Finding Safety in Numbers with Secure Allegation Escrows

Venkat Arun*, Aniket Kate†, Deepak Garg‡, Peter Druschel‡ and Bobby Bhattacharjee§

*Massachusetts Institute of Technology
venkatar@mit.edu

†Purdue University
aniket@purdue.edu

‡Max Planck Institute for Software Systems
{dg, druschel}@mpi-sws.org

§University of Maryland
bobby@cs.umd.edu

Abstract—**For fear of retribution, the victim or witness of a crime may be willing to report it only if other victims of the same perpetrator also step forward. Examples include 1) identifying oneself as the victim of sexual harassment, especially by a person in a position of authority or 2) accusing an influential politician, an authoritarian government, or one's own employer of corruption. To handle such situations, legal literature has proposed the concept of an *allegation escrow*: a neutral third-party that collects allegations, matches them against each other, and discloses them only after reveal thresholds (in terms of number of co-allegers), pre-specified by the allegers, are reached. Until then, allegations and allegers' identities are kept confidential.**

An allegation escrow can be realized as a single trusted third party; however, this party must then be trusted to keep the identity of the alleger and content of the allegation private despite any threats or coercion from perpetrators. To address this problem, this paper introduces Secure Allegation Escrows (SAE, pronounced "say"). A SAE is a group of parties with independent interests and motives, acting *jointly* as an allegation escrow. By design, SAEs provide a very strong property: No less than a majority of parties constituting a SAE can de-anonymize or disclose the content of an allegation without a sufficient number of matching allegations (even in collusion with any number of other allegers). Once a sufficient number of matching allegations exist, the joint escrow discloses the allegation and the allegers' identities. We describe how SAEs can be securely constructed using a distributed authentication protocol and a novel allegation matching and bucketing algorithm and evaluate a prototype implementation, demonstrating feasibility in practice.

I. Introduction

In many cases, the victim or the witness of a crime may be too afraid to accuse the perpetrator for fear of retribution by the perpetrator. In other cases, in particular those involving sexual harassment, the survivor may not report the crime anticipating negative social consequences or further harassment by the perpetrator. In such situations, the victim (or the witness) may find it easier to act against the perpetrator if others also accuse the same perpetrator of similar crimes. Examples of this abound, a notable one being the recent #MeToo movement [1], which led to many public allegations of sexual abuse in the US film industry and elsewhere, all triggered by the courage of an initial few.

An *allegation escrow* aids such collective allegations by matching allegations against a common perpetrator confidentially. Technically, an allegation escrow allows a victim or witness of a crime to file a confidential allegation, which is to be released to the designated recipient(s) once a pre-defined number of matching allegations against the same party have been filed. The identities of the accusers and the accused, as well as the content of the allegation, remain confidential until the release condition holds. The designated recipient(s) can be the allegers themselves or a concerned authority.

Besides helping fearful victims to report crimes (safe in the knowledge that their allegation will be revealed only as part of a larger group), allegation escrows help improve reporting in cases where the victim is uncertain if the perpetrator's actions constitute a crime. Escrowed allegations also enjoy higher credibility since, to all appearances, they are filed independently of each other (as opposed to public allegations, where the credibility of subsequent allegations may be questioned). It has been argued that allegation escrows mitigate the *first-mover disadvantage* that perpetrators typically benefit from [4].

Project Callisto [3] is an allegation escrow system that has been deployed in 13 universities with over 100k students, to help report sexual assault on university campuses. A victim can instruct the system to release the allegation only when another allegation against the same person exists. Sexual assault survivors who visit the Callisto website of their university are 5 times more likely to report the crime than those who do not, and Callisto has reduced the average time taken by a student to report an assault from 11 to 4 months [2]. This makes a very strong case for the usefulness of allegation escrows.

However, existing allegation escrows such as Project Callisto are implemented as a *single* trusted third party, similar to ombuds-offices in many organizations. Although technically simple and effective in many cases, the use of a single party may raise concerns about the escrow's trustworthiness, impartiality and fallibility to influential perpetrators, thus driving away potential users. In the case of a university or corporate escrow (e.g. an ombuds-office), students or employees may be unsure that an allegation against a high-ranking official would be treated with integrity. A commercial escrow may raise concerns about its independence from funding sources and long-term security, just as a government-run escrow may raise concerns about its independence from high-ups in law enforcement and the judiciary. In all these cases, users may not trust the escrow enough to file allegations against people they deem to have the power to coerce

Network and Distributed Systems Security (NDSS) Symposium 2020
23-26 February 2020, San Diego, CA, USA
ISBN 1-891562-61-4
https://dx.doi.org/10.14722/ndss.2020.24331
www.ndss-symposium.org

or compromise the escrow. When they do file allegations, strong perpetrators may abuse their power to prematurely discover escrowed allegations, suppress and alter the allegations, or even seek retribution against the victims. Finally, even if one victim trusts an escrow, other victims of the same perpetrator may not, making it impossible for the escrow to match their allegations. This suggests the need for allegation escrows based on *several independent parties*, none of which in itself is a single point of coercion or attack by strong adversaries. In this paper, we present a cryptographic design of such escrows.

A. Contributions

Our escrows, called SAEs (short for Secure Allegation Escrows, prounounced *says*), distribute client secrets—confidential allegations and identities of the alleger and accused—among several parties by threshold secret-sharing [38]. These parties, called escrows, act together and perform multi-party computations (MPCs) to provide the same functionality as a single-party allegation escrow, but compromising less than half of the escrows provides no information about escrowed allegations, accusers or the accused. The escrows can span diverse administrative, political and geographic domains, mitigating the chances of simultaneous attacks over a majority by the same adversary. To enable SAE, we make three key technical contributions.

First, SAE needs to provide a strong accountability property: Every filed allegation can be linked to a real-world (strong) identity, which is revealed to the concerned authority once the allegation has found enough matches. This discourages fake allegations and probing attacks that all allegation escrows are susceptible to (see §II-A). Providing accountability and privacy simultaneously requires a nontrivial authentication protocol (see §IV-C). When filing an allegation, no minority of escrows learn the identity of the filing user. But when the allegation is to be revealed, a majority can determine the identity.

Second, we need to efficiently match allegations to each other, even when each escrow only has shares of the allegations, *while* providing accountability. For this, we use a novel construction of distributed (verifiable) pseudorandom functions (DVRFs) over shared secrets. This is necessary, since traditional Oblivious PRFs will not provide accountability (see §I-B).

Third, SAE allows each alleger to decide their *reveal threshold*—how many matching allegations must be available before their allegation may be revealed. A set of matching allegations, A, is revealed if and only if all their thresholds are $\leq |A|$. For instance, if three matching allegations with thresholds $\{2,3,5\}$ are filed, no allegation should be revealed, since 5's threshold isn't met. But if another allegation with threshold 3 is filed, then the ones with thresholds $\{2,3,3\}$ (but not 5) should be revealed. We design a novel bucketing algorithm to support reveal thresholds > 2 efficiently (see §IV-D2).

This flexibility is important since one size doesn't fit all. In many cases of sexual misconduct, a small threshold is desired to maximize the probability of a match; indeed Project Callisto which always uses a threshold of two—where an allegation is revealed if another matching one is filed—has demonstrated real-world utility. However, when the perpetrator is powerful, such as an influential politician, allegers may need many more corroborators to get justice while avoiding adverse consequences for themselves. Similarly, when accusing one's employer (or government) of misconduct or corruption, a person risks getting fired or persecuted. Having just 1 or 2 corroborators may not be much better than being alone. A much higher threshold could be more appropriate. SAE's flexibility allows each alleger to tailor the reveal threshold to the semantics of their allegation and their assessment of risk and comfort. The computational complexity doesn't increase with the reveal threshold (see §IV-E).

We formally prove the end-to-end security of our SAE cryptographic design in the universal composability (UC) framework [10]. Specifically, we present an ideal functionality which, by definition, captures the expected security and accountability properties of a SAE, and then show that our cryptographic design *realizes* this functionality. We also implement a prototype of SAE to understand the latency and throughput of the system. We find that our design is efficient enough for typical use-conditions of allegation escrows.

To summarize, the contributions of our work are: 1) The concept of SAE, a distributed allegation escrow, that is robust to compromise or coercion of minority subsets of constituting parties. 2) A cryptographic realization of SAEs using verifiable secret sharing (VSS) and efficient multi-party computation (MPC) protocols. In particular, a new protocol for user authentication and a new matching and bucketing protocol. 3) A formal security analysis of our cryptographic realization. 4) A prototype implementation and empirical evidence of reasonable performance in practice.

B. Related Work

Ayres and Unkovic [4] discuss the legal and social utility of allegation escrows in encouraging reporting of sexual misconduct. As discussed, Project Callisto [3] is a real deployment that uses a single trusted-party escrow for allegations of sexual misconduct, and has demonstrated the utility of such a system in university settings.

WhoToo [32] is a concurrent work that proposes a secure allegation escrow for allegations of sexual misconduct. Like SAE, it distributes trust among multiple parties using MPC. SAE differs from WhoToo in a two key ways. First, WhoToo forces all allegations to use a global pre-determined reveal threshold. As discussed above, this inflexibility limits its scope of application. To our knowledge, SAE is the first system to allow each allegation to have its own reveal threshold.

Second, if there are N allegations already in the system, to file the $(N+1)^{th}$ allegation, WhoToo needs to perform $O(N)$ cryptographic operations, including $O(N)$ multi-party computations. SAE is more scalable. Its running time is $O(1)$, independent of the number of pre-existing allegations. We compare the compute complexity in detail in section IV-E. To obtain such efficiency, SAE sometimes reveals which allegations match which others, before their thresholds are met. Nevertheless, an adversary is unlikely to be able to exploit this (see §IV-D2)

Project Callisto has also developed a prototype cryptographic solution to distribute the trust assumptions [36, 37]. It uses a (potentially distributed) Oblivious PRF (OPRF) server. Allegers query the server to learn the (deterministic) PRF of the accused's identity, while the server just learns the alleger's identity (not the accused's). The alleger uploads the PRF to a database server, which compares them, in clear-text, to match allegations. Callisto's security analysis is informal and has a weaker threat model that admits two attacks.

First, the OPRF server learns the alleger's identity. If a perpetrator compromises this server and learns that one of their victims filed an allegation soon after the crime, they may be able to deduce the content of the allegation. In contrast, in SAE, no minority set of escrows learn the identity of any alleger until enough matches are found.

Second, it doesn't hold allegers accountable, which allows the adversary to probe how many allegations exist against a given person. They may then guess who filed the allegation from context. To mount the attack, they query the OPRF server to learn the PRF of that person's identity, and compromise the database server to learn how many previously filed allegations match this PRF. SAE prevents this attack by ensuring that if a PRF of an accused's identity is computed, the identity of the alleger is irrevocably tied to the allegation. This enforces accountability and disincentivizes such attacks (see §II-A).

Trusted Hardware. In recent work, Harnik et al. [26] use a hardware-backed secure enclave (built on Intel SGX) to isolate a fully autonomous, single-party allegation escrow. The ideas can be combined with SAE to obtain a threat model stronger than either: SAE's escrows can be hosted in SGX enclaves to provide a second line of defense even when the administrators of a *majority* of escrows are acting maliciously.

Generic MPC and Covert Computation. Generic black-box MPC can also be used to solve the problem. However, like WhoToo, it too incurs at-least $O(N)$ cost per allegation, and doesn't scale. Like covert computation [11, 40], SAE hides even the participation of a user in the protocol, revealing the result only if a pre-defined condition is met. However, black-box covert protocols don't scale well to large numbers of users, and require users to be online for matching to occur.

II. SAE Design

A. Requirements for Secure Allegation Escrow

A secure allegation escrow system should provide the following security and privacy properties:

- **Allegation secrecy**. The escrow should hold each allegation secret until enough matches are found. An allegation should be released only as part of a group of matching allegations.
- **Alleger anonymity**. Similarly, the escrow should hold each alleger's identity secret until enough matches are found.
- **Scalability**. The escrow should scale to many allegers and allegations. In section §IV-F we discuss why scalability 1) helps avoid crippling DoS attacks, 2) increases probability of a correct match and, 3) enhances privacy by preventing timing side-channel attacks.
- **Accountability**. Each allegation is bound to a strong real-world identity. Once a match is found, the real identities of the matched allegers are revealed to the designated authority. Accountability discourages fake/bogus allegations, and acknowledges that the primary source of authenticity of an allegation, escrowed or otherwise, is the human backing it.

All allegation escrows (not just SAEs) are fundamentally vulnerable to probing attacks where the adversary files fake probe allegations in the hope of revealing other genuine allegations against the same accused before sufficiently many genuine matching allegations have been filed. For example, the adversary may be a guilty perpetrator seeking vengeance or a journalist seeking a story. While the ultimate defense against such attacks lies in preventing this kind of abuse by non-technical means (e.g., by criminalizing probe allegations), SAEs aid such defenses through the property of accountability, which ensures that the real-world identities of all allegers, including fake allegers, are revealed to the designated authority after a match. For this to work, we assume the adversary is afraid of law and/or public perception.

Accountability doesn't deter an adversary who knows their allegation (and hence identity) will never be revealed, perhaps because their reveal threshold is too high or their allegation is unlikely to match any others. We ensure a probe is useful for discovering the presence of only those allegations, that would be revealed at the same time as the probe itself (see §IV-D2). Hence, the probe is just as likely to be revealed as the victim allegation.

Additionally, allegation escrows are most useful in asymmetric situations, where individual allegers are at a disadvantage compared to the accused. Allegation escrows enable the allegers to build "strength in numbers" without fear of premature retaliation. However, the very information held by allegation escrows motivate powerful attacks against them, since the accused can gain by learning about allegers before a large enough group has formed. Thus, allegation escrows should *expect* to be targeted. This leads to the following *meta-property* that spans the previous properties.

- **Robustness**. The escrow should resist coercion and compromisation attacks. It should continue to provide the properties above even if some constituent parts are compromised or willingly cooperate with the adversary.

B. Threat model and assumptions

A SAE adversary is interested in prematurely learning the identities of one or more allegers or discovering unrevealed allegations. For instance, the adversary may be a guilty perpetrator interested in determining whether there is any allegation against them. Or, they may be journalists/trolls looking for a story against a famous person. To this end, an adversary may *actively* compromise some escrows into revealing information they hold and/or not following the SAE protocol correctly. By design, SAEs are robust to such attacks on up to half the escrows simultaneously: allegation secrecy, alleger anonymity, scalability and accountability hold even if the adversary learns all cryptographic and allegation-related material possessed by up to half the escrows, and causes them to behave arbitrarily. Any individual escrow can halt the protocol and affect liveness by not cooperating. We disincentivize this by ensuring that honest escrows can determine the identity of any escrow who is not cooperating.

We make the standard assumption that the adversary cannot break cryptography. Technically, the adversary is a probabilistic polynomial time (PPT) algorithm with respect to a chosen security parameter λ. We assume, as usual, that uncompromised parties (escrows and allegers) keep their long-term secrets safe.

For alleger anonymity, we assume that allegers do not reveal any information beyond that explicitly mentioned in our protocols. For example, they should hide their IP addresses using standard network anonymity solutions like Tor [18]. To ensure the time of allegation filing doesn't reveal extra information, honest escrows regularly file 'garbage' allegations at random times. These are indistinguishable from genuine allegations, and hence serve to hide them. SAE scalability ensures that this doesn't hurt performance significantly.

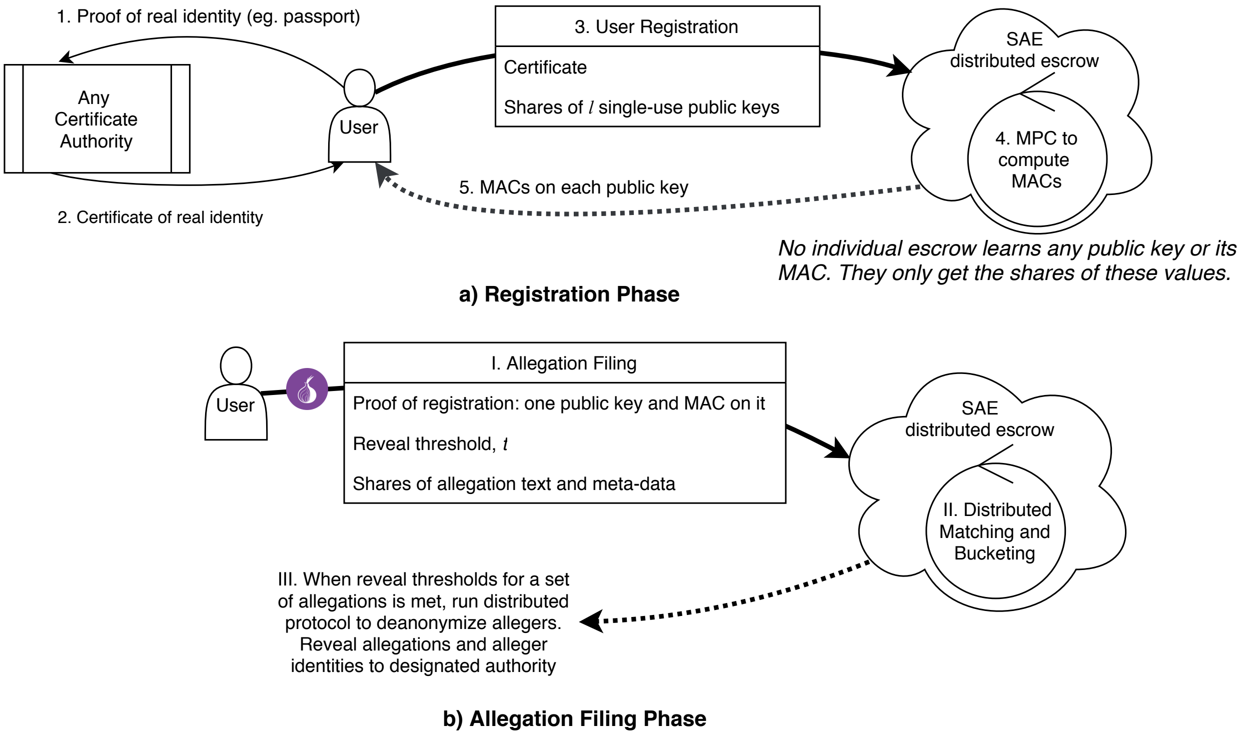
1. Proof of real identity (eg. passport)
Any Certificate Authority
2. Certificate of real identity
User
3. User Registration
Certificate
Shares of l single-use public keys
SAE distributed escrow
4. MPC to compute MACs
5. MACs on each public key
No individual escrow learns any public key or its MAC. They only get the shares of these values.
a) Registration Phase
User
I. Allegation Filing
Proof of registration: one public key and MAC on it
Reveal threshold, t
Shares of allegation text and meta-data
SAE distributed escrow
II. Distributed Matching and Bucketing
III. When reveal thresholds for a set of allegations is met, run distributed protocol to deanonymize allegers. Reveal allegations and alleger identities to designated authority
b) Allegation Filing Phase

Fig. 1. An overview of the SAE protocol. The figure shows a) the user registration phase and b) the allegation filing phase. Numbers indicate the order in which operations are performed. Thick-continuous and thick-dotted lines indicate one-to-many and many-to-one communication respectively. The line in b) with the 'Tor' symbol denotes an anonymous channel.

C. *Protocol Overview*

Figure 1 shows the high-level flow of the SAE protocol. We describe the individual sub-protocols for each of the stages in §IV.

Registration. SAE uses real-world (strong) identities to ensure accountability. Prior to registering with SAE, a potential alleger (called a user in the sequel) proves their real identity to a certifying authority (CA) and gets its signature on their public key. The CA may be the user's employer or university registering all its employees and students into the system, or even an independent entity verifying physical identities like passports.

To register with a SAE, the user authenticates to all escrows using the CA certificate. The escrows and the user then run a cryptographic protocol during which the user gets authentication tokens (in particular, MACs) on a fixed number l of fresh public keys. Each of these l keys can be used once to file an allegation (for instance, this step be repeated annually to allow l allegations per user per year). Importantly, the escrows only learn individual *shares* of these keys, but neither the full keys, nor the MACs on them. This prevents the escrows from learning the identity of a user when the user files an allegation later, but allows a majority of escrows to reconstruct the identity (by pooling their shares of the public key) when an allegation has to be revealed.

For their own benefit, users should register ahead of time, even when they see no need to file an allegation. This prevents timing correlation channels. For example, if an accused is expecting an allegation due to a recent incident, and colludes with a escrow, then the act of registration by the potential alleger may provide a strong hint of a pending allegation. Ahead of time registration removes this channel of inference and could be enforced, for instance, by a company asking its employees to register with an allegation escrow service as soon as they join the company.

Allegation filing. When the user wants to file an allegation, they contact the escrows, providing one of the l public keys and the MAC on it, which the escrows can verify. The verification tells each escrow that this user has registered before, but doesn't reveal the identity of the user, since no escrow has seen these in cleartext before. After this, the user provides the allegation's text along with some meta-data in a specific cryptographic form, and a *reveal threshold*—the minimum number of allegations that must match before this one is revealed.

Matching, thresholding and revelation. The material provided with each allegation is fed into a novel matching and bucketing algorithm. This algorithm matches allegations to each other. As soon as a set A of matching allegations, each with a reveal threshold $\leq |A|$ (the size of A), is found, they are revealed to a designated authority for further action. The revelation contains the real identities of the allegers and the full texts of their allegations. The designated authority can then take appropriate action.

III. Threshold Cryptographic Tools

In this section, we present threshold cryptographic protocols that we use in SAE. We first describe the necessary threshold primitives, and then design a distributed verifiable pseudo-random function (DVRF) protocol.

A. Multi-Party Computation and Secret Sharing

An MPC protocol enables a set of parties $\{P_1, P_2, ..., P_n\}$ to jointly compute a function on their private inputs in a privacy-preserving manner [7, 12, 25, 43]. In particular, every party P_i holds a secret input value x_i, and $P_1,...,P_n$ agree on some function f that takes n inputs and provide $y = f(x_1,...,x_n)$ to a recipient while making sure that the following two conditions are satisfied: (i) *Correctness:* the correct value of y is computed; (ii) *Secrecy:* the output y is the only new information that is released to the recipient.

An (n,f) Shamir secret sharing [38] allows a dealer to distribute shares of a secret among n parties $\{P_1,...,P_n\}$ such that any number set of $\leq f$ shares reveals no information about the secret itself, while an arbitrary subset of shares larger than f allows full reconstruction of the shared secret. Since in some secret sharing applications the dealer may benefit from behaving maliciously, parties also require a mechanism to confirm that each $f+1$ subset of shares combine to form the same value. To solve this problem, Chor *et al.* [13] introduced verifiability in secret sharing, which led to the concept of *verifiable secret sharing* (VSS) [6, 20, 24, 35].

In our construction we use the MPC protocol by Gennaro *et al.* [24]. It uses VSS; Pedersen commitments [35] on the Shamir shares are provided to all parties. It works on secrets in a prime-order ring $\mathbb{Z}_q$ and a multiplicative group $\mathbb{G}$ of order q, which is linear in the security parameter λ. To aid secret-sharing, we use collision resistant hash functions H_1 and H_2 that map arbitrary length strings and public keys respectively to $\mathbb{Z}_q$.

B. Bilinear Pairings

Let $\mathbb{G}_1$, $\mathbb{G}_2$, $\mathbb{G}_T$ be multiplicative, cyclic groups of prime order q. Let g_1, g_2 be generators of $\mathbb{G}_1, \mathbb{G}_2$ respectively. A map $e: \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is called bilinear if it has the following properties. (1) **Non-degenerate:** $e(g_1,g_2) \neq 1$. (2) **Bilinear:** For all $u \in \mathbb{G}_1, v \in \mathbb{G}_2, x,y \in \mathbb{Z}$, $e(u^x,v^y) = e(u,v)^{xy}$. (3) **Computable:** There is an efficient algorithm to compute $e(u,v)$ for all $u \in \mathbb{G}_1, v \in \mathbb{G}_2$. For ease of exposition, we assume that the pairing employed is symmetric, i.e., $\mathbb{G}_1 = \mathbb{G}_2 = \mathbb{G}$ [22, 29]. $g \in \mathbb{G}$ is a publicly known generator.

C. MPC Tools and Notation

VSS and MPC Notation. We denote the n shares of a secret value s by the set $[\![s]\!] = \{[\![s]\!]_1,...,[\![s]\!]_n\}$, where $[\![s]\!]_j$ represents the VSS share of party P_j. For SAEs, we use $n = 2f+1$.

As the employed VSS protocol is additively homomorphic, operations $[\![x_1+x_2]\!]_j = [\![x_1]\!]_j + [\![x_2]\!]_j$, and $[\![cx_1]\!]_j = c[\![x_1]\!]_j$ for a known constant $c \in \mathbb{Z}_q$ can be computed by each P_j locally using her shares $[\![x_1]\!]_j$, $[\![x_2]\!]_j$. The computation of $[\![xy]\!]_j$ from given $[\![x]\!]_j$, $[\![y]\!]_j$ is an interactive process and requires cooperation from $2f+1$ parties [24]. This protocol has identifiable abort [28] and can identify which the non-cooperating parties (if any) are; thus, a malicious-but-cautious party will always cooperate. We formalize identifiable abort as $\texttt{IdentifiableAbort}(i)$, where i indicates that P_i either offered wrong or no input.

Given threshold addition and multiplication, we can efficiently perform some additional operations. In the following, we list the employed VSS and MPC operations. These functions are cooperatively called by each escrow with their share of the inputs. When enough escrows cooperate, the functions return their values.

- $\text{VSS}(x)$: Verifiably secret share x among all the escrows such that $f+1$ of them can reconstruct x, but no fewer can [24].
- $\textsc{CombineShares}([\![x]\!]_j)$: Broadcast $[\![x]\!]_j$, gather shares from other $\geq f$ parties, and reconstruct the secret $x \in \mathbb{Z}_q$ if at least $f+1$ honest shares (including $[\![x]\!]_j$) are available.
- $\textsc{RandomCoinToss}()$: Return a share $[\![r]\!]_j$ of $r \in \mathbb{Z}_q$ chosen uniformly at random using distributed key generation [23, 30].
- $\textsc{PublicExponentiate}(h, [\![x]\!]_j, \texttt{recipients})$: Compute h^x, where x is secret-shared and h is a publicly known generator of a bilinear group $\mathbb{G}$ or $\mathbb{G}_T$ (see §III-B). The result is revealed as clear-text *only* to $\texttt{recipients}$, which is a set of parties. In our protocol, $\texttt{recipients}$ is either a given client or the set of all the escrows. This operation can be done efficiently with interaction since the result is revealed in the clear, not in secret-shared form.
- $\text{DVRF}([\![SK]\!]_j, [\![x]\!]_j, \texttt{flag\_proof}, \texttt{recipients})$: Return the VRF $F_{SK}(x)$ to $\texttt{recipients}$ (see §III-D). If $\texttt{flag\_proof}$ is true, also return the proof $\pi_{SK}(x)$, along with the VRF.
- $\textsc{Verify}\text{VRF}(PK,\pi,x)$: Verify that $\pi = \pi_{SK}(x)$, where SK is the secret key of the VRF (see §III-D) corresponding to the public key PK. Unlike above, this function can be executed locally by each escrow without interaction.

D. Distributed Verifiable Pseudorandom Functions (DVRFs)

VRFs. A verifiable pseudo-random function is a pseudo-random function $F_{SK}(x)$, along with a proof function $\pi_{SK}(x)$. A PPT adversary cannot distinguish $F_{SK}(x)$ from a random function if it doesn't have access to SK or $\pi_{SK}(x)$. However, given $\pi_{SK}(x)$ and a public key PK, a PPT can verify that $F_{SK}(x)$ was computed correctly. The formal definition of VRFs is given in Appendix A.

In SAE, we need to compute VRFs in a multi-party computation where both the key and the input values are available in a secret-shared form. Any VRF scheme can be transformed using general purpose MPC to work with shared key and shared input tags. However, keeping efficiency and practicality in mind, we choose a VRF construction by Dodis and Yampolskiy from [19].

In this construction, if a q-Decisional Bilinear Diffie Hellman Inversion (q-DBDHI) assumption holds in a bilinear group $\mathbb{G}$ with generator g, then

$$F_{SK}(x) = e(g,g)^{1/(x+SK)} \tag{1}$$

is a PRF. When coupled with a proof $\pi_{SK}(x) = g^{1/(x+SK)}$, it is a VRF. Here, SK is a private key chosen randomly from $\mathbb{Z}_q$, and $PK = g^{SK}$. To verify whether $y = F_{SK}(x)$, we can test whether $e(g^x \cdot PK, \pi) = e(g,g)$ and whether $y = e(g,\pi)$.

Distributed Input VRF. We need a distributed protocol for computing a VRF. However, we could not use distributed VRF (DVRF) schemes in [8, 9, 34] as, in SAE, the VRF computing parties (the escrows) know the input only in a secret-shared formed (this will become clear in §IV). So, we design a DVRF with secret-shared (or distributed) input messages. Our construction may be of independent interest to other distributed security systems.

Algorithm 1 Efficient MPC algorithm to compute DVRFs.

```
function DVRF($[\![SK]\!]$, $[\![x]\!]$, flag_proof, recipients)
    $[\![t_1]\!] \longleftarrow [\![SK]\!] + [\![x]\!]$
    $[\![blind]\!] \longleftarrow$ RANDOMCOINTOSS()
    ($[\![t_2]\!]$,IdentifiableAbort(i)) $\longleftarrow [\![t_1]\!] * [\![blind]\!]$
    $t_2 \longleftarrow$ COMBINESHARES($[\![t_2]\!]$)
    $[\![exp]\!] \longleftarrow t_2^{-1} * [\![blind]\!]$
    if proof_flag = True then
        PUBLICEXPONENTIATE($g$,$[\![exp]\!]$, recipients)
    else
        PUBLICEXPONENTIATE($e(g,g)$,$[\![exp]\!]$, recipients)
    end if
end function
```

A set of $2f+1$ escrows can efficiently compute $F_{SK}(x)$ or $\pi_{SK}(x)$ if each has a share of x and SK as shown in Algorithm 1. Here the result is sent only to recipients (either *all-escrows* or the given *client*). If flag_proof is false, DVRF computes $F_{SK}(x) = e(g,g)^{1/(x+SK)}$. Else, it computes $\pi_{SK}(x) = g^{1/(x+SK)}$. $g \in \mathbb{G}$ is a group generator. Given $\pi_{SK}(x)$, anybody can compute $F_{SK}(x) = e(\pi_{SK}(x), g)$. This is also why we omit F from VERIFYVRF's inputs.

Algorithm 1 first inverts $[\![x]\!] + [\![SK]\!]$ which takes two multiplications, and then exponentiates it. The only values available in clear-text (i.e., not information-theoretically hidden by the secret sharing) are t_2 and the final output. t_2 is uniformly distributed and independent of the input since it is blinded. Hence, this algorithm does not reveal any information about the inputs beyond what is revealed by the output.

IV. SAE CONSTRUCTION

In this section, we present the detailed cryptographic protocols we use to implement a secure allegation escrow. A formal summary of the protocol is given in Figure 2.

A. Format of an Allegation

An allegation escrow must have some mechanism to determine whether or not two allegations match. For this, along with free-form text describing their allegation, allegers provide structured meta-data describing the allegation. Escrows deem that two allegations match if the allegations' meta-data are identical. Although simple, this mechanism is quite effective—it is also used in Callisto [3], a deployed (non-cryptographic) escrow.[1]

Allegation meta-data is a formatted string containing specific fields. For instance, it could contain: 1) the identity of the accused and, 2) the type and intensity of a crime. The identity can be specified either as a name or as a unique identifier, if available. In an institutional setting for instance, the user could select from a drop-down list of other employees/students in that institute. The 'type and intensity' of crime is selected from a drop-down list containing entries like 'sexual harassment', 'sexual assault', 'petty theft', 'fraud ($< \$10^3$)', 'fraud ($\geq \$10^3, < \$10^6$)', 'fraud ($> \$10^6$)' and 'racial discrimination by a person in power'.

Along with the meta-data and free-form text, the user also submits a reveal threshold—the size of the smallest group of matching allegations that this allegation can be revealed with. Unlike prior work [26, 32, 37], which only supports a single matching threshold throughout the system, we allow each user to pick a threshold to their own satisfaction with each allegation.

B. Initialization

All escrows register with a standard PKI. They use this to form secure, two-way authenticated TLS links with each other pairwise. These are used for all communications among the escrows. During both registration and filing, the user and escrows use a session ID to ensure all escrows are talking to the same user.

The escrows use RANDOMCOINTOSS() to generate individual shares of private DVRF keys SK_I, SK_R and SK_i that are later used to, *respectively*: 1) register and authenticate users, 2) reveal user identities when required and 3) match allegations in each bucket i. Since there are infinitely many buckets, SK_i (for $i \in \{0,1,...\}$) is generated lazily when required. How these keys are used will be explained later. The public component of SK_I, denoted PK_I, is also generated using PUBLICEXPONENTIATE and publicly published. All shares use a fixed recombination threshold of $f+1 = \lfloor \frac{n+1}{2} \rfloor$, so all the escrows must cooperate to perform operations with these keys, and any minority can be compromised by an adversary without violating any of SAE's properties.

C. User Registration, Allegation Filing and Revelation

Registration. The user first obtains a certificate of real identity (e.g., using their passport/employee ID) from an appropriate certificate authority (e.g., their employer). This authority is trusted to verify the identity of the user in the real world, denoted as ID. During registration, the user forms a two-way authenticated TLS link with each escrow using this certificate, and non-repudiably signs all communication during registration.

The user generates l random one-time public-private key pairs and secret shares the public parts, $pk_1,...,pk_l$ among the escrows. Each of these can be used to file one allegation later.

Using the MPC protocol for DVRF, the escrows compute a MAC on each of these public keys pk_i, as $(F_{SK_I}(H_2(pk_i)), \pi_{SK_I}(H_2(pk_i))$ using their secret-shared private DVRF key generated at initialization, SK_I. H_2 is a collision resistant hash function from the set of public keys to $\mathbb{Z}_q$. Each escrow learns only its share of the public key pk_i and its share of the computed MAC. Neither the registering user nor any escrow learns the full MAC.

The escrows also compute a PRF $F_{SK_R}(H_2(pk_i))$ using a different private key SK_R. Individual escrows learn the PRF, but nothing else. Each escrow stores the association between the user's real-world identity and $F_{SK_R}(H_2(pk_i))$ in a local map. This association is used when revealing allegations later.

At the end of the registration, every escrow knows the real user, but knows only one share of each of the public keys pk_i the user provided and one share of the MAC computed on it. Consequently, when presented with one of these public keys and its MAC later, no minority of escrows can link the key back to a specific registered user.

[1] Note that if a different matching criterion is used, it must be unambiguous, since false positive matches can cause allegations to be revealed prematurely.

Allegation filing. A registered alleger files an allegation by connecting to the escrows over an anonymous communication channel that conceals the alleger's identity. The escrows' identities are authenticated with standard PKI. This is modelled by the functionality $\mathcal{F}_{\mathsf{anon}}$ (e.g., see [5]). In practice, solutions such as ToR [18] can be used.

During the filing, the alleger chooses a random public key pk from the set $\{pk_1,...,pk_l\}$ previously registered and submits 1) pk, 2) $\pi_{SK_I}(pk)$, the escrows' MAC on it, 3) the allegation's full text encrypted with a fresh symmetric key k, $Enc_k(a)$ (where a is the allegation's full text), 4) shares of k, 5) shares of a hash $H_1(m)$ of the allegation's meta-data m, where $H_1:\{0,1\}^* \rightarrow \mathbb{Z}_q$ is a collision resistant hash function, 6) an arbitrary reveal threshold t, and 7) signatures on all the above with the private key corresponding to pk.[2]

Since no escrow has seen the whole public key pk or the entire MAC on it before, no escrow can link it back to the specific user. However, all escrows can locally verify with VERIFYVRF that the MAC on the public key is legitimate and, hence, that the public key comes from a user who has previously registered. This verification only requires local computation by each escrow and no MPC, which improves efficiency (see §IV-F1 for why this is important). This is why we use a VRF. No other part of the protocol requires verifiability, so we only compute a PRF.

Note that no escrow has enough information to reconstruct the allegation, its meta-data or the identity of the alleger. A majority must cooperate to reconstruct any of these. This ensures the properties of allegation secrecy and alleger anonymity (§II), even if a minority of the escrows cooperate with the adversary.

Allegation revelation. Allegations are matched using a dedicated algorithm by the escrows. The algorithm is described in §IV-D. Once a majority of escrows determine that a set A of matching allegations can be revealed, i.e., they all have thresholds $\leq |A|$, the escrows combine their shares to decode the symmetric keys used to encrypt the texts of the allegations in A. These texts are provided to the designated recipient(s) for further action.

Along with the allegation texts, the escrows also reveal the real-world identities of the allegers who filed A. To obtain the identity of an alleger, the escrows compute the PRF $F_{SK_R}(pk)$ on the public key pk the alleger used to file the allegation (using DVRF as described in §III-D). Recall that the escrows also computed this PRF when the alleger registered and mapped the PRF to the victim's identity in a local store. Hence, to discover the user's identity, they merely need to look up the PRF in the store. This search is done in clear-text locally by each individual escrow and is efficient.

Real-world identities allow the escrows to reach out to all the allegers and provide the accountability property from §II-A.

Registered public keys must not be used twice. As just described, after an allegation filed with public key pk has been matched and revealed, the escrows map pk to the strong identity of the individual. Consequently, the key pk should not be used to file a second allegation unless the alleger wishes to de-anonymize themselves to the escrows. To allow users to file multiple allegations anonymously, a user registers l different keys during a single registration. As mentioned earlier, this registration can be repeated periodically, allowing for l allegation filings for every user within each registration period.

D. Matching and Thresholding

Next, we discuss how the escrows match allegations to each other and reveal sets of matching allegations when thresholds are met.

1) Matching protocol: We describe a simple MPC protocol that matches two allegations when their meta-data hashes are equal. We start by noting that, by design, our matching protocol does not allow any minority set of escrows to match two allegations on their own. Recall that each escrow receives only a share of the (collision-resistant) hash of the meta-data, $H_1(m)$, of each allegation. The shares are randomized, so a minority of escrows cannot check the equality of $H_1(m)$ and $H_1(m')$ using the shares alone. This property is important, else, an adversary who corrupts a minority of escrows can probe existing allegations to discover if an allegation against a specific individual exists without any honest parties being aware of such probing.

To compare a set of allegations for equality, all the escrows participate in DVRF (see §III-D) to compute a pseudo-random function $F_{SK}(H_1(m))$ for all allegations in the set. The resulting PRF is revealed in the clear to all escrows, but SK and $H_1(m)$ aren't. SK is a shared secret specially generated for each *set* of allegations being compared. The sets are determined by the bucketing protocol described below.

Escrows can locally compare meta-data m and m' by testing whether $F_{SK}(H_1(m)) = F_{SK}(H_1(m'))$. Since H_1 is collision resistant and $F_{SK}(\cdot)$ is a PRF, the adversary cannot produce $m \neq m'$ such that the test passes. Since SK is not used for any other purpose, the PRF reveals no additional information about m beyond equality relations between every pair of allegations for which $F_{SK}(m)$ has been computed. This takes $O(N)$ time for N allegations ($O(1)$ per allegation).

2) Bucketing Protocol for Reveal Thresholds > 2: The above matching protocol is secure when the reveal thresholds equal 2. Supporting higher thresholds securely *and* scalably requires more work. A collection A of matching allegations should be revealed when every allegation in A has a reveal threshold no more than the size of A (written $|A|$). One way to find such collections would be to run the above matching protocol on the set of *all* allegations irrespective of their thresholds and then locally determine whether an appropriate set A exists. However, this design is susceptible to a probing attack where an adversary interested in probing for the *existence* of a specific allegation, files the same allegation with a very high threshold. By corrupting just one of the escrows, the adversary could then compare this allegation to all other allegations in the system, without any risk that its own false allegation would ever be revealed (since the probe allegation has a very high threshold). To deter such attacks, we control which allegations *can* be compared to each other. We ensure that *if two allegations can ever be compared by a minority of escrows, then they will be revealed at the same time, if at all.* That is, two allegations can be compared by a minority only if they are waiting for the same number of matching allegations. Now, if the adversary tries to probe with a fake allegation, the fake allegation (and hence the adversary's real-world identity) is exactly as likely to be revealed as the alleger's actual matching allegation. Additionally, this allegation now requires one less match to be revealed.

To just learn the *number* of allegations against a person, the adversary must risk leaving a non-repudiable paper trail. If the adversary is a guilty party seeking to determine the number of escrowed allegations against them, they also risk precipitating the revelation of

[2] For the formal security, we demand $Enc_k(\cdot)$ to be non-committing encryption. We define it formally in Appendix B, and refer to [14] for a simple construction.

an honest allegation, which may have otherwise remained escrowed forever. We assume the adversary won't take such risks.

Note that the above attack only works for threshold > 2. If an honest alleger's threshold is 2, SAE doesn't admit any attacks not present in a single trusted-party implementation, even if the adversary is willing to risk filing a probe allegation. Prior work on single trusted-party based allegation escrows [3] only supports thresholds of 2, and still demonstrates social utility in allegations of sexual misconduct.

It is possible to use generic MPC to avoid such probing attacks. However, the time taken to process one allegation would then increase with the number of allegations already present in the system (i.e. $O(N)$). Our protocol is $O(1)$. We discuss why efficiency is important in §IV-F.

To keep track of how many matches each allegation needs, each escrow independently maintains buckets numbered 0, 1, 2, 3, $\ldots$. An allegation is in the i^{th} bucket only if it is waiting for $\leq i$ more allegations. Allegations are revealed iff they reach bucket 0. An allegation may be present in more than one bucket. Algorithm 2 determines which allegation occupies which buckets.

Only allegations *within* a bucket may be compared to each other. To ensure this, each bucket i is associated with an independently chosen secret key SK_i, which is shared among the escrows (SK_i is generated lazily when bucket i is first used). When an allegation is added to bucket i, the escrows compute $F_{SK_i}(H_1(m))$ for that allegation using DVRF (see III-D). Any escrow can use this to locally compare any two allegations in bucket i. Since, by design, $SK_i \neq SK_j$ if $i \neq j$, $H(m)$ and $H(m')$ cannot be compared using $F_{SK_i}(H(m))$ and $F_{SK_j}(H(m'))$ when $i \neq j$. Allegations that are known to match each other, either directly because they are in the same bucket or indirectly by transitivity, are said to belong to the same 'collection'. When allegations from two different collections are found to match, the collections coalesce into one. The resulting collection spans the union of buckets spanned by the parent collections and contains the union of allegations. Every allegation belongs to exactly one collection at any given time. To copy all allegations in a collection into a new bucket, the PRF for only one allegation's meta-data needs to be computed, since all allegations in a collection have identical meta-data.

This algorithm trivially satisfies the property that, once two allegations are known to be equal to each other, they belong to the same collection and are revealed together (if at all). This deters the probing attacks described above that motivated this elaborate mechanism. We also prove that the thresholding algorithm is 'correct':

Theorem 1 (Correctness)**.** *Algorithm 2 reveals a collection if and only if the thresholds of all allegations in it are satisfied.*

Proof: We begin by proving that the following three properties hold whenever all five rules of Algorithm 2 have been applied to saturation (meaning no further rule applies). (1) every collection spans a contiguous range of buckets, (2) every collection A spans $|A|$ buckets, i.e. $|A| = Max(A) - Min(A) \overset{\text{def}}{=} Span(A)$, (3) every allegation in a collection A has a threshold $\leq |A| + Min(A)$ and hence can be revealed if $Min(A)$ more matches are available.

The first property can be proved as an invariant that is trivially maintained by rules 2, 4 and 5 with rule 1 as the base case. Now, two collections coalesce only if they share a bucket (and hence their allegations may be compared). Since the union of contiguous, overlapping segments is contiguous, rule 3 also maintains the invariant.

To prove the second property, note that in any collection A, all allegations have a threshold $\leq Max(A)$ (because $Max(A)$ doesn't decrease). If $Span(A) < |A|$, $Max(A) = (Max(A) - Min(A)) + Min(A) = Span(A) + Min(A) < |A| + Min(A)$. Hence, rule 2 can be applied repeatedly until $Span(A)$ increases to equal $|A|$. So, $Span(A) \geq |A|$. We now prove that $Span(A) \leq |A|$ is an invariant with rule 1 as the base case. Rules 4 and 5 trivially maintain the invariant. Rule 2 would not apply if it causes the invariant to be broken, as there is at least one allegation with threshold $Max(A)$ if A is not yet revealed (which is when rule 2 applies). The threshold condition for this allegation will not be met if $Span(A) > |A|$, as it implies the threshold $t = Max(A) \overset{\text{def}}{=} Min(A) + Span(A) > Min(A) + |A|$. Applying Rule 3 to create C out of A and B maintains the invariant. $|C|$ spans a union of the parent's buckets, hence $Span(C) \leq Span(A) + Span(B) \leq |A| + |B| = |C|$ because A and B are disjoint. Hence, the invariant is maintained.

Algorithm 2 Rules for the secure thresholding algorithm BUCKETING, whose interface is described in §V. It reveals a set of allegations if and only if all of their thresholds are satisfied by that set.

If applicable, apply rule 3 first. Else apply the other rules (in any order) till no further rules apply. Rules 1-4 apply only to collections that haven't yet been revealed. $Max(A)$ and $Min(A)$ are the highest and lowest buckets occupied by collection A respectively.

1) When an allegation with threshold t is filed, it forms a singleton collection and is added to bucket $t-1$ (since $t-1$ other allegations must match the allegation before it is revealed).
2) If $Min(A)$ is the smallest bucket occupied by a collection A and every allegation in A has a threshold $< |A| + Min(A)$, A is copied to bucket $Min(A) - 1$. Note that A still occupies the buckets it used to occupy. Copying merely adds the collection to a new bucket.
3) When two collections overlap and occupy the same bucket, and their allegations are found to match (see §IV-D1), they coalesce into one collection.
4) When a collection reaches bucket 0, all of its allegations are revealed as described in §IV-C.
5) If a collection A is revealed, we make sure it continues to occupy buckets 1,...,$|A|$, even as A grows. This enables future matching allegations to be revealed.

The third property is explicitly maintained as an invariant by rule 2 and is trivially satisfied by rules 1 and 5. Rule 3 is applicable in two ways. First, when a new allegation arrives in between an older collection, the property is not broken. Second, if two existing collections, A and B, coalesce into C by rule 3, one is 'above' another (remember, rule 3 is applied preferentially). Let $Min(B) = Max(A)$, without loss of generality. Then, $Min(C) = Min(A)$, hence allegations in A satisfy the property. The drop in Min for allegations in B is $Min(B) - Min(C) = Max(A) - Min(A) = Span(A) \leq |A|$ (we proved above that all rules maintain $Span(A) \leq |A|$ as an invariant). This drop in Min is compensated by a corresponding increase in size of the collection by $|A|$.

We now use these properties to prove correctness. The third property implies that when a collection A is revealed, the threshold condition is satisfied for all revealed allegations, since $Min(A) = 0$. To prove the other direction, let there be n matching allegations such that all their thresholds are $\leq n$. Assume for contradiction that they are not revealed. This means that they all belong to buckets $1,...,n-1$. By the pigeonhole principle, there will be one bucket with multiple allegations which will start coalescing with rules 2 and 3. If the process stops with a collection of size $k<n$, $n-1-k$ buckets will be left with $n-k$ allegations, because property 2 ensures the size of a collection equals its span. Again, by the pigeonhole principle, the coalescing process starts. This continues till there is only one collection with n allegations that spans buckets $0...n-1$ and all n allegations get revealed. Hence, a set of n matching allegations are revealed if and only if all their thresholds are $\leq n$. ■

BUCKETING(`buckets`): Interface to the Algorithm The real protocol (Figure 2) and ideal protocol (Figure 3) interface the bucketing algorithm with the BUCKETING function. It takes as input the set of buckets, `buckets`. Each bucket i, (denoted as `buckets[`i`]`) is a set of tuples (id,M) describing the allegations in that bucket. id is a unique allegation identifier[3], and M is a representation of the meta-data which can be compared for equality to determine which allegation matches which others *within a bucket*. The function BUCKETING returns a task T if more rules apply in Algorithm 2. Else, it returns $\perp$. Task T instructs the caller to move an allegation $T.ID$ to a bucket $T.i$. For ease of exposition, when a collection is added to a new bucket, BUCKETING produces one task per allegation. In practice, only one is necessary, since all meta-data are identical. Note that BUCKETING doesn't need to implement step 1 of Algorithm 2, since it is implemented by $\mathcal{F}_{SAE}$. It doesn't maintain its own state.

E. Computation Cost

The computationally expensive steps in the SAE protocol are the VRF/PRF computations that require interaction between the escrows, since they involve multiplication and exponentiation over shared secrets. However, the number of such computations scales very well and doesn't increase with the number of users as well as the number of allegations filed.

Registering a new user requires two computations for every public key pk that the user provides, one to compute the MAC on the key, and one to compute $F_{SK_R}(pk)$. Filing an allegation does not, of itself, require any PRF computation. The cost to move a collection of allegations to a new bucket can be reduced by observing that, we only need to compute PRF for one of the allegations, since they all have identical meta-data. We can prove that, in an amortized sense, we need to compute the `PRF` at most twice for each allegation, independent of its threshold. Revealing an allegation requires one more PRF computation (to discover the identity of the alleger). To summarize, on average, every filed allegation requires 2 PRF computations if it is never revealed, and 3 PRF computations if it is revealed.

F. Why is Efficiency Important?

1) Resist DoS attacks: Like any public service, SAE can be flooded with client requests to mount a denial-of-service (DoS) attack. MPC systems in particular, are susceptible to such attacks since the client can trigger expensive computations at little cost to itself. SAE has two properties that prevent the adversary from exploiting this asymmetric computation cost. 1) **Bounded MPC:** Each registered real identity can trigger only a bounded number of MPC operations. 2) **Scalability:** The amount of work required to process a user request doesn't increase with the number of allegations or number of registered users.

Bounded MPC holds in SAE since MPC is only triggered after authentication. Failed authentications do not cause MPC operations, since authentication is a local computation. Further, each user (with a real identity) is only allowed to register a fixed number l of public keys in any registration period and each key can only be used to file one allegation. So, a registered user can cause at most $(2+3)l=5l$ PRF computations in any registration period (see IV-E). For $l=10$ and a registration period of one year, this amounts to at most 50 PRF computations per year per *real* user, which is an extremely low rate for an effective DoS attack.

The concurrent work WhoToo [32] is not **scalable** in the sense above. Filing the N^{th} allegation takes $O(N)$ MPC operations, bringing the cost of filing N allegations to $O(N^2)$. Asymptotically, this is the same as black-box MPC. Further, it allows users to file an arbitrary number of allegations. There is no obvious way to prevent this with their authentication protocol without breaking the "**bounded MPC**" property. Hence, an adversary could file any number of arbitrary allegations (e.g., against random strings), and prevent proper functioning.

2) Larger Allegation Pool: To maximize the probability that a matching allegation will be found, the escrow must support many users. This increases the pool of potential corroborators and the probability of a match. Scalability is essential to enable this.

3) Avoid Timing Side-Channels: If a crime's perpetrator learns (through a compromized escrow) that an allegation was filed two days after the crime was committed *and* filings are otherwise rare, then the perpetrator may reasonably conclude that their victim filed the allegation. We excluded such side channels in the threat model. To realize this in practice, honest escrows (and other external well-wishers) can regularly file decoy allegations. Since SAE maintains anonymity and privacy, the adversary cannot distinguish decoys from real allegations. Those filing decoy allegations must register their separate (real) identities for doing so, and enter a contractual obligation to ensure no decoy ever gets matched with real ones. This can be ensured by filing allegations against random strings. Decoy allegations will not slow down the system significantly since SAE is scalable.

V. Security Analysis

To model security and privacy we use the UC framework, which allows SAE to compose with other cryptographic schemes while maintaining security.

Attacker Model. Agents (allegers and escrows) in our system are interactive Turing machines that communicate with an ideal functionality $\mathcal{F}_{SAE}$. The adversary $\mathcal{A}$ is a PPT machine with access to an interface $\mathsf{corrupt}(\cdot)$. It takes an agent identifier and returns the internal state of the agent to the adversary. All subsequent incoming and outgoing communication of the agent is then routed through $\mathcal{A}$. The adversary is f-bounded, and can corrupt a minority $f<n/2$ of escrows and any number of allegers. For formal security, we

[3] In the real protocol, pk can be the allegation identifier, since each key is used only once. For ease of exposition, each user can register for and file only one (not l) allegations in our ideal protocol. Hence, we use ID as id. The full protocol can be modeled by each user having l distinct IDs.

Escrow Initialization: Every j^{th} escrow executes:

```
1. $[\![SK_R]\!]_j \leftarrow$ RandomCoinToss()
2. $[\![SK_I]\!]_j \leftarrow$ RandomCoinToss()
3. $PK_I \leftarrow$ PublicExponentiate($g$, $[\![SK_I]\!]_j$, all-escrows)
4. $[\![SK_i]\!]_j \leftarrow$ RandomCoinToss() $\forall i \in \{0,1,2,...\}$
   #$SK_i$ is generated lazily when required
5. Set identities←{}; allegations←{}; buckets←{}
```

Client Initialization and Registration: The client uses ID, a certificate previously obtained from a CA, to authenticate to the escrows. The escrow's identities are managed with PKI. The secure authenticated channel is idealized using $\mathcal{F}_{\mathsf{smt}}$ [10]. Client signs all messages with ID. The client executes:

```
1. $(pk,sk) \leftarrow \mathrm{Gen}(1^\lambda)$ # Signing key-pair
2. Broadcast (“Register”, ID) idealized using $\mathcal{F}_{\mathsf{B}}$
3. VSS($H_2(pk)$) among the escrows
Every $j^{th}$ escrow executes:
4. $\perp \leftarrow$ DVRF($[\![SK_I]\!]_j$,$[\![H_2(pk)]\!]_j$, True, client)
5. $R \leftarrow$ DVRF($[\![SK_R]\!]_j$,$[\![H_2(pk)]\!]_j$, False, all-escrows)
6. identities[$R$] ← ID
The client executes:
7. Receive $\pi_{SK_I}(H_2(pk))$ from escrows' DVRF call in step 4
8. Store $(pk,sk)$ and $\pi_{SK_I}(H_2(pk))$ for future use.
```

Allegation Filing: With allegation text a, $m = H_1(\text{meta}-\text{data})$, reveal threshold t, fresh symmetric encryption key k and, $(pk,sk,\pi_{SK_I}(H_2(pk)))$ produced during registration, the client connects to each escrow over an anonymous communication channel, idealized using functionality $\mathcal{F}_{\mathsf{Anon}}$ [5]. It signs all communication with sk, which escrows verify before accepting the input. The escrows process allegations serially. They identify an allegation by the pk used to file it. The client executes:

```
1. Broadcast (“File”, $pk$, $t$), idealized using $\mathcal{F}_{\mathsf{B}}$
2. Broadcast ($\pi_{SK_I}(H_2(pk))$,$Enc_k(a)$), idealized using $\mathcal{F}_{\mathsf{B}}$
3. VSS($m$), VSS($k$) among the escrows

Every $j^{th}$ escrow executes:
          # Identity Verification and Filing
4. If VerifyVRF($PK_I$,$\pi_{SK_I}(H_2(pk))$,$H_2(pk)$) fails, abort
5. If allegations[$pk$] exists, abort
6. allegations[$pk$] ← ($t$,$pk$,$Enc_k(a)$,$[\![m]\!]_j$,$[\![k]\!]_j$)
7. $M \leftarrow$ DVRF($[\![SK_{t-1}]\!]_j$,$[\![m]\!]_j$, False, all-escrows)
8. buckets[$t-1$] ← buckets[$t-1$] $\cup$ {($pk$,$M$)}
          # Matching and Bucketing
9. $T \leftarrow$ Bucketing(buckets)
10. While $T \neq \perp$
11.    # Move allegation $T.id$ to bucket $T.i$
12.    $[\![m]\!]_j \leftarrow$ allegations[$T.id$].$[\![m]\!]_j$
13.    $M' \leftarrow$ DVRF($[\![SK_{T.i}]\!]_j$,$[\![m]\!]_j$, False, all-escrows)
14.    buckets[$T.i$] ← buckets[$T.i$] $\cup$ {($T.id$, $M'$)}
15.    $T \leftarrow$ Bucketing(buckets)
          # Reveal Allegations
16. For ($id$,$M$) in buckets[0]
17.    buckets[0] ← buckets[0]$\setminus$($id$,$M$)
18.    ($t$,$pk$,$Enc_k(a)$,$[\![m]\!]_j$,$[\![k]\!]_j$) ← allegations[$id$]
19.    $R \leftarrow$ DVRF($[\![SK_R]\!]_j$,$H_2(pk)$, False, all-escrows)
20.    ID ← identities[$R$]
21.    $k \leftarrow$ CombineShares($[\![k]\!]_j$)
22.    $a \leftarrow Dec_k(Enc_k(a))$
23.    Output the revealed allegation ($t$,$a$,ID)
```

Fig. 2. The SAE protocol. The client has a certificate of identity, ID, from a certificate authority. MPC, cryptographic and communication primitives used in this protocol are defined in §III-C. PublicExponentiate and DVRF may return `IdentifiableAbort(i)` if the i^{th} escrow behaves maliciously. This aborts the protocol, which we omit to reduce clutter. We show the registration of only one key for clarity. In practice, the client registers l keys, and randomly picks one to use during filing. We formalize the functionalities $\mathcal{F}_{\mathsf{smt}}$, $\mathcal{F}_{\mathsf{B}}$ and $\mathcal{F}_{\mathsf{Anon}}$ in Section V. Bucketing is defined in §IV-D2.

consider the static corruption model, i.e., the adversary commits to the identifiers of the agents it wishes to corrupt ahead of time.[4]

Communication Model. We assume the network to be bounded-synchronous [24] such that the protocol execution occurs in discrete rounds. The agents are aware of the current round, and if a message is created at round i, it is delivered at the beginning of round $(i+1)$. Our model assumes that computation is instantaneous. In practice, this is justified by setting a maximum publicly known time bound on message transmission. If no message is delivered by beginning of the next round, then the message is set to $\perp$. For an example of the corresponding ideal functionality $\mathcal{F}_{\mathsf{syn}}$, we refer the reader to [10, 31]. The attacker is informed whenever some communication happens between two agents and the attacker can arbitrarily delay the delivery of the message between honest parties within the round boundaries.

The real-world protocol assumes the existence of a functionality $\mathcal{F}_{anon}$ (see [5] for an example), which provides user with an anonymous communication channel. Moreover, the protocol also assumes the existence of a broadcast channel for allegers to reliably communicate with all escrows and we model this as a bulletin board visible to all escrows (such as [42]) with an ideal functionality $\mathcal{F}_{\mathsf{B}}$.

Concretely, our real-world protocol uses $\mathcal{F}_{\mathsf{anon}}$, $\mathcal{F}_{\mathsf{B}}$ and $\mathcal{F}_{\mathsf{syn}}$ as subroutines. Moreover, we idealize the secure and authenticated channels that connect two parties in the real world using $\mathcal{F}_{\mathsf{SMT}}$ [10]. As a result, we specify in the $(\mathcal{F}_{\mathsf{B}},\mathcal{F}_{\mathsf{anon}},\mathcal{F}_{\mathsf{syn}},\mathcal{F}_{\mathsf{SMT}})$-hybrid model.

Universal Composability. Let $\mathsf{EXEC}_{\rho,\mathcal{A},\mathcal{E}}$ be the ensemble of the outputs of the environment $\mathcal{E}$ when interacting with the f-bounded adversary $\mathcal{A}$ and parties running the protocol ρ (over the random coins of all the involved machines).

Definition 1 (UC-Security). *A protocol ρ UC-realizes an ideal functionality $\mathcal{F}$ if for any adversary $\mathcal{A}$ there exists a simulator $\mathcal{S}$ such that for any environment $\mathcal{E}$ the ensembles $\mathsf{EXEC}_{\rho,\mathcal{A},\mathcal{E}}$ and $\mathsf{EXEC}_{\mathcal{F},\mathcal{S},\mathcal{E}}$ are computationally indistinguishable.*

Ideal Functionality. Figure 3 describes an ideal functionality $\mathcal{F}_{SAE}$, which models the intended behavior of a SAE in terms of functionality and security properties.

For a more modular treatment, our (UC) models only focus on the cryptographic aspects and we assume that all allegers have certificates of real identity from a trusted offline authority. Further, we omit the handling of session IDs (SIDs) in $\mathcal{F}_{SAE}$ to reduce clutter. Messages are assumed to be implicitly associated with SIDs.

In the ideal functionality, registered is the set of registered

[4] The static adversary remains a standard assumption employed by most practically relevant MPC systems today. [15]

```
Initialization
1. registered ← {}, allegations ← {},
   buckets ← {}, unique ← {}
Registration Invoked by client with identity ID
1. Send (“Register”, ID) to all escrows
2. If received $\perp$ from escrow $i$, then IdentifiableAbort($i$)
3. registered ← registered ∪ {ID}
Allegation Filing Invoked by client with identity ID, allegation
$a$, reveal-threshold $t$, and metadata $m$
1. If ID $\notin$ registered, then Abort
2. registered ← registered \ ID
3. Send (“File”, UNIQUE(ID), $t$) to all escrows.
4. If received $\perp$ from escrow $i$, then IdentifiableAbort($i$)
5. allegations[ID] ← $(t, m, a)$
6. buckets[$t-1$] ← buckets[$t-1$] ∪ {(ID, $m$)}
7. Send ($t-1$, UNIQUE(ID), UNIQUE(($t-1$, $m$))) to all
   escrows
              # Matching and Bucketing
8. $T$ ← BUCKETING(buckets)
9. While $T \neq \perp$
10.    # Move allegation $T$.ID to bucket $T.i$
11.    If received $\perp$ from escrow $i$, then IdentifiableAbort($i$)
12.    $m'$ ← allegations[$T$.ID].$m$
13.    buckets[$T.i$] ← buckets[$T.i$] ∪ {($T$.ID, $m'$)}
14.    Send ($T.i$, UNIQUE($T$.ID), UNIQUE(($T.i$, $m'$)))
        to all escrows
15.    $T$ ← BUCKETING(buckets)
              # Reveal Allegations
16. For each (ID, $M$) in buckets[0]
17.    buckets[0] ← buckets[0] \ (ID, $M$)
18.    If received $\perp$ from escrow $i$, then IdentifiableAbort($i$)
19.    ($t$, $m'$, $a$) ← allegations[ID]
20.    Send ($t$, $a$, ID) to all escrows


function UNIQUE($x$) begin
1. # Map input objects to unique numbers
2. if $x$ is in unique
3.    return unique[$x$]
4. end if
5. unique[$x$] ← |unique|
6. return unique[$x$]
end function
```

Fig. 3. The ideal functionality for SAE, $\mathcal{F}_{SAE}$. BUCKETING is a *local* algorithm (§IV-D2) that determines which matches are safe to reveal to the escrows.

users' identities—only registered users may file an allegation. The set of allegations filed so far is denoted by allegations. Each allegation is a tuple of the reveal threshold, meta-data, allegation text and real identity (which is known to the ideal functionality). unique counts the number of allegations filed so far, which allows us to assign a unique identifier to each allegation. If an escrow (say, the i^{th}) refuses to cooperate, $\mathcal{F}_{SAE}$ aborts, reporting the i^{th} escrow to the other escrows. This is denoted as IdentifiableAbort(i). We assume the adversary is malicious-but-cautious, and wouldn't want to get reported. If authentication fails, then $\mathcal{F}_{SAE}$ calls Abort without a parameter, since it wasn't any escrow's fault. In the real protocol, clients can file l allegations for each registration. For notational simplicity, we only show $l=1$ here but the $l>1$ case is a straightforward extension. (The textual description in Section§IV describes the real protocol for all l.)

Bucketing Algorithm. To scalably and efficiently implement reveal thresholds, we propose a bucketing protocol (see §IV-D2) that divides allegations into buckets. All escrows know which allegations *within* a bucket match each other. This makes the ideal functionality admit a somewhat surprising attack: If an *adversary* files an allegation, it learns whether other matching allegations exist in the same bucket in which the adversary's allegation is placed. These attacks are consistent with our threat model, which allows for probing attacks by adversaries. As explained in §IV-D2, the buckets in which an allegation is placed are carefully chosen to disincentivize these attacks by relying on our accountability property (see II-A). Note that BUCKETING is a local and non-cryptographic algorithm. It merely determines what information can be revealed to the adversary, and hence can be called from the ideal functionality.

Discussion. $\mathcal{F}_{SAE}$ satisfies the allegation secrecy, alleger anonymity and accountability properties described in §II, relative to our threat model. *Accountability* is ensured since, if a user files an allegation, $\mathcal{F}_{SAE}$ reveals the user's real identity (ID) as soon as the threshold is met. We already proved the bucketing protocol correct. *Allegation secrecy and alleger anonymity* are ensured because $\mathcal{F}_{SAE}$ reveals information about an allegation only in the following scenarios: (1) $\mathcal{F}_{SAE}$ reveals a user's identity when they register in the system (step 1). This is harmless since users register irrespective of whether or not they intend to file an allegation. (2) As the bucketing protocol progresses, $\mathcal{F}_{SAE}$ reveals which allegations match which others (step 14). We discussed above why this information doesn't violate our properties. (3) The threshold of an allegation is revealed when it is filed (step 3), but hiding the threshold is not part of our threat model. (4) Finally, the entire allegation is revealed when its threshold is met (step 20). Note that, we prove the bucketing protocol correct in §IV-D2, Theorem 1.

Figure 2 presents the pseudocode for our cryptographic protocol. We prove UC-security in the $(\mathcal{F}_{\mathsf{B}}, \mathcal{F}_{\mathsf{anon}}, \mathcal{F}_{\mathsf{syn}}, \mathcal{F}_{\mathsf{SMT}})$-hybrid model. Theorem 2 holds for any UC-secure realization (as defined in Definition 1) of $\mathcal{F}_{\mathsf{B}}$, $\mathcal{F}_{\mathsf{anon}}$, $\mathcal{F}_{\mathsf{syn}}$, and $\mathcal{F}_{\mathsf{SMT}}$. We provide a proof sketch of Theorem 2 in Appendix B.

Theorem 2. *Let* VSS *be a secure verifiable secret sharing scheme,* (RANDOMCOINTOSS, PUBLICEXPONENTIATE) *be a secure DKG protocol, (*DVRF, VERIFYVRF*) be a secure distributed input DVRF protocol, H_1 and H_2 be collision resistant hash functions, and (E,D) be a non-committing symmetric encryption scheme. Let the employed signature scheme be strongly existentially unforgeable. Then, the SAE protocol UC-realizes the ideal functionality $\mathcal{F}_{SAE}$ defined in Figure 3 in the $(\mathcal{F}_{\mathsf{B}}, \mathcal{F}_{\mathsf{anon}}, \mathcal{F}_{syn}, \mathcal{F}_{\mathsf{SMT}})$-hybrid model.*

VI. Implementation and Evaluation

Implementation. We built a prototype of our design in Java, with our own implementation of the GRR MPC protocol [24]. We use SCAPI [39] version 2.3 for establishing communication channels

	SAE DVRF	WhoToo (online)	WhoToo (precompute)
Interactive MPC Operations			
Multiply	1	1	N
PublicExponentiate	1	$3N+q+1$	q
RandomCoinToss	1	q	$q+2$
CombineShares	1	1	1
Local MPC Operations			
Add	1	0	0
Multiply Constant	1	N	1

Fig. 4. Number of various MPC operations required per allegation per escrow for DVRF computation in SAE, compared with WhoToo [32]. Here N is the number of allegations filed so far, and q is WhoToo's global reveal threshold. Here we only show the cost for WhoToo's matching protocol; secure identity verification is separate. SAE requires at-most five DVRF computations per allegation (amortized), including for identity verification. Note that, WhoToo's *per-allegation* complexity is $O(N)$, and hence $O(N^2)$ complexity for N allegations.

and its bindings to OpenSSL [21] version 1.1 for hashing, symmetric encryption and public-key encryption. We use the Java bindings to the Pairing Based Cryptography library, jPBC [16] version 2.0 for pairing based cryptography primitives. For operations in $\mathbb{Z}_q$, we use Java BigInteger. To maintain each escrow's persistent state, we use a MySQL database to achieve security and scalability. To demonstrate scalability, we pre-populate the database to simulate one million pre-existing allegations. Since our computational complexity per allegation/registration does not depend on the number of pre-existing allegations/registrations, this imposes a negligible overhead on the protocol. We evaluate our implementation to show that SAEs are fast enough for practical use.

Comparison with WhoToo [32]. There is no implementation for WhoToo to directly compare performance with. Instead, we count the number of cryptographic operations required for SAE and WhoToo, shown in figure 4. WhoToo requires $O(N)$ work per allegation, while SAE requires $O(1)$ work. To improve scalability, WhoToo offloads some work to an offline precomputation step. But non-trivial 'online' computation remains. As discussed in §IV-F, scalability is essential for a practical allegation escrow.

Latency and throughput. We first measure the latency and throughput of user-SAE interaction in a realistic setting. We set up to 9 escrows on Amazon AWS cloud servers, chosen to maximize geographical extent. In an experiment involving n escrows, the escrows run on servers in the first n of Virginia, Frankfurt, Sydney, N. California, Singapore, Sao Paulo, London, Seoul, and Mumbai. Each escrow runs on a M4.large AWS instance. At the time of the experiments, this provided 2 vCPUs, 8GB of RAM, and 'moderate' network performance. Each server runs up to 60 threads, the maximum supported on the machines; each thread handles one concurrent client request. We use up to 60 client replicas, all hosted on a single c4.4xlarge instance of AWS in Virginia. At the time of our experiments, this provides 16 vCPUs, 30GB RAM and 'High' network performance. Note that the SAE registration is embarrassingly parallel with respect to client requests—cost is dominated by network latencies and MPC computation, which require no syncing across client requests; synchronization is only needed for storing registered identities to a database. Allegation filing must be done serially one-by-one.

Latency: Figure 5 (top) shows the average latency for registering a new key as the number of escrows varies, in two configurations: When the escrows are lightly loaded (no concurrent requests) and when they are heavily loaded (60 concurrent clients). Latency is the time between when a user sends its request, to when it gets the SAE's MACs on its keys. There are three notable aspects here. First, as expected, the latency increases with the number of escrows (since the MPC becomes more complex). Second, increasing the number of concurrent clients does not increase the latency significantly. This suggests that the cost is *dominated* by the number of escrows and inter-escrow network latencies. Finally, even though the absolute latency numbers might look high (of the order of 10s of seconds), they are acceptable since user interaction with SAEs is relatively infrequent. In particular, users register new keys once every few months, so such latencies seem quite practical. Latency is not a concern for filing an allegation, since the user does not expect any immediate response from the escrows. The cost of matching and bucketing is better captured in terms of throughput.

Throughput: Next, we measure the throughput of SAE in terms of the number of key registrations and allegation filings it can handle per second. For registration, we use 60 concurrent clients. Allegations are filed serially. Figure 5 (bottom) shows the throughput as a function of the number of escrows. As expected, the throughput number decreases with increasing number of escrows.

For allegation filing, each client repeatedly files allegations with thresholds varying between 2 and 20, chosen from a truncated exponential distribution with mean 5. When a threshold of t is chosen, t matching allegations are created with 50% probability, and $t-1$ matching allegations are created the rest of the time. These, respectively, represent the cases where the allegation is eventually revealed and the worst-case (for performance) when the allegation is not actually revealed.

We believe that the throughputs in Figure 5 (bottom) are acceptable for SAE, since user operations are expected to be very infrequent. Moreover, each escrow can be separately replicated on several servers to get proportionally higher throughput.

Impact of network latency. Inter-escrow network latency dominates user-perceived latency. To test this, we emulate a network with Linux qdiscs [27, 33] on a single Amazon AWS c4.4xlarge instance. The escrow servers and our client occupy one core each. Every pair of escrow servers is given an emulated 100 Mbps link and 1 bandwidth×delay worth of buffer (the recommended buffer size for TCP to obtain full link utilization and minimal delay). We vary the latency of the emulated network links and Figure 6 plots the latencies of a) registering a key, and b) *processing* one allegation completely with no matches. These require two and one VRF computations, respectively. (Note that the user perceived latency of allegation *filing* is different from that of processing the allegation. Filing does not require any PRF computation.) The client latency increases linearly with the network latencies, showing that network latencies dominate. The rate of increase also increases with the number of escrows.

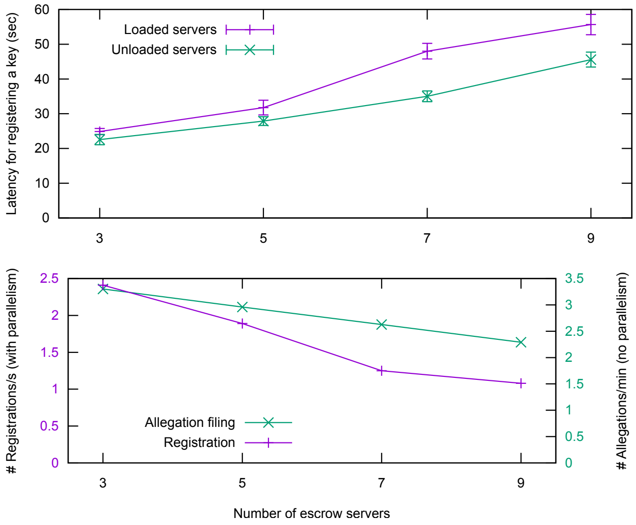

Fig. 5. Latency and throughput with SAE servers running on AWS instances in different locations. Registrations occur in parallel, while allegations are filed serially. Hence throughput is different for the two (note the units: seconds vs minutes).

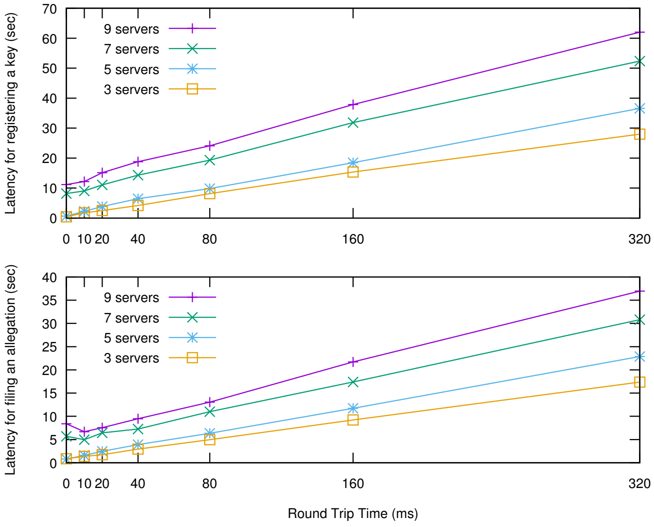

Fig. 6. End-to-end latency for registering a key and filing an allegation as the communication delay between every pair of Escrows on an emulated network is varied. Values are shown for number of Escrows varying from 3 to 9.

VII. Discussion

A. Deployment Considerations

Client Software. Users need client software to participate in the protocol. For users, a simple web-based interface is conveient. However, it is challenging to access anonymity services like ToR from within a browser. Asking users to download special software exposes them to other security risks. The very act of downloading the software indicates an intent to file an allegation, since not all users will use an anonymity service like ToR to do so. To prevent this channel of inference, the client software should be bundled with other commonly used software. Alternately, a small fraction of all visits to a popular web-page (e.g. organization's home page) could automatically trigger a download, producing cover traffic.

Practical Security. Like all software, SAE will have security vulnerabilities (there is no use in encrypting secrets if a buffer-overflow attack leaks the secret keys!). In addition to careful code audits, we could make multiple implementations that use independent hardware/software stacks and compare their outputs to see they are identical. If not, we halt the system until a security expert can find and fix the bug. This forces an attacker to find the *same* vulnerability on different hardware/software stacks, which is much more difficult. Such a heavy-handed approach is prudent since, here, security is much more important than performance and availability.

Non-technical considerations. Since SAE handles sensitive information, its social design requires considerable thought. While a full analysis is out-of-scope for this primarily technical paper, we discuss some issues here. When thresholds for a set of allegations are met, who are the designated recipient(s) to whom they are revealed? To avoid centralization, we could reveal them to the allegers themselves. They can then coordinate and report to the relevant authorities if they so choose. On the other hand, if allegations went to a central authority first, it could provide legal assistance to each alleger *before* introducing them to each other. It could also filter out any obivously fake/probe allegations. To keep this authority accountable, the escrows could notify allegers that their allegation has been matched, but not to whom. Project Callisto [3] currently follows a similar approach.

To discourage fake/probe allegations, SAE ensures allegers create a paper-trail when filing an allegation. This should be complemented with effective legal mechanisms to make this a significant deterrent. We refer the reader to prior social-science work [2, 4] for more discussions.

B. Future Work

Withdrawing/modifying allegations. A user can readily update an allegation's free-form text by sharing a new value. However, SAE cannot always let them withdraw an allegation or modify its meta-data/reveal threshold. Since an allegation's threshold could be met as soon as it is filed, allegers should file one only if they are comfortable with it being revealed. Nevertheless, allegers may want to modify one. For example if an allegation hasn't been revealed in several months/years, they may want to either withdraw it or reduce its reveal threshold. In SAE, this is hard, since for thresholds > 2, escrows may know that two allegations match even before they are revealed. An adversary can use this to probe for allegations; if they can withdraw their probe allegation, they can delete the paper trail that disincentivizes such probes (see §IV-D2).

Note that if an allegation isn't yet known match any other, we can allow its withdrawal. To do so, we must pick a new secret key for the bucket it is in and recompute all PRFs with this new key. Since this is computationally expensive, we can limit the overhead by recomputing PRFs at most once per week. Users are notified that withdrawal can take upto one week, which is acceptable in this context. Fully supporting allegation withdrawal and modification is interesting future work.

Other matching/reveal criteria. SAE matches allegations based on exact string equality. Could we support other criteria? Could we match on multiple fields, e.g. 'match only if at least two of (name/phone number/email address/employee id) match', 'match against this accused person only if the crime happened within this time-frame/physical coordinates', or 'match only against allegations filed in the last year'. In applications where some ambiguity is acceptable, it would be interesting to match based on softer criteria provided by machine learning, e.g., based on a person's picture, or a textual description. Note that these more complex criteria break transitivity. That is 'A matches B' and 'B matches C' doesn't imply 'A matches C'. Future work would need to define what it means for the thresholds for a group of allegations to be satisfied. Some allegers may be victims of the crime and others may be witnesses. Can we support different thresholds based on type of alleger?

Identity Management. To enable real-world identities, allegation escrows require a robust public-key infrastucture (PKI), for which users should validate their identity with a trusted authority. If a user registers immediately before filing an allegation, then this act reveals an intent to file an allegation. Hence the PKI must be established beforehand. SAE requires a pre-registration step in addition to PKI, which is acceptable in many cases since a new PKI must be established anyway: most organizations either don't have one for their employees/students, or the ones they have aren't very robust. For instance, some administrators may have access to employee logins/emails. Nevertheless, it is interesting future work to explore how to effectively exploit a pre-existing PKI to avoid a separate pre-registration for the escrow system. For instance, if each person had an identity certificate, they could secret share their identity and prove in zero knowledge that they own the secret-shared identity. Prior work demonstrates how to do this while being backwards compatibile with X.509 certificates [17] and email services [41]. Future work must establish that this is practical and efficient. Additionally, many organizations use multi-factor authentication. Exploiting this additional layer of security is also interesting future work.

VIII. Conclusion

We have presented SAE, a robust allegation escrow system with strong cryptographic security guarantees. SAE keeps allegations and the identities of allegers and the accused confidential until alleger-specified match-thresholds are reached. The system's security and privacy guarantees provably hold as long as a majority of the escrow parties are uncorrupted. Our empirical evaluation suggests that SAE is efficient enough to be used in practice, and scales well to large numbers of users and allegations.

Acknowledgements

We thank the anonymous reviewers. We are also grateful to Krishna Gummadi, Prabhanjan Ananth, Hari Balakrishnan, Derek Leung, Omer Paneth, Malte Schwarzkopf, Frank Wang and Nickolai Zeldovich for many interesting discussions and valuable feedback on the paper.

References

[1] "Me Too movement," https://metoomvmt.org/, 2017, accessed: 2020-01-27.

[2] "Project callisto 2016-2017 school year report," https://www.projectcallisto.org/Callisto_Year_2_highres.pdf, 2017, accessed: 2020-01-27.

[3] "Callisto: Tech to combat sexual assault," http://projectcallisto.org, 2019, accessed: 2020-01-27.

[4] I. Ayres and C. Unkovic, "Information escrows," *Michigan Law Review*, pp. 145–196, 2012.

[5] M. Backes, I. Goldberg, A. Kate, and E. Mohammadi, "Provably secure and practical onion routing," in *2012 IEEE 25th Computer Security Foundations Symposium*, 2012, pp. 369–385.

[6] M. Backes, A. Kate, and A. Patra, "Computational verifiable secret sharing revisited," in *Advances in Cryptology - ASIACRYPT 2011 Proceedings*, 2011, pp. 590–609.

[7] M. Ben-Or, S. Goldwasser, and A. Wigderson, "Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation," in *Proc. 20th Annual ACM Symposium on Theory of Computing (STOC)*, 1988, pp. 1–10.

[8] D. Boneh, K. Lewi, H. Montgomery, and A. Raghunathan, "Key homomorphic prfs and their applications," in *Advances in Cryptology–CRYPTO 2013*. Springer, 2013, pp. 410–428.

[9] C. Cachin, K. Kursawe, and V. Shoup, "Random oracles in constantinople: Practical asynchronous byzantine agreement using cryptography," *Journal of Cryptology*, vol. 18, no. 3, pp. 219–246, 2005.

[10] R. Canetti, "Universally composable security: A new paradigm for cryptographic protocols," Cryptology ePrint Archive, Report 2000/067 (Revision July 2013), https://eprint.iacr.org/2000/067/20130717:020004.

[11] N. Chandran, V. Goyal, R. Ostrovsky, and A. Sahai, "Covert multi-party computation," in *48th Annual IEEE Symposium on Foundations of Computer Science (FOCS), 2007*, pp. 238–248.

[12] D. Chaum, C. Crépeau, and I. Damgård, "Multiparty Unconditionally Secure Protocols," in *Proc. 20th Annual ACM Symposium on Theory of Computing (STOC)*. ACM Press, 1988, pp. 11–19.

[13] B. Chor, S. Goldwasser, S. Micali, and B. Awerbuch, "Verifiable Secret Sharing and Achieving Simultaneity in the Presence of Faults," in *IEEE Foundations of Computer Science 1985 (FOCS)*, pp. 383–395.

[14] T. Chou and C. Orlandi, "The simplest protocol for oblivious transfer," in *LATINCRYPT*, 2015, pp. 40–58.

[15] I. Damgård, V. Pastro, N. P. Smart, and S. Zakarias, "Multiparty computation from somewhat homomorphic encryption," in *Advances in Cryptology - CRYPTO 2012*, 2012, pp. 643–662.

[16] A. De Caro and V. Iovino, "jpbc: Java pairing based cryptography," in *Proceedings of the 16th IEEE Symposium on Computers and Communications, ISCC 2011*, pp. 850–855. [Online]. Available: http://gas.dia.unisa.it/projects/jpbc/

[17] A. Delignat-Lavaud, C. Fournet, M. Kohlweiss, and B. Parno, "Cinderella: Turning shabby x. 509 certificates into elegant anonymous credentials with the magic of verifiable computation," in *2016 IEEE Symposium on Security and Privacy (SP)*, pp. 235–254.

[18] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: The second-generation onion router," DTIC Document, Tech. Rep., 2004.

[19] Y. Dodis and A. Yampolskiy, "A verifiable random function with short proofs and keys," in *International Workshop on Public Key Cryptography*. Springer, 2005, pp. 416–431.

[20] P. Feldman, "A practical scheme for non-interactive verifiable secret sharing," in *IEEE 28th Annual Symposium on Foundations of Computer Science (FOCS), 1987*, pp. 427–437.

[21] O. S. Foundation, "Openssl," https://openssl.org, 2019, version: 1.0.2.

[22] S. D. Galbraith, K. G. Paterson, and N. P. Smart, "Pairings for cryptographers," *Discrete Applied Mathematics*, vol. 156, no. 16, pp. 3113–3121, 2008.

[23] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin, "Secure distributed key generation for discrete-log based cryptosystems," *J. Cryptology*, vol. 20, no. 1, pp. 51–83.

[24] R. Gennaro, M. O. Rabin, and T. Rabin, "Simplified vss and fast-track multiparty computations with applications to threshold cryptography," in *Proceedings of the seventeenth annual ACM symposium on Principles of Distributed Computing*, 1998, pp. 101–111.

[25] O. Goldreich, S. Micali, and A. Wigderson, "How to Play any Mental Game or A Completeness Theorem for Protocols with Honest Majority," in *Proc. 19th Annual ACM Symposium on Theory of Computing (STOC)*, 1987, pp. 218–229.

[26] D. Harnik, P. Ta-Shma, and E. Tsfadia, "It takes two to #metoo-using enclaves to build autonomous trusted systems," *arXiv preprint arXiv:1808.02708*, 2018.

[27] S. Hemminger, F. Ludovici, and P. P. Hagen, "Linux manpage tc-netem," man7.org/linux/man-pages/man8/tc-netem.8.html.

[28] Y. Ishai, R. Ostrovsky, and V. Zikas, "Secure multi-party computation with identifiable abort," in *Advances in Cryptology - CRYPTO 2014*, 2014, pp. 369–386.

[29] A. Joux, “The Weil and Tate Pairings as Building Blocks for Public Key Cryptosystems,” in *ANTS-V*, 2002, pp. 20–32.

[30] A. Kate and I. Goldberg, “Distributed key generation for the internet,” in *2009 29th IEEE International Conference on Distributed Computing Systems*, 2009, pp. 119–128.

[31] J. Katz, U. Maurer, B. Tackmann, and V. Zikas, “Universally composable synchronous computation,” in *Theory of Cryptography - 10th Theory of Cryptography Conference, TCC, 2013. Proceedings*, pp. 477–498.

[32] B. Kuykendall, H. Krawczyk, and T. Rabin, “Cryptography for #metoo,” *Proceedings on Privacy Enhancing Technologies*, vol. 2019, no. 3, pp. 409–429, 2019.

[33] A. N. Kuznetsov and B. Hubert, “Linux manpage tc-tbf,” http://man7.org/linux/man-pages/man8/tc-tbf.8.html.

[34] M. Naor, B. Pinkas, and O. Reingold, “Distributed pseudo-random functions and kdcs,” in *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 1999, pp. 327–346.

[35] T. P. Pedersen, “Non-interactive and information-theoretic secure verifiable secret sharing,” in *Annual International Cryptology Conference*. Springer, 1991, pp. 129–140.

[36] A. Rajan, L. Qin, D. Archer, D. Boneh, and T. L. M. Varia, “Callisto: A cryptographic approach to detect serial predators of sexual misconduct,” 2018, online at: https://www.projectcallisto.org/callisto-cryptographic-approach.pdf.

[37] A. Rajan, L. Qin, D. W. Archer, D. Boneh, T. Lepoint, and M. Varia, “Callisto: A cryptographic approach to detecting serial perpetrators of sexual misconduct,” in *ACM COMPASS*, 2018, pp. 49:1–49:4.

[38] A. Shamir, “How to share a secret,” *Communications of the ACM*, vol. 22, no. 11, pp. 612–613, 1979.

[39] B.-I. University, “SCAPI - Secure Computation API,” https://crypto.biu.ac.il/scapi/, 2019, 2.3.

[40] L. Von Ahn, N. Hopper, and J. Langford, “Covert two-party computation,” in *Proceedings of the thirty-seventh annual ACM symposium on Theory of computing*, 2005, pp. 513–522.

[41] L. Wang, G. Asharov, R. Pass, T. Ristenpart, and A. Shelat, “Blind certificate authorities,” in *2019 IEEE Symposium on Security and Privacy (SP)*, pp. 1015–1032.

[42] D. Wikström, “A universally composable mix-net,” in *Theory of Cryptography Conference*, M. Naor, Ed., 2004.

[43] A. C.-C. Yao, “Protocols for Secure Computations (Extended Abstract),” in *Proc. 23rd IEEE Symposium on Foundations of Computer Science (FOCS)*, 1982, pp. 160–164.

Appendix

A. Verifiable Pseudorandom Functions (VRFs)

VRFs cannot be distinguished from a random function by a computationally bounded adversary that does not have access to the proof. For our purposes, we adopt the following formal definition of a VRF from [19]. Let $a_1:\mathbb{N}\rightarrow\mathbb{N}\cup\{*\}$ and $a_2:\mathbb{N}\rightarrow\mathbb{N}$ be functions computable in $\text{poly}(k)$ time[5]. $F_{(\cdot)}(\cdot):\{0,1\}^{a(\lambda)}\rightarrow\{0,1\}^{b(\lambda)}$ is a family of VRFs if there exists a PPT (probabilistic polynomial time computable) algorithm GEN and deterministic algorithms $PROVE$ and VER such that $GEN(1^\lambda)$ outputs a pair of keys (SK,PK); $PROVE_{SK}(x)$ computes $(F_{SK}(x),\pi_{SK}(x))$, where $\pi_{SK}(x)$ is a proof of correctness; and $VER_{PK}(x,y,\pi)$ verifies that $y=F_{SK}(x)$. They satisfy the following properties: 1) *Uniqueness:* No values $(PK,\ x,\ y_1,\ y_2,\ \pi_1,\ \pi_2)$ satisfy $VER_{PK}(x,y_1,\pi_1)=1=VER_{PK}(x,y_2,\pi_2)$ when $y_1\neq y_2$, 2) *Provability:* If $(y,\pi)=PROVE_{SK}(x)$, then $VER_{PK}(x,y,\pi)=1$ and, 3) *Pseudorandomness:* For any PPT algorithm $A=(A_1,A_2)$ that does not query its oracle on x, the following holds:

$$Pr\left[b=b'\left|\begin{array}{l}(SK,PK)\leftarrow GEN(1^\lambda),\\(x,st)\leftarrow A_1^{PROVE(\cdot)}(PK),\\y_0\leftarrow F_{SK}(x),y_1\leftarrow\{0,1\}^{a_2(\lambda)},\\b\leftarrow\{0,1\},b'\leftarrow A_2^{PROVE(\cdot)}(y_b,st)\end{array}\right.\right]\leq\frac{1}{2}+\eta(\lambda)$$

where $\eta(\cdot)$ is a negligible function. Further, it satisfies the following unpredictability property.

For any PPT algorithm A, who does not query its oracle on x, the following holds:

$$Pr\left[y=F_{SK}(x)\left|\begin{array}{l}(PK,SK)\leftarrow GEN(1^\lambda);\\(x,y)\leftarrow A^{PROVE(\cdot)}(PK)\end{array}\right.\right]\leq\eta(k)$$

B. Postponed Security Analysis

Definition 2. *[14]. A symmetric encryption scheme (E,D) is non committing if there exist two PPT algorithms $(\mathcal{A}_1,\mathcal{A}_2)$ s.t. (c,k) and (c',k') are computationally indistinguishable when $c'\leftarrow\mathcal{A}_1(1^\lambda)$, $k'\leftarrow\mathcal{A}_2(c',M)$, $k\leftarrow\mathcal{K}$ and $c\leftarrow E(k,M)$ for all $M\in\mathcal{M}$ where $\mathcal{K},\mathcal{M},\mathcal{C}$ denote key, message and ciphertext spaces respectively*

We refer [14] for a simple construction.

Proof Sketch for Proof Theorem 2: Our proof strategy consists of the description of a simulator $\mathcal{S}$ that handles users corrupted by the attacker and simulates the real world execution protocol while interacting with the ideal functionality $\mathcal{F}_{SAE}$.

The simulator $\mathcal{S}$ spawns honest users at adversarial will and impersonates them until the environment $\mathcal{E}$ makes a corruption query on one of the users: At this point $\mathcal{S}$ hands over to $\mathcal{A}$ the internal state of the target user and routes all of the subsequent communications to $\mathcal{A}$, who can reply arbitrarily. For operations exclusively among corrupted users, the environment does not expect any interaction with the simulator. Similarly, interactions exclusively among honest nodes happen through secure channels and therefore the attacker does not gather any additional information other than the fact that the interactions took place. For simplicity, we omit these operations in the description of our simulator. The simulator simulates the following honest nodes: 1) the honest escrows, 2) the honest users, 3) the CA for users' real identities. Next, we describe how the simulator behaves at various points of the protocol.

At several points in the SAE protocol, DKG is required. namely, SK_I used to compute MACs on identities, SK_R used for revealing alleger identity and SK_i for each i^{th} bucket used for thresholding. To simulate this with a minority of statically corrupted escrows, $\mathcal{S}$ chooses a random key pair, performs DKG simulation [23, Theorem 1], and sends the the public key to the corrupted escrows. As this simulation is exactly the distribution in the real protocol [23, Theorem 1], and hence is indistinguishable from it. Notice that the simulator knows all the DKG secret keys here. It participates in computing PK_I from SK_I. The simulator also generates the public-private key pairs for all the honest users and generates certificates for them from the CA.

For allegation filing and registration, we consider two cases depending on whether or not the alleger is honest.

Case 1: Honest alleger, corrupted minority of escrows

When an honest alleger registers, $\mathcal{F}_{SAE}$ sends (“Register”,ID) to the the simulator. The simulator proves the honest alleger's identity

[5] Except when a_1 takes the value $*$, which means the VRF is defined for inputs of all length.

to the corrupted escrows. This is possible because it simulates the CA and can generate arbitrary certificates. Then it generates l new public keys $pk_1,...,pk_l$ (note, figure 2 shows only $l=1$ for notational simplicity) and secret shares them among the escrows and participates in the distributed computation of $\pi_{SK_I}(H_2(pk_i))$ and $F_{SK_R}(H_2(pk_i))$ as described in§III-D (note, the simulator knows SK_I and SK_R). If the adversary refuses to participate in this computation, the simulator sends $\perp$ to $\mathcal{F}_{SAE}$ from a corrupted escrow's channel and aborts. Else it sends `OK`. As in the real protocol, the adversary obtains $F_{SK_R}(pk_i)$, but not $\pi_{SK_I}(pk_i)$. So far, this is exactly what happens in the real protocol, except that DKG and the honest parties' private keys are chosen by the simulator, but from the same distribution. Hence it is indistinguishable from the real execution.

When an honest alleger files an allegation, $\mathcal{F}_{SAE}$ sends (“File”,UID,t) to the simulator. The simulator chooses a random key-pair $(sk,pk) \leftarrow \text{Gen}(1^\lambda)$, generates a MAC $\pi_{SK_I}(H_2(pk))$ on it and sends (“File”,pk,t) and $(\pi_{SK_I}(H_2(pk)),\mathcal{C})$ to the corrupted escrows signed using sk, where the $\mathcal{C}$ is a random non-committing encryption ciphertext. The simulator generates a random meta-data $m = H_1(meta-data)$ and symmetric key k, and distributes a minority of shares among the corrupted escrows as $VSS(m)$ and $VSS(k)$, signed with sk. The distribution of meta-data doesn't matter since it is information theoretically hidden from the adversary. Since the adversary has not seen the honest alleger's public key before, the simulator can choose a random one. $\mathcal{F}_{SAE}$ now moves to matching and thresholding, returning $(i$, UID, $\textsc{Unique}((i,m)))$ each time an allegation identified by UID is added to bucket i. Let pk be the public key the simulator chose for UID (in the dishonest alleger case discussed below, the adversary provides pk, corresponding to which $\mathcal{F}_{SAE}$ provides UID).

At this point, the real protocol would be computing $F_{SK_i}(\texttt{allegations}[pk].m)$. The simulator can control the value of this result. If $\textsc{Unique}((i,m))$ matches any other allegations in bucket i, the simulator produces the value it previously returned for that allegation in bucket i. Else it produces a fresh random value. This works because H_1 is collision resistant, $H_1(m)=H_1(m')$ iff $m = m'$ for a computationally bounded adversary. Since F is a PRF, the adversary cannot distinguish between its output and truly random numbers. Note all matching allegations have the same (hash of) meta-data m by definition. If at any point, the adversary refuses to cooperate in distributed-input DPRF computation, the protocol is aborted, and the simulator sends $\perp$ to $\mathcal{F}_{SAE}$, which also halts execution. Else it sends OK each time to move the protocol forward.

To reveal identity in the real protocol, the escrows compute $F_{SK_R}(H_2(pk))$, where pk was the public key used during allegation. To simulate this, the simulator picks pk_i randomly from the set of unrevealed public keys it chose when `ID` was registered. It simulates the other escrows' behavior such that, if the adversary cooperates, it gets $F_{SK_R}(H_2(pk_i))$. Note, the simulator knows SK_R. The simulator sends shares of the (non-committing) symmetric encryption key from honest escrows such that the ciphertext C open to a to the corrupted escrows. Allegation reveal now succeeds.

Case 2: Corrupted alleger, corrupted minority of escrows

During registration, the adversary provides a proof of ID from a CA to the simulator. It also sends the honest escrows' shares of hashes of l public keys $H_2(pk_1),\ldots,H_2(pk_l)$ to the simulator. If the proof of ID is invalid, or the shares are incorrect (i.e. VSS verification fails), the simulator sends $\perp$ to adversary. Else, it sends (“Register”, ID) to $\mathcal{F}_{SAE}$ from the corrupted allegers' ID. Note, the simulator has a majority of shares of $H_2(pk_i)$ and can hence reconstruct them. It also knows the secret keys SK_I and SK_R. Hence it can participate in the computation of $\pi_{SK_I}(H_2(pk_i))$ and $F_{SK_R}(H_2(pk_i))$ on the l public keys to produce the correct result. If the adversary refuses to participate in the computation, it sends $\perp$ to $\mathcal{F}_{SAE}$.

When filing an allegation, the alleger sends $(t,pk,\pi_{SK_I}(H_2(pk)),Enc_k(a))$ to the simulator for broadcasting. It secret shares the key k and a collision-resistant hash of the meta-data, m. The simulator verifies that pk has not been used before and verifies the MAC on it. If the check fails, the simulator sends $\perp$ from the honest escrows to the corrupted alleger. If verification succeeds, the simulator determines the `ID` with which pk was registered (since it has all the registered keys), and connects to $\mathcal{F}_{SAE}$ from `ID`'s channel. It then invokes registration with $\mathcal{F}_{SAE}$ with (File,m,a,t), which responds with (File,C,t)[6]. Now the bucketing algorithm takes place, the simulation process for which is identical to the honest alleger case. $\mathcal{F}_{SAE}$ returns matching allegations for various buckets, and we simulate for the corrupted escrows, a pseudo-random function on the meta-data. This is possible since we know, for the relevant buckets, meta-data of which allegations match.

When an allegation filed by a corrupted party is to be revealed, $\mathcal{F}_{SAE}$ sends (“Reveal”, C, t) to the simulator. The simulator cooperates in computing $\pi_{SK_R}(pk)$, where pk is the corresponding key used to file the allegation identified by C. If the adversary refuses to cooperate, the simulator sends $\perp$ to $\mathcal{F}_{SAE}$. Else, it sends `OK`, and cooperates to reveal a, which it knows. ■

[6] The simulator knows (m,a) since it has a majority of the necessary shares. Again, if the shares are invalid, it sends $\perp$ to the adversary as verifiable secret-sharing is used.