores the trapdoor τ. $\mathcal{S}$ runs $(\mathsf{pk}_{sign}, \mathsf{sk}_{sign}) \leftarrow \Pi_{Sign}.\mathsf{KeyGen}(1^\lambda)$ as the judge would in the real protocol, and outputs pk_{sign} to the adversary $\mathcal{A}$. $\mathcal{S}$ initializes an instance of the ideal functionality in prospective mode. Finally, $\mathcal{S}$ runs $\pi_{PRO}^{v,t,p,\theta}.\mathsf{ActivateWarrant}$ with an empty active warrants set.*

2. *$\mathcal{S}$ computes $(\mathsf{pk}_j, \mathsf{sk}_j) \leftarrow \Pi_{Enc}.\mathsf{KeyGen}(1^\lambda)$ and samples a unique $tag_j \leftarrow \mathbb{Z}_p$ for all honest P_j and sends (pk_j, tag_j) to $\mathcal{A}$. Additionally, $\mathcal{S}$ waits to receive pk_i, tag_i from P_i.*

3. *When $\mathcal{S}$ receives $(c_1, c_2, \pi, \mathsf{meta})$ from $\mathcal{A}$ intended for uncorrupted P_j, $\mathcal{S}$ begins by verifying π and checking that meta is correct. If these checks pass, set $b \leftarrow 1$, and $b \leftarrow 0$ otherwise. $\mathcal{S}$ computes $m \leftarrow \Pi_{Enc}.\mathsf{Dec}(\mathsf{sk}_j, c_1)$. $\mathcal{S}$ then sends $(\mathsf{SendMessage}, P_j, m, b)$ to the ideal functionality.*

4. *Upon receiving $(\mathsf{Sent}, \mathsf{meta}, c, m)$ from the ideal functionality, $\mathcal{S}$ computes $(c_1, c_2, \pi, \mathsf{meta})$ using $\pi_{PRO}^{v,t,p,\theta}$.$\mathsf{SendMessage}$ and forwards it to P_i.*

5. *Upon receiving $(\mathsf{NotifyWarrant}, t(w))$ from $\mathcal{F}_{ARLEAS}^{v,t,p,\theta,\mathit{pro}}$, $\mathcal{S}$ aggregates $t(w)$ from all $(\mathsf{NotifyWarrant}, \cdot)$ messages seen so far into info. $\mathcal{S}$ then randomly chooses a set of tags $\mathcal{T}$ such that $|\mathcal{T}| = |\mathsf{info}|$ and computes $(mpk, msk) \leftarrow \Pi_{LTE}.\mathsf{KeyGen}(1^\lambda, \mathcal{T})$. $\mathcal{S}$ then simulates the proof π and sends $(\mathsf{Post}, (mpk, \pi, \mathsf{info}))$ to $\mathcal{L}$.*

We proceed with a hybrid argument. Let $\mathcal{H}_0$ denote the distribution of the view of $\mathcal{A}$ in the real world interaction.

$\mathcal{H}_1$: Let $\mathcal{H}_1$ be the same as $\mathcal{H}_0$, but instead of having the common reference string generated by the $\mathcal{F}_{CRS}^{\Pi_{NIZK}.\mathsf{ZKSetup}}$, the common reference string is generated using $(\mathsf{crs}, \tau) \leftarrow \Pi_{NIZK}.\mathsf{ZKSetup}(1^\lambda)$. Note that the common reference string is selected from exactly the same distribution, so the difference in the distribution of the view of $\mathcal{A}$ between $\mathcal{H}_0$ and $\mathcal{H}_1$ is 0.

$\mathcal{H}_2$: Let $\mathcal{H}_2$ be the same as $\mathcal{H}_1$, but when an honest P_{LE} sends $(\mathsf{Post}, (mpk, \pi, \mathsf{info}))$to $\mathcal{L}$, the proof π is instead simulated with τ. Because of the perfect zero-knowledge property of Π_{NIZK}, the adversary's view in $\mathcal{H}_2$ and $\mathcal{H}_1$ is statistically close.

$\mathcal{H}_3$: Let $\mathcal{H}_3$ be the same as $\mathcal{H}_2$, but when an honest P_{LE} sends $(\mathsf{Post}, (mpk, \pi, \mathsf{info}))$to $\mathcal{L}$, let mpk be generated with an random set of tags with the correct size. Because of the indistinguishability of tag sets property of Π_{LTE}, the difference in the distribution in the view of the $\mathcal{A}$ is negligible.

Because the view of $\mathcal{A}$ in $\mathcal{H}_3$ is distributed the same as in the ideal world with simulator $\mathcal{S}$, the proof is done. Notice that this proof extends directly to multiple corrupt users, as the views of the users are independent, except when they send messages to each other. However, such messages do not require simulation. Thus it suffices to simulate each one independently.

P_{LE} is corrupted. *We now extend the previous proof to include corrupted P_{LE}. We extend $\mathcal{S}$ to simulate the view of P_{LE}. Note that step 5 described in $\mathcal{S}$ above is no longer applicable for corrupted users, as the notification mechanism from the actual protocol will look correct.*

1. *$\mathcal{S}$ generates the common reference string for the NIZK scheme directly, and stores the trapdoor τ. Then, $\mathcal{S}$ sets $\mathsf{ValidParameters}$ to true. $\mathcal{S}$ then runs $\Pi_{Sign}.\mathsf{KeyGen}(1^\lambda) \rightarrow (\mathsf{pk}_{sign}, \mathsf{sk}_{sign})$ as the judge would in the read protocol, and outputs pk_{sign} to the adversary $\mathcal{A}$. $\mathcal{S}$ initializes an instance of the ideal functionality in prospective mode. $\mathcal{S}$ computes $(\mathsf{pk}_j, \mathsf{sk}_j) \leftarrow \Pi_{Enc}.\mathsf{KeyGen}(1^\lambda)$ and samples a unique $tag_j \leftarrow \mathbb{Z}_p$ for all honest P_j and sends (pk_j, tag_j) to $\mathcal{A}$. When $\mathcal{S}$ detects $(mpk, \pi, \mathsf{info})$ posted on $\mathcal{L}$, it verifies the proof π, and sets $\mathsf{ValidParameters}$ to false if it does not verify or $|\mathsf{info}| \neq 0$. $\mathcal{S}$ then initializes an empty warrant table W.*

2. *We split the case of receiving $(\mathsf{Sent}, \mathsf{meta}, c)$ from $\mathcal{F}_{ARLEAS}^{v,t,p,\theta,\mathit{pro}}$ intended for P_{LE} into three cases. All begin by $\mathcal{S}$ extracting the recipient P_j from meta:*

   (a) If $\mathsf{ValidParameters}$ is false, $\mathcal{S}$ silently drops the message.

(b) *If* ValidParameters *is true and P_j is not corrupted, $\mathcal{S}$ and chooses a message m_0 and computes the ciphertext $(c_1, c_2, \pi, \mathsf{meta})$ by encrypting m_0 using $\pi_{PRO}^{v,t,p,\theta}.\mathsf{SendMessage}(\mathsf{pk}_j, tag_j, m_0)$.*

(c) *If* ValidParameters *is true and P_j is corrupted, $\mathcal{S}$ will also receive $(\mathsf{Sent}, \mathsf{meta}, c, m)$ from the ideal functionality, intended for P_j. $\mathcal{S}$ then encrypts m using $\pi_{PRO}^{v,t,p,\theta}.\mathsf{SendMessage}(\mathsf{pk}_j, tag_j, m)$*

Finally, $\mathcal{S}$ sends the resulting ciphertext to $\mathcal{A}$.

3. *Upon receiving $(\mathsf{Sent}, \mathsf{meta}, c, 0)$ from $\mathcal{F}_{ARLEAS}^{v,t,p,\theta,\mathsf{pro}}$ intended for P_{LE}, $\mathcal{S}$ samples a random message and creates a ciphertext with $\pi_{PRO}^{v,t,p,\theta}.\mathsf{SendMessage}$. Then, it generates a false proof instead of the real proof.*

4. *Upon receiving $(\mathsf{Sent}, \mathsf{meta}, c, m)$ from $\mathcal{F}_{ARLEAS}^{v,t,p,\theta,\mathsf{pro}}$ due to an active warrant, if* ValidParameters *is true, $\mathcal{S}$ calls the send message algorithm of the real protocol on $\pi_{PRO}^{v,t,p,\theta}.\mathsf{SendMessage}(\mathsf{pk}_j, tag_j, m)$ and sends the output to $\mathcal{A}$.*

5. *Upon receiving $(\mathsf{RequestWarrant}, tag_j)$ from the adversary, intended for P_J, $\mathcal{S}$ generates a warrant w for the ideal functionality that only targets P_j and sends $(\mathsf{RequestWarrant}, w)$ to $\mathcal{F}_{ARLEAS}^{v,t,p,\theta,\mathsf{pro}}$. If $\mathcal{F}_{ARLEAS}^{v,t,p,\theta,\mathsf{pro}}$ answers with $(\mathsf{Approve})$, $\mathcal{S}$ uses the sk_{sign} for P_J to form and sign (tag_j, σ) as in the real protocol and adds (w, False) to W. If $\mathcal{F}_{ARLEAS}^{v,t,p,\theta,\mathsf{pro}}$ responds with $(\mathsf{Disapprove})$, $\mathcal{S}$ send $\perp$ to the adversary.*

6. *$\mathcal{S}$ monitors $\mathcal{L}$. Upon seeing a new post on the ledger, $\mathcal{S}$ performs the following steps*

   (a) *Retrieve the post by sending $(\mathsf{GetCounter})$ to $\mathcal{L}$ and receive the current counter ℓ. Then query $\mathcal{L}$ on (GetVal, ℓ) to receive the latest posting $(\ell, (mpk, \pi, \mathsf{info}), \pi_{\mathsf{publish}})$*

   (b) *Verify $\Pi_{NIZK}.\mathsf{ZKVerify}(mpk, \mathsf{info}, \pi) = 1$. If the proof does not verify, the $\mathcal{S}$ sets* ValidParameters *to false and halts.*

   (c) *Set* ValidParameters *to true and run* Extract *to recover $(\mathcal{W}, \mathcal{T}, msk, r)$ from π. If extraction fails, the simulator halts with an error.*

   (d) *If there is an entry (w, False) in W for $w \in \mathcal{W}$ $\mathcal{S}$ sends $(\mathsf{ActivateWarrant}, w)$ to the ideal functionality and sets the entry to be (w, True).*

We proceed with a hybrid argument. Let $\mathcal{H}_0$ denote the distribution of the view of $\mathcal{A}$ in the real world interaction.

$\mathcal{H}_1$: Let $\mathcal{H}_1$ be the same as $\mathcal{H}_0$, but instead of having the common reference string generated by the $\mathcal{F}_{CRS}^{\Pi_{NIZK}.\mathsf{ZKSetup}}$, the common reference string is generated using $(\mathsf{crs}, \tau) \leftarrow \Pi_{NIZK}.\mathsf{ZKSetup}(1^\lambda)$. Note that the common reference string is selected from exactly the same distribution, so the the distribution in the view of $\mathcal{A}$ between $\mathcal{H}_0$ and $\mathcal{H}_1$ is statistically close.

$\mathcal{H}_2$: Let $\mathcal{H}_2$ be the same as $\mathcal{H}_1$, except the proof of ciphertexts consistency in a ciphertext $(c_1, c_2, \pi,$ $\mathsf{meta})$ bound for an honest user for which there is no active warrant is simulated. Due to the zero-knowledge property of Π_{NIZK}, the difference in the view of the adversary in $\mathcal{H}_2$ and $\mathcal{H}_1$ is negligible.

$\mathcal{H}_3$: Let $\mathcal{H}_3$ be the same as $\mathcal{H}_2$, except when $\mathcal{S}$ receives a ciphertext $(c_1, c_2, \pi, \mathsf{meta})$bound for an honest user for which there is no active warrant, $\mathcal{S}$ samples a message m_0 that would result in the same metadata and sets $c_1 \leftarrow \Pi_{Enc}.\mathsf{Enc}(\mathsf{pk}_j, m_0; r_1)$. By the CCA security of Π_{Enc} the advantage of $\mathcal{A}$ in distinguishing between $\mathcal{H}_3$ and $\mathcal{H}_2$ is negligible.

$\mathcal{H}_4$: Let $\mathcal{H}_4$ be the same as $\mathcal{H}_3$, except the second ciphertext element c_2 in a ciphertext $(c_1, c_2, \pi, \mathsf{meta})$ bound for an honest user for which there is no active warrant is computed as $\Pi_{LTE}.\mathsf{Enc}(mpk, tag_j, m_0; r_2)$. By the lossy property of Π_{LTE}, $\mathcal{H}_4$ and $\mathcal{H}_3$ are statistically indistinguishable.

$\mathcal{H}_5$: Let $\mathcal{H}_5$ be the same as $\mathcal{H}_4$, except the proof of ciphertexts consistency in a ciphertext $(c_1, c_2, \pi, \mathsf{meta})$ bound for an honest user for which there is no active warrant is computed honestly with respect to the plaintext message m_0. Again, by the zero-knowledge property of Π_{NIZK}, the difference in the view of the adversary in $\mathcal{H}_5$ and $\mathcal{H}_4$ is negligible.

$\mathcal{H}_6$: *Let* $\mathcal{H}_6$ *be the same as* $\mathcal{H}_5$*, except when* $\mathcal{A}$ *detects* $(mpk, \pi, \mathsf{info})$ *being posted on the ledger,* $\mathcal{S}$ *attempts to run the extractor* $\Pi_{NIZK}.\mathsf{Extract}$ *and abort the experiment if it fails. However, because the extractor only fails with negligible probability, the difference in the view of the adversary between* $\mathcal{H}_6$ *and* $\mathcal{H}_5$ *is negligible.*

$\mathcal{H}_6$ *has the same distribution as* $\mathcal{S}$*, concluding the proof. In the real world,* P_{LE} *would be denied warrants at the same rate as in the ideal world, as an honest* P_J *handles warrant requests in the same way. One note is that law enforcement can "deactivate" warrants in way not possible in the ideal functionality. However, when warrants are deactivated, honestly encrypting the message still hides the plaintext from the adversary.*

P_J ***and*** P_{LE} ***are corrupted.*** *We now focus on the case when* P_J *and* P_{LE} *are both corrupted. Note that step 4 of the above simulator description is no longer relevant, as the warrant request is handled internally by* $\mathcal{A}$*. Our simulator requires on minor changes to steps 1 and 5, which we show below.*

1. *Do the setup as before, but waiting to receive* pk_{sign} *from* $\mathcal{A}$

⋮

5. $\mathcal{S}$ *monitors* $\mathcal{L}$*. Upon seeing a new post on the ledger,* $\mathcal{S}$ *performs the following steps*
   (a) *Retrieve the post by sending* $(\mathsf{GetCounter})$ *to* $\mathcal{L}$ *and receive the current counter* ℓ*. Then query* $\mathcal{L}$ *on* (GetVal, ℓ) *to receive the latest posting* $(\ell, (mpk, \pi, \mathsf{info}), \pi_{\mathsf{publish}})$
   (b) *Verify* $\Pi_{NIZK}.\mathsf{ZKVerify}(mpk, \mathsf{info}, \pi) = 1$*. If the proof does not verify, the* $\mathcal{S}$ *sets* $\mathsf{ValidParameters}$ *to false and halts.*
   (c) *Set* $\mathsf{ValidParameters}$ *to true and runs* $(\mathcal{W}, \mathcal{T}, msk, r) \leftarrow \mathsf{Extract}(\mathsf{CRS}_{ZK}, \tau, x, \pi)$*. If extraction fails, the simulator halts with an error.*
   (d) *for each warrant* $w \in \mathcal{W}$ *for which there does not exist an entry* (w, True) *in* W*, the* $\mathcal{S}$ *executes the following steps*
      i. $\mathcal{S}$ *generates a warrant* w *for the ideal functionality that only targets* P_j *and sends* $(\mathsf{RequestWarrant}, w)$ *to* $\mathcal{F}_{ARLEAS}^{v,t,p,\theta,\mathsf{pro}}$*.*
      ii. *When* $\mathcal{F}_{ARLEAS}^{v,t,p,\theta,\mathsf{pro}}$ *sends* $(\mathsf{ApproveWarrant}, w)$ *to the* $\mathcal{S}$ *intended to* P_J*,* $\mathcal{S}$ *responds* $(\mathsf{Approve})$ *on behalf of* P_J
      iii. $\mathcal{S}$ *sends* $(\mathsf{ActivateWarrant}, w)$ *to the ideal functionality*
      iv. $\mathcal{S}$ *adds* (w, True) *to* W

In fact, the hybrid argument above holds directly in this case as well. First notice that there are no additional messages that require simulation. By giving the adversary control of P_J*, we give it the ability to create valid warrants independently. However, there is no change in behavior expected until those warrants are activated. As such, those warrants can be requested from the ideal functionality right as they are being activated.*

B Proof of Theorem 2

Proof 3 *As above, we now prove that our construction securely realizes* $\mathcal{F}_{ARLEAS}^{v,t,p,\theta,\mathsf{pro}}$ *by showing that there does not exist a distinguisher* $\mathcal{Z}$ *that can distinguish between an interaction with the ideal functionality and a simulator* $\mathcal{S}$ *and the real protocol* $\pi_{PRO}^{v,t,p,\theta}$*. We define the interaction with the real protocol as above.*

We start our proof by first considering the case where a single user P_i *is corrupted.*

P_i ***is corrupted.***

We begin by showing that $\pi_{PRO}^{v,t,p,\theta}$ *UC-realizes the* $\mathcal{F}_{ARLEAS}^{v,t,p,\theta,\mathsf{pro}}$ *when a user* P_i *is compromised. We construct the simulator* $\mathcal{S}$ *as follows:*

1. *$\mathcal{S}$ generates the common reference string for the NIZK scheme and* $(\mathsf{CRS}_{\mathsf{NISC}}, \tau_{\mathsf{NISC}}) \leftarrow \Pi_{NISC}.\mathsf{GenCRS}(1^\lambda)$ *directly, and stores the trapdoors* $(\tau, \tau_{\mathsf{NISC}})$*. $\mathcal{S}$ runs* $(\mathsf{pk}_{sign}, \mathsf{sk}_{sign}) \leftarrow \Pi_{Sign}.\mathsf{KeyGen}(1^\lambda)$*as the judge would in the real protocol, and outputs* pk_{sign} *to the adversary $\mathcal{A}$. $\mathcal{S}$ initializes an instance of the ideal functionality in prospective mode. Finally, $\mathcal{S}$ runs* $\pi_{PRO}^{v,t,p,\theta}.\mathsf{ActivateWarrant}$ *with an empty active warrants set.*

2. *$\mathcal{S}$ computes* $(\mathsf{pk}_j, \mathsf{sk}_j) \leftarrow \Pi_{Enc}.\mathsf{KeyGen}(1^\lambda)$ *for all honest P_j and sends* pk_j *to $\mathcal{A}$. Additionally, $\mathcal{S}$ waits to receive* pk_i *from P_i.*

3. *When $\mathcal{S}$ receives* $(c_1, c_2, \pi, \mathsf{meta})$ *from $\mathcal{A}$ intended for uncorrupted P_j, $\mathcal{S}$ begins by verifying π and checking that* meta *is correct. If these checks pass, set $b \leftarrow 1$, and $b \leftarrow 0$ otherwise. $\mathcal{S}$ computes* $m \leftarrow \Pi_{Enc}.\mathsf{Dec}(\mathsf{sk}_j, c_1)$*. $\mathcal{S}$ then sends* $(\mathsf{SendMessage}, P_j, m, b)$ *to the ideal functionality.*

4. *Upon receiving* $(\mathsf{Sent}, \mathsf{meta}, c, m)$ *from the ideal functionality, $\mathcal{S}$ computes* $(c_1, c_2, \pi, \mathsf{meta})$ *using* $\pi_{PRO}^{v,t,p,\theta}$*.* SendMessage *and forwards it to P_i.*

5. *Upon receiving* $(\mathsf{NotifyWarrant}, t(w))$ *from* $\mathcal{F}_{ARLEAS}^{v,t,p,\theta,pro}$*, $\mathcal{S}$ aggregates $t(w)$ from all* $(\mathsf{NotifyWarrant}, \cdot)$ *messages seen so far into* info*. $\mathcal{S}$ then chooses random inputs for the first round messages of the NISC scheme to get* $(\mathsf{nisc}_1^{public}, \cdot)$ *and simulates the proof π and sends* $(\mathsf{Post}, (\mathsf{nisc}_1^{public}, \pi, \mathsf{info}))$ *to $\mathcal{L}$.*

We proceed with a hybrid argument. Let $\mathcal{H}_0$ denote the distribution of the view of $\mathcal{A}$ in the real world interaction.

$\mathcal{H}_1$: Let $\mathcal{H}_1$ be the same as $\mathcal{H}_0$, but instead of having the common reference string generated by the $\mathcal{F}_{CRS}^{\Pi_{NIZK}.\mathsf{ZKSetup}}$*, the common reference string is generated using* $(\mathsf{CRS}_{ZK}, \tau) \leftarrow \Pi_{NIZK}.\mathsf{ZKSetup}(1^\lambda)$ *and* $(\mathsf{CRS}_{\mathsf{NISC}}, \tau_{\mathsf{NISC}}) \leftarrow \Pi_{NISC}.\mathsf{GenCRS}(1^\lambda)$*. Note that the common reference strings are selected from exactly the same distribution, so the difference in the distribution of the view of $\mathcal{A}$ between $\mathcal{H}_0$ and $\mathcal{H}_1$ is 0.*

$\mathcal{H}_2$: Let $\mathcal{H}_2$ be the same as $\mathcal{H}_1$, but when an honest P_{LE} sends $(\mathsf{Post}, (\mathsf{nisc}_1^{public}, \pi, \mathsf{info}))$*to $\mathcal{L}$, the proof π is instead simulated with τ. Because of the perfect zero-knowledge property of Π_{NIZK}, the adversary's view in $\mathcal{H}_2$ and $\mathcal{H}_1$ is statistically close.*

$\mathcal{H}_3$: Let $\mathcal{H}_3$ be the same as $\mathcal{H}_2$, but when an honest P_{LE} sends $(\mathsf{Post}, (\mathsf{nisc}_1^{public}, \pi, \mathsf{info}))$*to $\mathcal{L}$, let* nisc_1^{public} *be generated on random input of the right length. By the security of Π_{NISC}, the difference in the distribution in the view of the $\mathcal{A}$ is negligible.*

Because the view of $\mathcal{A}$ in $\mathcal{H}_3$ is distributed the same as in the ideal world with simulator $\mathcal{S}$, the proof is done. Notice that this proof extends directly to multiple corrupt users, as the views of the users are independent, except when they send messages to each other. However, such messages do not require simulation. Thus it suffices to simulate each one independently.

P_{LE} is corrupted. *We now extend the previous proof to include corrupted P_{LE}. We extend $\mathcal{S}$ to simulate the view of P_{LE}. Note that step 5 described in $\mathcal{S}$ above is no longer applicable for corrupted users, as the notification mechanism from the actual protocol will look correct.*

1. *$\mathcal{S}$ generates the common reference string for the NIZK scheme and* $(\mathsf{CRS}_{\mathsf{NISC}}, \tau_{\mathsf{NISC}}) \leftarrow \Pi_{NISC}.\mathsf{GenCRS}(1^\lambda)$ *directly, and stores the trapdoors* $(\tau, \tau_{\mathsf{NISC}})$*. $\mathcal{S}$ runs* $(\mathsf{pk}_{sign}, \mathsf{sk}_{sign}) \leftarrow \Pi_{Sign}.\mathsf{KeyGen}(1^\lambda)$*as the judge would in the real protocol, and outputs* pk_{sign} *to the adversary $\mathcal{A}$. $\mathcal{S}$ initializes an instance of the ideal functionality in prospective mode. Finally, $\mathcal{S}$ runs* $\pi_{PRO}^{v,t,p,\theta}.\mathsf{ActivateWarrant}$ *with an empty active warrants set. $\mathcal{S}$ computes* $(\mathsf{pk}_j, \mathsf{sk}_j) \leftarrow \Pi_{Enc}.\mathsf{KeyGen}(1^\lambda)$ *for all honest P_j and sends* pk_j *to $\mathcal{A}$. When $\mathcal{S}$ detects* $(\mathsf{nisc}_1^{public}, \pi, \mathsf{info})$ *posted on $\mathcal{L}$, it verifies the proof π, and sets* ValidParameters *to false if it does not verify or* $|\mathsf{info}| \neq 0$*. $\mathcal{S}$ then initializes an empty warrant table W.*

2. *We split the case of receiving* $(\mathsf{Sent}, \mathsf{meta}, c)$ *from* $\mathcal{F}_{ARLEAS}^{v,t,p,\theta,pro}$ *intended for P_{LE} into three cases. All begin by $\mathcal{S}$ extracting the recipient P_j from* meta*:*

   (a) *If* ValidParameters *is false, $\mathcal{S}$ silently drops the message.*

(b) *If* ValidParameters *is true and P_j is not corrupted, $\mathcal{S}$ and chooses a message m_0 and computes the ciphertext $(c_1, c_2, \pi, \mathsf{meta})$ by encrypting m_0 using $\pi_{PRO}^{v,t,p,\theta}.\mathsf{SendMessage}(\mathsf{pk}_j, m_0)$.*

(c) *If* ValidParameters *is true and P_j is corrupted, $\mathcal{S}$ will also receive $(\mathsf{Sent}, \mathsf{meta}, c, m)$ from the ideal functionality, intended for P_j. $\mathcal{S}$ then encrypts m using $\pi_{PRO}^{v,t,p,\theta}.\mathsf{SendMessage}(\mathsf{pk}_j, m)$*

Finally, $\mathcal{S}$ sends the resulting ciphertext to $\mathcal{A}$.

3. *Upon receiving $(\mathsf{Sent}, \mathsf{meta}, c, 0)$ from $\mathcal{F}_{ARLEAS}^{v,t,p,\theta,\mathsf{pro}}$ intended for P_{LE}, $\mathcal{S}$ samples a random message and creates a ciphertext with $\pi_{PRO}^{v,t,p,\theta}.\mathsf{SendMessage}$. Then, it generates a false proof instead of the real proof.*

4. *Upon receiving $(\mathsf{Sent}, \mathsf{meta}, c, m)$ from $\mathcal{F}_{ARLEAS}^{v,t,p,\theta,\mathsf{pro}}$ due to an active warrant, if* ValidParameters *is true, $\mathcal{S}$ calls the send message algorithm of the real protocol on $\pi_{PRO}^{v,t,p,\theta}.\mathsf{SendMessage}(\mathsf{pk}_j, m)$ and sends the output to $\mathcal{A}$.*

5. *Upon receiving $(\mathsf{RequestWarrant}, \hat{w})$ from the adversary, intended for P_J, $\mathcal{S}$ sends $(\mathsf{RequestWarrant}, \hat{w})$ to $\mathcal{F}_{ARLEAS}^{v,t,p,\theta,\mathsf{pro}}$. If $\mathcal{F}_{ARLEAS}^{v,t,p,\theta,\mathsf{pro}}$ answers with $(\mathsf{Approve})$, $\mathcal{S}$ uses the sk_{sign} for P_J to form and sign $w = (\hat{w}, \sigma)$ as in the real protocol and adds (w, False) to W. If $\mathcal{F}_{ARLEAS}^{v,t,p,\theta,\mathsf{pro}}$ responds with $(\mathsf{Disapprove})$, $\mathcal{S}$ send $\perp$ to the adversary.*

6. *$\mathcal{S}$ monitors $\mathcal{L}$. Upon seeing a new post on the ledger, $\mathcal{S}$ performs the following steps*

   (a) *Retrieve the post by sending $(\mathsf{GetCounter})$ to $\mathcal{L}$ and receive the current counter ℓ. Then query $\mathcal{L}$ on (GetVal, ℓ) to receive the latest posting $(\ell, (\mathsf{nisc}_1^{public}, \pi, \mathsf{info}), \pi_{\mathsf{publish}})$*

   (b) *Verify $\Pi_{NIZK}.\mathsf{ZKVerify}(\mathsf{nisc}_1^{public}, \mathsf{info}, \pi) = 1$. If the proof does not verify, the $\mathcal{S}$ sets* ValidParameters *to false and halts.*

   (c) *Set* ValidParameters *to true and run* Extract *to recover $(\mathcal{W}, \mathsf{nisc}_1^{private}, r)$ from π. If extraction fails, the simulator halts with an error.*

   (d) *If there is an entry (w, False) in W for $w \in \mathcal{W}$ $\mathcal{S}$ sends $(\mathsf{ActivateWarrant}, w)$ to the ideal functionality and sets the entry to be (w, True).*

We proceed with a hybrid argument. Let $\mathcal{H}_0$ denote the distribution of the view of $\mathcal{A}$ in the real world interaction.

$\mathcal{H}_1$:Let $\mathcal{H}_1$ be the same as $\mathcal{H}_0$, but instead of having the common reference string generated by the $\mathcal{F}_{CRS}^{\Pi_{NIZK}.\mathsf{ZKSetup}}$, the common reference string is generated using $(\mathsf{CRS}_{ZK}, \tau) \leftarrow \Pi_{NIZK}.\mathsf{ZKSetup}(1^\lambda)$ and $(\mathsf{CRS}_{\mathsf{NISC}}, \tau_{\mathsf{NISC}}) \leftarrow \Pi_{NISC}.\mathsf{GenCRS}(1^\lambda)$. Note that the common reference strings are selected from exactly the same distribution, so the the distribution in the view of $\mathcal{A}$ between $\mathcal{H}_0$ and $\mathcal{H}_1$ is statistically close.

$\mathcal{H}_2$: Let $\mathcal{H}_2$ be the same as $\mathcal{H}_1$, except when $\mathcal{S}$ receives a ciphertext $(c_1, c_2, \pi, \mathsf{meta})$such that no signed warrant w has been issued such that $\theta(w, \mathsf{meta}) = 1$, the the ciphertexts consistency proof π is simulated. Due to the zero-knowledge property of Π_{NIZK}, the difference in the view of the adversary in $\mathcal{H}_2$ and $\mathcal{H}_1$ is negligible.

$\mathcal{H}_3$: Let $\mathcal{H}_3$ be the same as $\mathcal{H}_2$, except when $\mathcal{S}$ receives a ciphertext $(c_1, c_2, \pi, \mathsf{meta})$such that no signed warrant w has been issued such that $\theta(w, \mathsf{meta}) = 1$, $\mathcal{S}$ samples a message m_0 that would result in the same metadata and sets $c_1 \leftarrow (\Pi_{Enc}.\mathsf{Enc}(\mathsf{pk}_j, m_0; r_1)$. By the CCA security of Π_{Enc} the advantage of $\mathcal{A}$ in distinguishing between $\mathcal{H}_3$ and $\mathcal{H}_2$ is negligible.

$\mathcal{H}_4$: Let $\mathcal{H}_4$ be the same as $\mathcal{H}_3$, except when $\mathcal{S}$ receives a ciphertext $(c_1, c_2, \pi, \mathsf{meta})$such that no signed warrant w has been issued such that $\theta(w, \mathsf{meta}) = 1$, $\mathcal{S}$ computes c_2 as $\Pi_{NISC}.\mathsf{NISC}_2(\mathsf{CRS}_{\mathsf{NISC}}, C, (m_0, \mathsf{meta}), \mathsf{nisc}_1^{public}; r_2)$. By the security of $\Pi_{NISC}.\mathsf{NISC}_2$, $\mathcal{A}$ can only learn the correct output of C. Because there is no issued warrant $\theta(w, \mathsf{meta}) = 1$, this means the output of C is independent of m_0. Therefore, $\mathcal{H}_4$ and $\mathcal{H}_3$ are computationally indistinguishable.

$\mathcal{H}_5$: Let $\mathcal{H}_5$ be the same as $\mathcal{H}_4$,except when $\mathcal{S}$ receives a ciphertext $(c_1, c_2, \pi, \mathsf{meta})$such that no signed warrant w has been issued such that $\theta(w, \mathsf{meta}) = 1$, π is computed honestly with respect to the plaintext

message m_0. Again, by the zero-knowledge property of Π_{NIZK}, the difference in the view of the adversary in $\mathcal{H}_5$ and $\mathcal{H}_4$ is negligible.

$\mathcal{H}_6$: Let $\mathcal{H}_6$ be the same as $\mathcal{H}_5$, except when $\mathcal{A}$ detects $(\mathsf{nisc}_1^{public}, \pi, \mathsf{info})$ being posted on the ledger, $\mathcal{S}$ attempts to run the extractor $\Pi_{NIZK}.\mathsf{Extract}$ and abort the experiment if it fails. However, because the extractor only fails with negligible probability, the difference in the view of the adversary between $\mathcal{H}_6$ and $\mathcal{H}_5$ is negligible.

$\mathcal{H}_6$ has the same distribution as $\mathcal{S}$, concluding the proof. In the real world, P_{LE} would be denied warrants at the same rate as in the ideal world, as an honest P_J handles warrant requests in the same way. One note is that law enforcement can "deactivate" warrants in way not possible in the ideal functionality. However, when warrants are deactivated, honestly encrypting the message still hides the plaintext from the adversary.

P_J and P_{LE} are corrupted. *We now focus on the case when P_J and P_{LE} are both corrupted. Note that step 4 of the above simulator description is no longer relevant, as the warrant request is handled internally by $\mathcal{A}$. Our simulator requires on minor changes to steps 1 and 5, which we show below.*

1. *Do the setup as before, but waiting to receive pk_{sign} from $\mathcal{A}$*

⋮

5. *$\mathcal{S}$ monitors $\mathcal{L}$. Upon seeing a new post on the ledger, $\mathcal{S}$ performs the following steps*
   - *(a) Retrieve the post by sending $(\mathsf{GetCounter})$ to $\mathcal{L}$ and receive the current counter ℓ. Then query $\mathcal{L}$ on (GetVal, ℓ) to receive the latest posting $(\ell, (\mathsf{nisc}_1^{public}, \pi, \mathsf{info}), \pi_{\mathsf{publish}})$*
   - *(b) Verify $\Pi_{NIZK}.\mathsf{ZKVerify}(\mathsf{nisc}_1^{public}, \mathsf{info}, \pi) = 1$. If the proof does not verify, the $\mathcal{S}$ sets $\mathsf{ValidParameters}$ to false and halts.*
   - *(c) Set $\mathsf{ValidParameters}$ to true and run $\mathsf{Extract}$ to recover $(\mathcal{W}, \mathsf{nisc}_1^{private}, r)$ from π. If extraction fails, the simulator halts with an error.*
   - *(d) for each warrant $w \in \mathcal{W}$ for which there does not exist an entry (w, True) in W, the $\mathcal{S}$ executes the following steps*
     - *i. $\mathcal{S}$ sends $(\mathsf{RequestWarrant}, w)$ to $\mathcal{F}_{ARLEAS}^{v,t,p,\theta,\mathsf{pro}}$.*
     - *ii. When $\mathcal{F}_{ARLEAS}^{v,t,p,\theta,\mathsf{pro}}$ sends $(\mathsf{ApproveWarrant}, w)$ to the $\mathcal{S}$ intended to P_J, $\mathcal{S}$ responds $(\mathsf{Approve})$ on behalf of P_J*
     - *iii. $\mathcal{S}$ sends $(\mathsf{ActivateWarrant}, w)$ to the ideal functionality*
     - *iv. $\mathcal{S}$ adds (w, True) to W*

In fact, the hybrid argument above holds directly in this case as well. First notice that there are no additional messages that require simulation. By giving the adversary control of P_J, we give it the ability to create valid warrants independently. However, there is no change in behavior expected until those warrants are activated. As such, those warrants can be requested from the ideal functionality right as they are being activated.

C Proof of Theorem 3

As in the prospective case in Section 5, we show that $\pi_{\text{RET}}^{v,t,p,\theta}$ UC-realizes $\mathcal{F}_{\text{ARLEAS}}^{v,t,p,\theta,\mathsf{ret}}$ in a series of steps. First, we prove this for a single corrupt user, then extend that argument to multiple users. We then expand our analysis to consider corrupted law enforcement and corrupted judges.

P_i is corrupted. We begin with the simple case of a single corrupted user P_i controlled by an adversary $\mathcal{A}$. We construct a simulator $\mathcal{S}$ to mediate $\mathcal{A}$'s interaction with the ideal functionality as follows.

1. $\mathcal{S}$ begins by generating $(\mathsf{pk}_{\text{sign}}, \mathsf{sk}_{\text{sign}})$ as P_J would do. $\mathcal{S}$ then performs key generation for each honest party P_j. $\mathcal{S}$ then outputs all the public information to $\mathcal{A}$. Finally, $\mathcal{S}$ receives pk_i from $\mathcal{A}$.

2. When receiving a (CRS) request from $\mathcal{A}$, $\mathcal{S}$ generates $(\mathsf{CRS}_{ZK}, \tau) \leftarrow \Pi_{\text{NIZK}}.\mathsf{ZKSetup}(1^\lambda)$ and returns CRS_{ZK} to $\mathcal{A}$ and keeps τ.

3. Whenever $\mathcal{S}$ receives (send, c) from P_i intended for P_j, $\mathcal{S}$ parses $c = (c_1, c_2, c_3, \pi, \mathsf{meta})$ and verifies π. If it does not verify, set $b \leftarrow 1$, and set $b \leftarrow 0$ otherwise. Next, $(r, r_1, r_2) \leftarrow \Pi_{\text{NIZK}}.\mathsf{Extract}(\mathsf{CRS}_{ZK}, \tau, x, \pi)$, queries the random oracle $(\mathsf{HashConfirm}, r_3) \leftarrow \mathcal{G}_{\mathsf{pRO}}(\mathsf{HashQuery}, (\text{“RP”} \| r))$. It then computes $m \leftarrow c_3 \oplus r_3$ and verifies the structure of c_1, c_2 by re-computing the ciphertexts c_1, c_2 as in the real protocol. If some passes do not check, set $b \leftarrow 0$. Finally, it sends $(\mathsf{SendMessage}, \mathsf{sid}, P_j, m, b)$ to the ideal functionality.

4. When $\mathcal{S}$ receives $(\mathsf{Sent}, \mathsf{sid}, \mathsf{meta}, c, m)$ from the ideal functionality destined for P_i, it identifies the public key for P_i and computes the ciphertext $c = (c_1, c_2, c_3, \pi, \mathsf{meta})$ as in the real protocol. It sends the resulting ciphertext to P_i.

5. When $\mathcal{S}$ receives $(\mathsf{NotifyWarrant}, t(w))$ from the ideal functionality, it simulates the proof π, and sends $(\mathsf{Post}, (t(w), \pi))$ to $\mathcal{L}$.

We prove that $\mathcal{A}$'s interaction with the real protocol and the ideal functionality, mediated by $\mathcal{S}$ are computationally indistinguishable by using the following hybrids.

Let $\mathcal{H}_0$ denote the distribution of the view of $\mathcal{A}$ in the real world interaction.

$\mathcal{H}_\mathbf{1}$: Let $\mathcal{H}_1$ be the same as $\mathcal{H}_0$, but instead of having the common reference string generated by the $\mathcal{F}_{CRS}^{\Pi_{\text{NIZK}}.\mathsf{ZKSetup}}$, the common reference string is generated using $(\mathsf{crs}, \tau) \leftarrow \Pi_{\text{NIZK}}.\mathsf{ZKSetup}(1^\lambda)$. Note that the common reference string is selected from exactly the same distribution, therefore, the distribution of the view of $\mathcal{A}$ between $\mathcal{H}_0$ and $\mathcal{H}_1$ is the same.

$\mathcal{H}_\mathbf{2}$: In this hybrid we will use the extractor $\Pi_{\text{NIZK}}.\mathsf{Extract}$ to get the randomness and create the ciphertext as in step 3. $\mathcal{H}_1$ and $\mathcal{H}_2$ are computationally indistinguishable as $\mathsf{Extract}$ will only fail with negligible probability.

$\mathcal{H}_\mathbf{3}$: At this point we will simulate the proof π when publishing on the ledger. $\mathcal{H}_2$ and $\mathcal{H}_3$ are indistinguishable because of the zero knowledge property of Π_{NIZK}.

The view of the adversary in $\mathcal{H}_3$ is the same as its view when talking with $\mathcal{S}$ in the ideal world, which concludes the hybrid argument.

This argument extends to multiple parties as all ciphertexts can be properly simulated by either extracting from the NIZK or actually receiving the message in case the receiver is corrupted.

Subset of users and P_{LE} are corrupted. We now extend $\mathcal{S}$ from above to handle a corrupt P_{LE}. Note that step 5 of the above description is not longer relevant, as notifications will come directly from the ledger for the corrupt parties.

1. $\mathcal{S}$ runs the same setup as above. Additionally, $\mathcal{S}$ initializes an empty message equivocation table T.

2. When receiving a (CRS) request from $\mathcal{A}$, $\mathcal{S}$ generates $(\mathsf{CRS}_{ZK}, \tau) \leftarrow \Pi_{\text{NIZK}}.\mathsf{ZKSetup}(1^\lambda)$ and returns CRS_{ZK} to $\mathcal{A}$ and keeps τ.

3. We split the case of receiving $(\mathsf{Sent}, \mathsf{sid}, \mathsf{meta})$ from $\mathcal{F}_{\text{ARLEAS}}^{v,t,p,\theta,\mathsf{ret}}$ intended for P_{LE} into two cases:

   (a) If P_j is not corrupted, $\mathcal{S}$ samples $r, r_1, r_2, r_3 \leftarrow \{0,1\}^{\mathsf{poly}(\lambda)}$ (where $\mathsf{poly}(\cdot)$ is a polynomial function that upper-bounds the longest random string necessary) and computes the ciphertext components as follows

   - $c_1 \leftarrow \Pi_{\text{Enc}}.\mathsf{Enc}(pk_j, r; r_1)$
   - $c_2 \leftarrow \Pi_{\text{EWE}}.\mathsf{Enc}(\mathsf{meta}, r; r_2)$
   - $c_3 \leftarrow r_3$
   - Use $\Pi_{\text{NIZK}}.\mathsf{ZKProve}$ to compute

$$\pi \leftarrow \mathit{NIZK}\left\{(r, r_1, r_2) \;\middle|\; \begin{array}{c} c_1 = \Pi_{\text{Enc}}.\mathsf{Enc}(\mathsf{pk}, r; r_1) \wedge \\ c_2 = \Pi_{\text{EWE}}.\mathsf{Enc}(\mathsf{meta}, r; r_2) \end{array}\right\}$$

$\mathcal{S}$ adds an entry $(r, r_1, r_2, r_3, c_1, c_2, c_3, \pi, \mathsf{meta})$ to T.

(b) If P_i or P_j is corrupted, $\mathcal{S}$ will also receive $(\mathsf{Sent}, \mathsf{sid}, \mathsf{meta}, c, m)$ from the ideal functionality, intended for P_j. $\mathcal{S}$ then encrypts m using $\pi_{\text{RET}}^{v,t,p,\theta}.\mathsf{SendMessage}$, programming the random oracle honestly as needed.

$\mathcal{S}$ then sends the resulting ciphertext to both the ideal functionality and $\mathcal{A}$.

4. Upon receiving $(\mathsf{Sent}, \mathsf{meta}, c, 0)$ from $\mathcal{F}_{\text{ARLEAS}}^{v,t,p,\theta,\mathsf{ret}}$ intended for P_{LE}, $\mathcal{S}$ samples a random message and creates a ciphertext with $\pi_{\text{RET}}^{v,t,p,\theta}.\mathsf{SendMessage}$. Then, it generates a false proof instead of the real proof.

5. When $\mathcal{S}$ receives $(\mathsf{RequestWarrant}, \hat{w})$ from $\mathcal{A}$, intended for P_J, $\mathcal{S}$ sends $(\mathsf{RequestWarrant}, \hat{w})$ to the trusted party. If the trusted party responds with $\perp$, $\mathcal{S}$ sends $(\mathsf{Disapprove})$ to $\mathcal{A}$. Otherwise, $\mathcal{S}$ generates a warrant $\hat{w}$ and signs it with $\sigma = \Pi_{\text{Sign}}.\mathsf{Sign}(\mathsf{sk}_{\text{sign}}, \hat{w})$ and sends $w = (\hat{w}, \sigma)$ to $\mathcal{A}$.

6. When $\mathcal{S}$ notices new posts on $\mathcal{L}$ it retrieves that information by sending $t \leftarrow (\mathsf{GetCounter})$ and $(t(w), \pi)) \leftarrow (\mathsf{GetVal}, t)$ to $\mathcal{L}$. $\mathcal{S}$ uses τ to extract w from π. Without loss of generality, let w be for user P_i. $\mathcal{S}$ sends $(\mathsf{ActivateWarrant}, w, P_j)$ to the ideal functionality. Next, $\mathcal{S}$ checks to see if there is an entry $(r, r_1, r_2, r_3, c_1, c_2, c_3, \pi, \mathsf{meta})$ for which $\theta(\hat{w}, \mathsf{meta}) = 1$ in T and sends $(\mathsf{AccessData}, (c_1, c_2, c_3, \pi), w)$ to the ideal functionality for all such records. If the ideal functionality responds with $\perp$, abort. Otherwise, the ideal functionality will return a message m. $\mathcal{S}$ then programs the random oracle by sending $(\mathsf{ProgramRO}, r, (r_3 \oplus m))$, $(\mathsf{ProgramRO}, (\text{“WE”} r \| m), r_2)$ and $(\mathsf{ProgramRO}, (\text{“ENC”} \| r \| m), r_1)$, and responds to the initial query.

To show that the simulation above is computationally indistinguishable from the real experiment in the view of $\mathcal{A}$, we proceed with a hybrid argument. Let $\mathcal{H}_0$ denote the distribution of the view of $\mathcal{A}$ in the real world interaction.

$\mathcal{H}_1$: Let $\mathcal{H}_1$ be the same as $\mathcal{H}_0$, but instead of having the common reference string generated by the $\mathcal{F}_{CRS}^{\Pi_{\text{NIZK}}.\mathsf{ZKSetup}}$, the common reference string is generated using $(\mathsf{crs}, \tau) \leftarrow \Pi_{\text{NIZK}}.\mathsf{ZKSetup}(1^\lambda)$. Note that the common reference string is selected from exactly the same distribution, therefore, the distribution of the view of $\mathcal{A}$ between $\mathcal{H}_0$ and $\mathcal{H}_1$ is the same.

$\mathcal{H}_2$: In this hybrid we change the NIZK proof for L_{NIZK}^1 to be a simulation. Because of the zero knowledge property of Π_{NIZK}, $\mathcal{H}_2$ is statistically close to $\mathcal{H}_3$.

$\mathcal{H}_3$: Choose r_1, r_2, and r_3 uniformly at random in $\{0,1\}^\lambda$ to replace the randomness used to compute c_1, c_2, and c_3 respectively. Also, send $(\mathsf{ProgramRO}, (\text{“ENC”} \| r \| m), r_1)$, $(\mathsf{ProgramRO}, (\text{“WE”} \| r \| m), r_2)$, and $(\mathsf{ProgramRO}, r, r_3)$ to $\mathcal{G}_{\mathsf{pRO}}$. Clearly the only way the view between $\mathcal{H}_1$ and $\mathcal{H}_2$ can differ is when $\mathcal{G}_{\mathsf{pRO}}$ was queried on these inputs before $\mathcal{S}$ programs them, in which case the protocol aborts, but this only happens with negligible probability which we prove in Lemma 1.

$\mathcal{H}_4$: Change the simulated NIZK proof back to a real proof. Again, by the zero-knowledge property $\mathcal{H}_3$ and $\mathcal{H}_4$ are indistinguishable.

$\mathcal{H}_5$: Now, do everything as described in step 5 to recover the message m. Next, we change the ciphertext c_3 in step 3 to be r_3 and change the programming of $\mathcal{G}_{\mathsf{pRO}}$ to be $(\mathsf{ProgramRO}, r, m \oplus r_3)$ in step 5. Note that by security of the one-time pad $\mathcal{H}_4$ and $\mathcal{H}_5$ are indistinguishable.

Now, the view of the adversary in $\mathcal{H}_5$ is exactly the same as its view when talking with $\mathcal{S}$ in the ideal world, which concludes the hybrid argument.

Lemma 1 *For any adversary $\mathcal{A}$ against $\mathcal{H}_4$ in the security proof of $\pi_{RET}^{v,t,p,\theta}$ where P_{LE} is compromised, the probability that $\mathcal{A}$ queries or programs $\mathcal{G}_{\mathsf{pRO}}$ on r, “ENC”$\|r\|m$, or “WE”$\|r\|m$ without sending $(\mathsf{Post}, (t(w), \pi))$ to $\mathcal{L}$ is negligible, for $r \leftarrow\$ \{0,1\}^\lambda$ and m uniformly at random from the message space, both used inside $\mathcal{H}_4$.*

Proof 4 *Assume such adversary $\mathcal{A}$ exists, then if $\mathcal{A}$ has queried or programmed $\mathcal{G}_{\mathsf{pRO}}$ on one of the inputs it must have had r. Either $\mathcal{A}$ has selected r by accident which can only happen with probability $2^{-\lambda}$, or it has extracted it from the ciphertext, which we will now show with a series of hybrids can only happen with negligible probability.*

$\mathcal{H}_0'$: This looks exactly the same as $\mathcal{H}_4$. The encryption is created in the following way:

- *sample* $r \leftarrow \{0,1\}^\lambda$
- $(\mathsf{HashConfirm}, r_1) \leftarrow \mathcal{G}_{\mathsf{pRO}}(\mathsf{HashQuery}, (\textit{"ENC"} \| r \| m))$,
- $(\mathsf{HashConfirm}, r_2) \leftarrow \mathcal{G}_{\mathsf{pRO}}(\mathsf{HashQuery}, (\textit{"WE"} \| r \| m))$, *and*
- $(\mathsf{HashConfirm}, r_3) \leftarrow \mathcal{G}_{\mathsf{pRO}}(\mathsf{HashQuery}, (\textit{"RP"} \| r))$
- *Create the components or the encryption in the following way:*
  - $c_1 \leftarrow \Pi_{Enc}.\mathsf{Enc}(pk_j, r; r_1)$
  - $c_2 \leftarrow \Pi_{EWE}.\mathsf{Enc}(\mathsf{meta}, r; r_2)$
  - $c_3 \leftarrow m \oplus r_3$
  - *Use* $\Pi_{NIZK}.\mathsf{ZKSimulate}$ *to simulate the proof*

$\mathcal{H}'_\mathbf{1}$: *Replace all three calls to* $\mathcal{G}_{\mathsf{pRO}}$ *with uniform random values* r_1, r_2, r_3. *By the properties of* $\mathcal{G}_{\mathsf{pRO}}$ *the view of* $\mathcal{A}$ *doesn't change between* $\mathcal{H}'_0$ *and* $\mathcal{H}'_1$. *Note that we only care about the view of* $\mathcal{A}$ *up until it calls* $\mathcal{G}_{\mathsf{pRO}}$ *on one of the three values, so we don't have to program* $\mathcal{G}_{\mathsf{pRO}}$ *accordingly.*

$\mathcal{H}'_\mathbf{2}$: *Now we replace* c_1 *by* $\Pi_{Enc}.\mathsf{Enc}(pk_j, r'; r_1)$, *which is possible because the proof is simulated. By the security of* Π_{Enc} *the view of the adversary doesn't change.*

$\mathcal{H}'_\mathbf{3}$: *Next, we replace* c_2 *with* $\Pi_{EWE}.\mathsf{Enc}(\mathsf{meta}, r'; r_2)$. *If there would be a distinguisher for* $\mathcal{H}'_2$ *and* $\mathcal{H}'_3$ *we can build a distinguisher for* Π_{EWE}, *by the extractable security of* Π_{EWE} *we can now extract a witness, but this is in contradiction with the ideal functionality* $\mathcal{G}_{\mathsf{ledger}}$, *as we assumed it was not called. Therefore,* $\mathcal{H}'_2$ *and* $\mathcal{H}'_3$ *must be indistinguishable.*

We see that $\mathcal{H}'_3$ *does not contain any reference to* r, *and* $\mathcal{A}$ *would not have changed its strategy as the difference between its view in* $\mathcal{H}'_0$ *and* $\mathcal{H}'_3$ *is computationally indistinguishable.*

Subset of users, P_J**, and** $P_{\mathbf{LE}}$ **are corrupted.** We further extend our analysis to cover both corrupt law enforcement and corrupt judges. Notice that step 4 of the previous simulator description is no longer relevant, as these requests are handled inside $\mathcal{A}$. The proof is exactly the same is in the previous case.

D An ARLEAS' Parameterizing Functions in Practice

In the name of being generalizable, we have presented both our definition and protocols with a significant number of parameterized functions, making our construction very abstract. To better understand how an ARELAS might actually work in practice, we give possible choices for various parts of the system.

D.1 Service Providers

For simplicity, in the protocols and ideal functionalities we present in this work, there is no service provider. Users send ciphertexts *directly to law enforcement*. In practice, law enforcement does not directly operate a communication network, and thus a service provider is crucial. We do not give a formal treatment of the responsibilities of the service provider, as it is not central to our work, but assume that the service provider's role is similar to modern systems. This includes authenticating users, delivering messages, and verifying that messages are properly constructed. Moreover, we assume that the service provider is transparent to law enforcement, *i.e.* all requests for data will be approved.

D.2 Transparency Functionalities

Recall that when law enforcement wants to activate an issued warrant, an ARLEAS requires that they make some information, determined by the transparency function, about that warrant public. In practice, choosing a transparency function is a balance between robust accountability and operability (*i.e.* not tipping off individuals that they are under surveillance).

We briefly consider three possible transparency functions:

- **Counting Warrants.** A baseline transparency function would leak the *number* of warrants activated over time. Each time law enforcement activates a group of warrants, the transparency function counts these valid warrants and outputs this count. While the impact of this simple transparency function may seem limited, it provides a way for the government to detect key exfiltration. This can be done by observing when the number of warrants activated is greater than those known about.

- **Issuing Court.** Another possible transparency function would leak the identity of the court that issued the warrant. While this could potentially leak more information about *where* an activated warrant is going to be used (*i.e.* if it was issued by a local court), it also provides significantly stronger protections and oversight. For instance, if the warrant signing keys of a particular court are compromised, a system with this transparency will quickly identify the problem and be able to re-key just that court.

- **Differential Private Analytics.** The above transparency functions do not leak any information about the *contents* of the warrant. As our final example transparency function, we consider a function that leaks differentially private [DMNS06,Dwo08] information about activated warrants. The analytics could, for instance, include the racial identity of targeted individuals, so that civil liberties groups could monitor courts with a problematic history. Computing these differential private analytics could either be done with randomized response or by modifying the interface to the transparency function to take in *all* active warrants. Either way, the random coins used must be deterministically generated from the warrants, as a possibly adversarial law enforcement would be able to choose them otherwise. Additionally, special attention must be paid to the privacy budget if iterative analytics will be run over the same warrants.

D.3 Policy Functionalities

While the transparency function allows for *detection* of malicious behavior, the policy function *prevents* certain types of warrants altogether. As part of activating a warrant, law enforcement must prove (in zero-knowledge) that all the warrants being activated satisfy the policy function. Thus it will be impossible to *activate* warrants that are not compliant with the policy function, even if a malicious judge does issue such warrants.

We consider three possible choices of policy functions, which limit the space of possible valid warrants:

- **Warrants Must Specify Individual Targets.** In order to limit the risk of unfettered surveillance, it may be prudent to require that warrants specify individuals, rather than an entire group. Note that this does not inherently limit the power of law enforcement, as a group can be monitored by simply issuing individual warrants for each member. However, it could prevent *unintentionally* issuing a warrant with an overly broad scope. To mitigate this risk, the system can be set up with a policy function that checks the structure of each warrant and ensures that it specifies only a single individual.

- **Warrants Must Have A Limited Time Scope.** It may be desirable to require all warrants to specify the length of time for which they are valid. Put another way, warrants should only apply to messages encrypted within a fixed time-frame. This could prevent unintended mission creep or reuse of old, stale warrants. As before, this does not affect the power of law enforcement (renewed warrants can be issued at will) but it does reduce the possibility for unintentionally using warrants after a case has been concluded.

- **Warrants Issued By Problematic Courts Must Be Subject To Additional Oversight.** Many law enforcement organizations in the United States have a problematic history of violating civil rights. In these cases, it is common to have Federal bodies oversee the actions of these law enforcement organizations. This oversight continues until the federal government is convinced that it is no longer warranted. This arrangement can be formalized by requiring warrants to bear the signature of the federal oversight body. Alternatively this role could be outsourced to a civil liberty group instead of a federal body.

D.4 Metadata and Warrant Scope Check Functionalities

Each message sent in an ARLEAS has associated metadata that is added by the sender. This metadata is information about the message that is *public* to the service provider and law enforcement. In modern systems, some metadata may be added by the sender while other metadata may be added by the service provider relaying the message. We explicitly consider the following three kinds of metadata, but note that the space of potential metadata information could be significantly larger:

- **Sender/Receiver Identity.** A service provider is responsible for routing messages from a sender to a receiver. As such, it is important the service provider is able to know these identities so that it can fulfill this functionality. Warrants that targets specific users can use this information to determine if a message should be decryptable.

- **Timestamp.** Timestamp information is a common piece of information included with a message in modern systems. As mentioned above, including timestamp information in message metadata and in warrant scopes is an important way of ensuring that a warrant is only used to surveil the intended targets and not used surreptitiously after an investigation has finished.

- **Geolocation.** Geolocation information gives a message a physical origin and can be provided either by global positioning systems or by identifying the first piece of static networking equipment through which the message is transmitted (*e.g.* a cell tower). This information could be used by law enforcement to target groups moving through a specific location. For instance, law enforcement may have a strong justification that illicit activity is happening in a remote area, but are unable to identify *who* is visiting the area. Allowing warrants to reason over geolocation is a powerful investigative tool, but should be considered carefully as to not accidentally enable dragnet surveillance.

While we are not *overly* interested in designing a system secure against malicious senders (see Section 1.3), it would be preferable for a service provider to be able to drop messages that are flagrantly flaunting the system using the metadata verification function $v(\cdot, \cdot)$. This function verified if metadata associated with a particular message is correctly. There are two main approaches for a metadata verification functionality. The first is a "common sense" approach: the service provider checks that the metadata supplied by the user is reasonable from the view of the service provider. A second approach would be to get *authenticated* metadata sources, where appropriate. For instance, existing GPS satellite messages could be modified to transmit a *cryptographically signed* version of their signal, and messages could include this in the metadata. Alternatively, a mobile device get an attestation from all connected cell towers proving their location. Neither approach is foolproof — colluding devices in different geolocations might be able to spoof a location — but they could raise the difficulty of circumventing the system.

Warrant Scope Check. In addition to being input to the metadata verification functionality, the metadata also is an input to the *warrant scope check* $\theta(\cdot, \cdot)$. This functionalities take in a warrant and some metadata, and decides if the associated message is within the scope of the supplied warrant. We do not specify how this check would work in practice, as it is tightly coupled to the format of warrants and metadata. A minimal implementation might be to split the warrant into a list of clauses or requirements and ensure the metadata satisfies each one.