Shield: Secure Allegation Escrow System with Stronger Guarantees

Nishat Koti, Varsha Bhat Kukkala, Arpita Patra, Bhavish Raj Gopal
kotis,varshak,arpita,bhavishraj@iisc.ac.in
Indian Institute of Science
Bangalore, India

ABSTRACT

The rising issues of harassment, exploitation, corruption, and other forms of abuse have led victims to seek comfort by acting in unison against common perpetrators (e.g., #MeToo movement). One way to curb these issues is to install allegation escrow systems that allow victims to report such incidents. The escrows are responsible for identifying victims of a common perpetrator and taking the necessary action to bring justice to them. However, users hesitate to participate in these systems due to the fear of such sensitive reports being leaked to perpetrators, who may further misuse them. Thus, to increase trust in the system, cryptographic solutions are being designed to realize secure allegation escrow (SAE) systems.

In the work of Arun et al. (NDSS'20), which presents the state-of-the-art solution, we identify attacks that can leak sensitive information and compromise victim privacy. We also report issues present in prior works that were left unidentified. To arrest all these breaches, we put forth an SAE system that prevents the identified attacks and retains the salient features from all prior works. The cryptographic technique of secure multi-party computation (MPC) serves as the primary underlying tool in designing our system. At the heart of our system lies a new duplicity check protocol and an improved matching protocol. We also provide additional features such as allegation modification and deletion, which were absent in the state of the art. To demonstrate feasibility, we benchmark the proposed system with state-of-the-art MPC protocols and report the cost of processing an allegation. Different settings that affect system performance are analyzed, and the reported values showcase the practicality of our solution.

KEYWORDS

secure allegation escrows, sensitive data processing, secure multi-party computation, privacy-preserving computation

1 INTRODUCTION

Mechanisms such as stringent policies, improved norms and standards, installation of allegation escrows, etc., are being put in place to deter crimes such as abuse, corruption, exploitation, and harassment. For instance, institutions are mandated to appoint an organizational ombudsperson or a Chief Vigilance Officer (CVO) responsible for the prevention, detection, and punishment for malpractices. The victims are expected to report the inflicted crime to the CVO. Since the report contains identities of the accused and victim, details of the inflicted crime, etc., it is regarded to be highly sensitive. The profound harm that can be inflicted on victims if the CVO leaks this sensitive data to the perpetrator, which is likely when the latter is a person of influence, instills great fear in victims and prevents many from coming forward. Thus, such a system requires the victims to place enormous trust in the integrity of the CVO. Instead, a secure platform for reporting crimes is a more reliable solution. Further, the victims are comfortable reporting the crime to a digital platform rather than to a human counterpart [24]. Additionally, such a platform is more accessible and scalable. Thus, our work aims to design a secure allegation escrow system that empowers victims to securely report allegations and seek justice.

Desirable properties of secure allegation escrow. Having a system that merely records allegations and reveals them to the concerned authorities (for further action) may not suffice. Instead, the system should reveal allegations to the concerned authorities only when a sufficient number of allegations are recorded against a common perpetrator. This is because victims often find it effective and comforting to come out as a group. Further, acting against a common perpetrator in unison reduces the fear of retribution discussed previously. Some noteworthy examples of acting in unison are the #MeToo movement [1], and Project Callisto [37] which was deployed to help report sexual assaults on university campuses. To facilitate the reporting and processing of *collective* allegations, an allegation escrow system should have the following properties– (i) each victim must be able to independently file an allegation against a perpetrator, (ii) the system must be capable of matching allegations filed against a common perpetrator, (iii) these matched allegations should be revealed to the concerned authorities only once a predetermined condition for disclosure is met (e.g., Project Callisto requires at least two allegations against the same perpetrator before these can be revealed), (iv) the identity of the accuser, accused, and the details of the allegation must remain hidden until the allegation is revealed as a part of a collection. Additionally, a centralized solution (i.e., one escrow) for the same is a misfit since it forms a single point of failure. Hence, similar to the CVO-based solution, one may compromise the escrow and learn the sensitive allegation data. Thus, it is desirable to have several independent escrows which collectively effectuate a secure allegation escrow (SAE) system with the above-mentioned properties and guarantee that *none* of the escrows can individually learn allegations on clear.

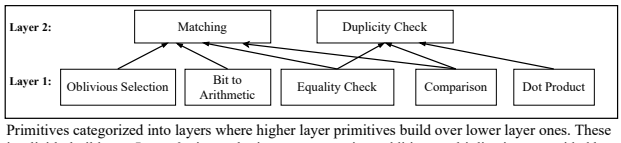
The condition for disclosure is one of the most crucial features of an SAE. It defines the system's sensitivity towards handling an alleger's discomfort. This condition is calibrated using a parameter called *reveal threshold*. The parameter captures the minimum size of the unison the alleger wishes to be a part of (excluding the alleger) when its allegation is revealed in clear to the concerned authorities. In the literature, the reveal threshold has evolved from being a parameter that is globally fixed (i.e., common to all allegers) and public (i.e., known on clear to all the escrows) to an alleger-defined (variable) public parameter. Project Callisto [37] uses a globally-fixed public reveal threshold of one. The work of [28] extends support

for a reveal threshold of more than one, yet it is globally fixed and public as before. However, *not every* alleger may be comfortable in coming out against a common perpetrator with just one other alleger (or even a system-defined threshold number of allegers). That is, setting a low (high) system-defined threshold will not allow participation of victims who prefer more (few) supporters, making the sytem non-inclusive. The work of [6] that forms state of the art recognizes this pressing requirement and allows an alleger the flexibility of deciding its reveal threshold. Elaborately, each alleger can decide a reveal threshold, t, for its allegation, which indicates that the allegation can be revealed if there exist at least t other *matching* allegations (i.e., those that allege the same perpetrator) which can be revealed. Thus, a subset $\mathcal{S}$ of matching allegations can be revealed if and only if the threshold of each allegation in $\mathcal{S}$ is $< |\mathcal{S}|$ (size of $\mathcal{S}$). This is referred to as the *reveal criteria* of the set $\mathcal{S}$. For example, if there exists a set of matching allegations with reveal thresholds 2, 3, 3, 4, then no allegation is revealed because there does not exist any subset $\mathcal{S}$ of allegations that satisfies the reveal criteria. However, if another matching allegation with a reveal threshold 3 is filed in the system, all the allegations with thresholds 2, 3, 3, 3, 4 can be revealed. The system must thus allow secure identification of such a set of revealable matching allegations. We note that allowing a variable threshold is not the end of the road. Although [6] provides this key feature, it fails to do so while guaranteeing complete privacy to the victims. We explain our concern with one example below. Consider the scenario described above when a system has matching allegations with reveal thresholds 2, 3, 3, 4, none of which can be revealed. Observe that if an alleger among these files another copy of its allegation, [6] treats the copy as a new allegation. Thus, these set of 5 allegations will satisfy the reveal criteria despite having an insufficient number of *distinct* allegers. Note that this results in prematurely revealing the genuine (unique) allegations and compromises the privacy of the allegers. Similarly, there arise other privacy issues owing to the reveal threshold being public, which are detailed later (§3). Since the user-defined reveal threshold captures the vulnerability of an alleger, it must be regarded as highly sensitive information. Thus, in this work, we develop the *first* SAE system that offers not only a flexible user-defined threshold, but also guarantees to keep thresholds private, and thereby arrests all concerns raised above. Additionally, we consider the possibility of allegation modification and deletion.

Privacy-preserving tool for realizing SAE. To ensure the desired level of privacy for an SAE system, we rely on the technique of secure multi-party computation (MPC). At a high level, MPC allows n mutually distrusting parties to carry out computations on private inputs such that an adversary controlling up to t parties does not learn anything other than the output of the computation. Many interesting applications have proposed the use of MPC to enable privacy which include tax fraud detection by the Estonian Tax and Customs Board [8], secure sugar beet auctions for Danish farmers [10], secure aggregation tool for Boston wage inequity [7], financial data analysis [9], privacy-preserving machine learning [15, 18, 26, 31, 33, 36], to name a few. Concretely, the current work entails designing an MPC-based SAE system, which is realized via a set of (untrusted) escrows (acting as parties inside the MPC) who carry out the necessary computations of SAE via an MPC protocol on the submitted allegations. MPC guarantees privacy of computation so that nothing beyond allowed outcomes of SAE system (a bunch of matched allegations when reveal criteria is met) is leaked.

1.1 Our contributions

We identify shortcomings in the prior systems. This also includes identifying attacks on the state-of-the-art system of [6], that can compromise a victim's privacy. To address these privacy breaches, we design a secure allegation escrow system called Shield, while retaining the salient features from prior works. The features provided by Shield, in comparison to the prior works, appear in Table 1. As evident from the table, [6] focuses on providing an $O(1)$ complexity solution which comes at the expense of privacy. On the contrary, we prioritize user *privacy* over system *efficiency* since privacy is essential for an SAE system. Hence, our protocol aims at achieving as efficient a solution as possible while guaranteeing *no* privacy breach. A new replacement to the secure matching protocol that identifies a revealable set of matching allegations and the inclusion of a new duplicity check protocol that prevents users from filing duplicates lies at the heart of Shield. The challenge in designing the matching protocol lies in handling a user-defined and private reveal threshold. The subtlety in designing the duplicity check is in ensuring that a genuine alleger is allowed to complain against multiple perpetrators if required. This should be done while simultaneously guaranteeing that a corrupt alleger cannot forge the identity of another honest alleger to file duplicates (unless these two allegers collude). For completeness, we additionally provide features such as allegation modification and deletion, which were absent in the state of the art. Note that these can optionally be included in the system, depending on the deployment scenario. We resort to a modular approach to design the protocols, as shown in Fig. 1, by identifying the MPC building blocks that would be required and their interdependence. These building blocks have been extensively used in realizing privacy-preserving machine learning (PPML) [15, 17, 18, 26, 31, 33, 36]. Importantly, we allow Shield to make black-box use of the MPC building blocks. This not only allows Shield to inherit the latter's security guarantees and efficiency, but also opens up the possibility of utilizing the future advancements of MPC in a seamless way. We focus on benchmarking the complexity of allegation processing in our system and report the overhead involved in the enhancement. We instantiate the MPC using state-of-the-art 3-party computation (3PC) and 4-party computation (4PC) frameworks of SWIFT [26] and Tetrad [27], respectively, from which we obtain layer 0 and layer 1 primitives. Finally, we elaborately discuss on the design choices that such a system should incorporate when attempting to achieve an ideal solution.



Primitives categorized into layers where higher layer primitives build over lower layer ones. These implicitly build over Layer 0 - input sharing, reconstruction, addition, multiplication - provided by underlying MPC.

Figure 1: Hierarchy of primitives

Protocol	*Flexible* reveal threshold	*Private* reveal threshold	Duplicity complexity	Matching complexity
[37]	✘	✘	NA*	$O(\mathsf{N})^{\dagger}$
[28]	✘	✘	$O(\mathsf{N})$	$O(\mathsf{N} \cdot q)$
[6]	✔	✘	✘	$O(1)$
Ours	✔	✔	$O(\mathsf{N})$	$O(\mathsf{N} \cdot \mathsf{maxths})$

q: fixed reveal threshold, maxths: upper bound on flexible reveal threshold, N: number of allegations in the system.
*Duplicity check is not applicable here. Filing a duplicate allegation, to prematurely reveal a genuine one, requires a threshold of at least 2 as opposed to 1 in Callisto.
†Due to missing details in Callisto, the complexity reported assumes requirement of a linear scan to identify matching allegations.

Table 1: Comparison of SAE protocols

Organization. Prior systems, their shortcomings and specific attacks on [6] appear in §3. The threat model, design of Shield, followed by the duplicity check and matching protocols appear in §4. Additional features and discussions appear in §5 and §6, respectively. Our benchmarks appear in §7.

2 PRELIMINARIES

Threat model. We design Shield to comprise n escrows (e.g., computationally powerful hired servers) $\mathcal{P} = \{P_1, P_2, \ldots, P_n\}$ that are connected via pairwise private and authentic channels in a synchronous network. These escrows enact the role of parties in the underlying MPC protocol. We assume a static, malicious probabilistic polynomial time (PPT) adversary that can corrupt up to $t < n$ escrows and arbitrarily deviate from protocol specification. Note that n is a tunable parameter, and a larger n allows tolerating a higher number of corruptions, thereby increasing the trust in the system. We assume that an arbitrary number of system users may collude with the maliciously corrupt escrows. Security of protocols is guaranteed in the real-world/ideal-world paradigm [22, 30] (see §B).

Secret sharing. To ensure privacy, MPC relies on *secret sharing* [25, 38] to distribute the input among the computing parties. Let $\mathbb{G}$ denote a finite algebraic structure such as ring or field[1]. Secret sharing enables distributing a secret $\mathsf{v} \in \mathbb{G}$ among a set of parties $\mathcal{P}$, such that $P_i \in \mathcal{P}$ holds a share $[\![\mathsf{v}]\!]_i \in \mathbb{G}$ of v where $[\![\mathsf{v}]\!]_i$ reveals no information about v. However, if more than t parties combine their shares, they can retrieve (reconstruct) the secret. MPC allows the parties to evaluate the required function on secret-shared inputs such that the intermediate values and the output remain secret-shared. Since Shield operates over arithmetic as well as Boolean values, we use the following notations. $[\![\mathsf{v}]\!]$ denotes *arithmetic* secret sharing of $\mathsf{v} \in \mathbb{G}$, and $[\![\mathsf{v}]\!]^{\mathbf{B}}$ denotes its *Boolean* secret sharing where each bit of v is shared over the Boolean ring $\mathbb{Z}_2$.

3 PRIOR SYSTEMS AND THEIR DRAWBACKS

Callisto provides a first-step solution to guarantee user privacy when filing allegations. However, it can be greatly improved to better cater to user privacy. Callisto does not provide a formal threat model and selectively accounts for misbehaviour from malicious user. It also requires placing trust on the various entities involved. For instance, the DB server in Callisto which stores perpetrator-id (pid) and the encrypted allegation text is not modeled as a trusted entity, even when compromising the same allows an adversary to learn the number of allegations against a specific pid (*probing attack*). Additionally, during allegation modification, the DB server can learn whether pid is modified or if the allegation text was modified. Thus, the DB server cannot be modeled as an untrusted entity. Moreover, Callisto only has an invitation based registration of users and policy based solution to ensure explicit tracking of users (via mapping the real-world identity to the unique-ID provided during registration) is not done. Further, unlike stated in Callisto, trivially extending support to the mentioned features results in privacy issues: (i) support for higher reveal thresholds leads to premature allegation revealing due to duplicates and (ii) support for allegation matching when users are given the flexibility to file using different identifying attributes (name, email-id, phone number, etc.) of a perpetrator leads to unidentified matching allegations due to mismatch in perpetrator attributes. We elaborate the issues in each case next. Since Callisto has a reveal threshold of 1, it does not have to explicitly check for duplicate allegations being filed. However, the check for duplicity is indispensable for higher thresholds. Since Callisto does not hold its users accountable for the filed allegations, the current framework is not equipped to check for duplicates. Similarly, we see possible issues in matching allegations using multiple identifiers for a given perpetrator. Callisto suggests viewing the $\overrightarrow{\text{pid}}$ as a vector issued by the key server, with each component of the vector corresponding to an identifier of the perpetrator, as submitted by the victim. The DB server would then identify a match if the vectors have a common component. However, we would like to note that such a solution can in fact hinder the correctness of the system. The flexibility of having victims query on different identifiers for the same victim could result in two allegations being filed with non-overlapping identifiers. This would result in the DB server failing to match the allegations, despite both being against the same perpetrator.

In an attempt to overcome the issues present in Callisto, the work in [28] relies on the cryptographic technique of MPC. It provides a distributed escrow system referred to as WhoToo for reporting sexual misconduct. This distributed variant of the system ensures that the entities in the system are no longer required to be treated as a single point of trust. The authors in [28] identify the following issues with Callisto–(i) Callisto fails to bind the alleger identity to the allegation (i.e., alleger is not *accountable* for its allegation), which facilitates attacks that prematurely reveal allegations to the LOC, (ii) Callisto is susceptible to probing attack as stated earlier, and (iii) Callisto leaks the user's identity to the key server, each time the former authenticates itself to the latter, in the process of querying for a pid. The authors in [28] propose a solution that specifically addresses these three attacks.

The recent work in [6] not only addresses the issues pointed in [28] but also enhances WhoToo by identifying and addressing the limitations present therein. First, the authors in [6] make a strong argument in favor of empowering the users with the flexibility of having user-defined reveal threshold instead of enforcing a globally predetermined reveal threshold, as in WhoToo. Second, they focus

[1]While MPC in the past has been performed over fields, the recent literature focuses on operating over rings $\mathbb{Z}_{2^\ell}$, due to the efficiency improvements arising from leveraging the computer architecture and avoiding the operator overloading, etc. In fact, all recent works in PPML [15, 26, 31, 33, 35, 36, 39] including SWIFT [26] and Tetrad [27] work over rings.

on providing a computationally more efficient and hence a scalable solution. Unlike the solution in WhoToo that requires $O(q\mathsf{N})$ computations to process a newly filed allegation, the solution presented in [6] requires $O(1)$ computations. Here, q is the globally-fixed public threshold and N denotes the total number of allegations in the system. Additionally, the authors in [6] showcase how the system can be generalized to handle allegations against different types of crimes, rather than limit to sexual misconduct alone. Thus, [6] presents the state-of-the-art solution to realizing a secure allegation escrow system. Similar to WhoToo, [6] distributes the trust among multiple parties via MPC. However, in an attempt to achieve $O(1)$ computational efficiency, it loses out on guaranteeing user privacy. Attacks that breach user privacy are given in §3.1.

Finally, to reduce the trust placed on escrows, the work of [23] uses secure enclaves (built on Intel SGX) as an additional defence mechanism. However, due to the backlash faced by SGX-based solutions, we do not delve into it [14, 29, 40].

3.1 Attacks on [6]

Attack by filing fake allegations. To determine matching allegations, the protocol in [6] only compares allegations that are waiting for the *same* number of additional allegations required to satisfy their public reveal threshold. Whenever a comparison results in a match, such allegations are grouped (known as *collections*) and processed as a single unit from then on. Although the allegation remains hidden, each escrow learns the lifecycle of an allegation, that includes the time of allegation filing, whether a comparison results in a match, grouping of matched allegations, and the number of additional allegations that each filed allegation awaits. We refer an interested reader to §A for the details of the matching and bucketing protocol of [6]. Consider now a scenario where a perpetrator colludes with an escrow and has access to the view of the escrow. Based on the information of when the perpetrator launched the assault and the timing of a filed allegation, it may be suspicious that this allegation is indeed against it. To confirm the suspicion, the perpetrator can file a fake allegation against itself by setting the same (publicly known) threshold as that of the suspected one. The colluding escrow can thus learn if the suspected allegation and the fake allegation are a match. [6] argues that such adversarial behavior would result in leaving a non-repudiable paper trail (i.e. each alleger is held *accountable* for its allegation), and hence an adversary would not take such risks. However, the counterargument is that the adversary will be at risk only when its fake allegation is revealed, which may not always be the case. Given access to information such as the number of users victimized, the timing of the assault, etc., and the view of the escrow, an adversary (perpetrator) can take a well-educated risk and file such fake allegations with the confidence that it will not be revealed. For instance, let an allegation be filed with threshold $\mathsf{t} > 2$ after a perpetrator launched an assault. A suspicious perpetrator can launch the above attack with the confidence that it will not be revealed if it had harmed only a single victim, implying absence of other matching allegations. This makes it disadvantageous for a user to set high thresholds, and thus, the whole purpose of having a system with flexible reveal threshold is lost. To avoid such attacks, we design protocols that keep the reveal threshold private and leak no intermediate information to the escrows. We also bound the highest threshold that can be set, to increase the probability of an allegation being matched and revealed. This also tackles the issue of an adversary trying to overload the system by flooding it with fake allegations with very high threshold that will remain unrevealed if the threshold is unbounded.

Attack by filing duplicate allegations. Recall that a duplicate allegation is one that is filed by the same alleger against the same perpetrator more than once. Since the system allows each user to file multiple allegations, a corrupt user can file duplicate allegations against a targeted perpetrator. Thus, duplicate allegations together with the genuine allegations may form a revealable set, leading to the possibility of prematurely revealing genuine allegations against the same perpetrator (see example in introduction). Clearly, the privacy of an honest user is breached since its allegation may be revealed even when its threshold criteria is not met by other genuine allegations. One may argue that the above attack may be deterred because the system maintains a non-repudiable paper trail, and the filer of duplicates will eventually be penalized. The counterargument, however, is that despite the corrupt user being punished, the damage to a genuine victim is irrevocable. Presence of such an attack lowers trust of genuine victims in the system and may discourage them from using it. Hence, we design duplicity check protocol to prevent these.

4 DESIGN OF Shield

4.1 Shield functionality

In this section, we design an ideal functionality $\mathcal{F}_{\mathsf{Shield}}$ for our Shield system (Fig. 2) that follows on similar lines to [6]. For the ease of readability, we describe the functionality for a *robust* system here. At a high-level, $\mathcal{F}_{\mathsf{Shield}}$ aims to achieve *alleger's anonymity and allegation secrecy*. That is, an alleger's identity and its allegation should remain hidden from the escrows until the allegation is revealed as a part of a revealable collection. $\mathcal{F}_{\mathsf{Shield}}$ consists of six phases–(i) initialization, (ii) user registration, (iii) allegation filing, (iv) duplicity check, (v) allegation matching and (vi) allegation revealing. Throughout the phases, it maintains a few data structures–R: the registered users of the system; A: all the allegations filed in the system, $\mathcal{S}$: collection of revealable allegations; and P: details of the revealed perpetrators. The i^{th} entry in A is denoted as $\mathsf{a_i}$, which mainly has three attributes pid (perpetrator's id), t (reveal threshold) and Text (crime details). Similarly, the i^{th} entry of P is denoted as p_i and it has two attributes pid (perpetrator's id) and ac (count of allegations revealed against this perpetrator).

In the initialization phase, $\mathcal{F}_{\mathsf{Shield}}$ initializes these data structures to empty lists. During the registration phase, a user sends a request to get registered and the functionality adds the entry to R and notifies the escrows. Here, $\mathcal{F}_{\mathsf{Shield}}$ must ensure that every allegation should be associated with an authentic, real-world identity, which ensures *accountability*. To tie up with a real-world identity, a user must submit a certificate c, obtained earlier from a certification authority (CA), along with its request, which $\mathcal{F}_{\mathsf{Shield}}$ verifies before registering the user. This helps in tracing back an allegation to its alleger in the case of misbehavior, which discourages *fake* allegations.

Additionally, it can be used to discourage *duplicate* allegations. For every registered user, $\mathcal{F}_{\mathsf{Shield}}$ sets the maximum number of allowed allegations to maxalg. This count is decreased every time the user files an allegation in the allegation filling phase.

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

$\mathcal{F}_{\mathsf{Shield}}$ interacts with escrows ($\mathcal{P}$), user, ideal world adversary $\mathcal{S}_{\mathcal{A}}$ and works as follows:

Initialization Initialize empty lists R, A, $\mathcal{S}$ and P.

Registration On receiving a message ("Register", c, ID) from a user with identifier ID, send message ("Registered", c, ID) to all escrows if the certificate c verifies. Include ID in list R and set ID.count = maxalg.

Allegation Filing On receiving a message ("Allege", ID, $\mathsf{a}_{\mathsf{new}}$) from a user ID and allegation $\mathsf{a}_{\mathsf{new}}$, send message ("Failed attempt") to escrows if ID $\notin$ R or if ID.count = 0. Else send message ("Allege"), reduce ID.count by 1, and enter next phase.

Duplicity Check Check if ($\mathsf{a}_{\mathsf{new}}$.pid, ID) is part of some a_i in A. If found, *ignore* and send message ("Duplicate") to escrows. Else, include ($\mathsf{a}_{\mathsf{new}}$,ID) in A, send message ("File allegation") to all escrows, and enter the matching phase.

Matching If $\mathsf{a}_{\mathsf{new}}.\mathsf{pid} = \mathsf{p}_i.\mathsf{pid}$, then set $\mathsf{a}_{\mathsf{new}}.\mathsf{t} = \mathsf{a}_{\mathsf{new}}.\mathsf{t} - \mathsf{p}_i.\mathsf{ac}$. Determine if there exists a revealable subset $\mathcal{S}$ in A. If found, do as follows and continue to the next phase

- Delete each $\mathsf{a}_\mathsf{i} \in \mathcal{S}$ from A and send ("Found", I) to escrows where I denotes indices of allegations in $\mathcal{S}$ with respect to A.
- If $\mathsf{a}_\mathsf{i}.\mathsf{pid} = \mathsf{a}_{\mathsf{new}}.\mathsf{pid}$ and $\mathsf{a}_\mathsf{i} \notin \mathcal{S}$, set $\mathsf{a}_\mathsf{i}.\mathsf{t} = \mathsf{a}_\mathsf{i}.\mathsf{t} - |\mathcal{S}|$.
- If $\mathsf{p}_i.\mathsf{pid} = \mathsf{a}_{\mathsf{new}}.\mathsf{pid}$, set $\mathsf{p}_i.\mathsf{ac} = \mathsf{p}_i.\mathsf{ac} + |\mathcal{S}|$. Else include a new entry ($\mathsf{a}_{\mathsf{new}}.\mathsf{pid}, |\mathcal{S}|$) in P and send message ("New P entry") to the escrows.

Allegation Revealing Reveal allegations $\mathcal{S}$ to the escrows. Reset list $\mathcal{S}$ to be empty.

Figure 2: Ideal functionality for Shield

Next, during allegation filing phase, a user submits a new allegation $\mathsf{a}_{\mathsf{new}}$. If the user is not registered or the user's quota of maximum allowed allegation count is 0, $\mathcal{F}_{\mathsf{Shield}}$ ignores this allegation. Else, it decreases the corresponding user's allowed allegation count by 1. It notifies the escrows of a failed/successful incoming allegation (but nothing beyond) and moves on to the next phase where duplicity of $\mathsf{a}_{\mathsf{new}}$ is checked with respect to the allegations in A. If found to be a non-duplicate, $\mathsf{a}_{\mathsf{new}}$ is added in A (and accordingly, a message is sent to the escrows indicating duplicate/non-duplicate). Next, the matching phase is started. Here, $\mathcal{F}_{\mathsf{Shield}}$ first checks if pid of $\mathsf{a}_{\mathsf{new}}$ matches with any entry p_i of P. If this is true, then $\mathsf{a}_{\mathsf{new}}$'s reveal threshold is reduced by the number of allegations against p_i. This ensures that the alleger of $\mathsf{a}_{\mathsf{new}}$ only has to find this reduced number of supporters from the unrevealed ones in A. Next, $\mathcal{F}_{\mathsf{Shield}}$ finds if there is a revealable subset $\mathcal{S}$ in A. If so, it performs a series of adjustments in the maintained lists before publishing $\mathcal{S}$ in the next phase– (a) the allegations in $\mathcal{S}$ to be revealed is erased from A, (b) it reduces the threshold of each matched, yet not to be revealed, allegation a_i (pid of a_i equals to that of $\mathsf{a}_{\mathsf{new}}$) by $|\mathcal{S}|$, (c) it increases the allegation count of the perpetrator of $\mathsf{a}_{\mathsf{new}}$ in list P or includes an entry for $\mathsf{a}_{\mathsf{new}}$'s pid in P if it was not present earlier in the list. In the reveal phase, $\mathcal{F}_{\mathsf{Shield}}$ reveals $\mathcal{S}$.

4.2 Shield Overview

4.2.1 Primitives used to realize Shield. Here, we elaborate on the primitives relied upon by Shield.

Verifiable pseudorandom function (VRF) [20]. Informally, a VRF is a pseudorandom function $F_{\mathsf{sk_v}}(\cdot)$ along with a proof generation function $\pi_{\mathsf{sk_v}}(\cdot)$ such that a PPT adversary cannot distinguish $F_{\mathsf{sk_v}}(x)$ from an output of a random function without the access to the VRF secret key $\mathsf{sk_v}$ or $\pi_{\mathsf{sk_v}}(x)$. The correct computation of $F_{\mathsf{sk_v}}(x)$ can be verified given a matching public key $\mathsf{pk_v}$ and proof $\pi_{\mathsf{sk_v}}(x)$. Let Π_{vrf} be a protocol that inputs $[\![\mathsf{sk_v}]\!]$ and $[\![\mathsf{x}]\!]$, and outputs $[\![F_{\mathsf{sk_v}}(x)]\!]$ and $[\![\pi_{\mathsf{sk_v}}(x)]\!]$.

Message authentication code (MAC). Informally, a MAC is a function which takes as input a secret key $\mathsf{sk_m}$ and a message x. The output, denoted by mac, can be used to authenticate x, and confirm its origin from the entity holding $\mathsf{sk_m}$. Let Π_{mac} be the protocol that inputs $[\![\mathsf{sk_m}]\!]$, $[\![\mathsf{x}]\!]$, and outputs $[\![\mathsf{mac}]\!]$.

Following [6], we instantiate Π_{vrf}, Π_{mac} using the PRF construction of [20]. For the VRF, we restrict the output to only $\pi_{\mathsf{sk_v}}(x)$ since $F_{\mathsf{sk_v}}(x)$ can be generated using $\pi_{\mathsf{sk_v}}(x)$ and public key $\mathsf{pk_v}$, in our instantiation.

MPC building blocks. The designed protocols rely on MPC building blocks described in Fig. 1. Their description and semantics of the inputs and outputs are provided in Table 2.

Building block	Notation	Description
Comparison	$[\![\mathsf{b}]\!]^{\mathbf{B}} = \Pi_{\mathsf{comp}}([\![\mathsf{x}]\!], [\![\mathsf{y}]\!])$	Outputs b = 1 if x < y, else outputs b = 0
Equality	$[\![\mathsf{b}]\!]^{\mathbf{B}} = \Pi_{\mathsf{eq}}([\![\mathsf{x}]\!], [\![\mathsf{y}]\!])$	Outputs b = 1 if x = y, else outputs b = 0
Oblivious Select	$[\![\mathsf{x_b}]\!] = \Pi_{\mathsf{sel}}([\![\mathsf{x_0}]\!], [\![\mathsf{x_1}]\!], [\![\mathsf{b}]\!]^{\mathbf{B}})$	Obliviously selects $\mathsf{x_b}$ among $\mathsf{x_0}, \mathsf{x_1}$
Bit2A	$[\![\mathsf{b}]\!] = \Pi_{\mathsf{bit2A}}([\![\mathsf{b}]\!]^{\mathbf{B}})$	Converts bit to its arithmetic equivalent
Dot product	$[\![\mathsf{z}]\!]^{\mathbf{B}} = \Pi_{\mathsf{dotp}}([\![\vec{\mathsf{x}}]\!]^{\mathbf{B}}, [\![\vec{\mathsf{y}}]\!]^{\mathbf{B}})$	Computes $\mathsf{z} = \bigoplus_{i=1}^{\mathsf{n}} x_i \wedge y_i$ †

† x_i: i^{th} element of vector $\vec{\mathsf{x}}$; n: vector size; $\bigoplus$: XOR; $\wedge$: AND.

Table 2: Description of building blocks

4.2.2 The Shield *system.* Shield is designed to realize the $\mathcal{F}_{\mathsf{Shield}}$ ideal functionality described above. The designed system continues to have six phases, and the details of the cryptographic tools used to realize each of these phases is described next. These follow similar to [6] and set the stage for our new duplicity check and matching protocol described next.

Escrow initialization. This allows the escrows to establish the necessary setup required for the underlying MPC and for securely processing a filed allegation. Escrows establish authenticated communication links between themselves. They generate $[\![\cdot]\!]$-shares of the secret keys required for a VRF ($[\![\mathsf{sk_v}]\!]$) and a MAC ($[\![\mathsf{sk_m}]\!]$) primitive. The public key ($\mathsf{pk_v}$) of the VRF is, however, known on clear to all the escrows. Use of these primitives during the later phases (registration, filing, and duplicity check) is discussed in place.

User registration. This involves escrows performing initialization with respect to users in preparation for them to securely participate in the system. High-level overview is discussed next.

(i) Escrows must be able to verify and register only valid users of the system. For this, a user contacts the CA, who verifies its

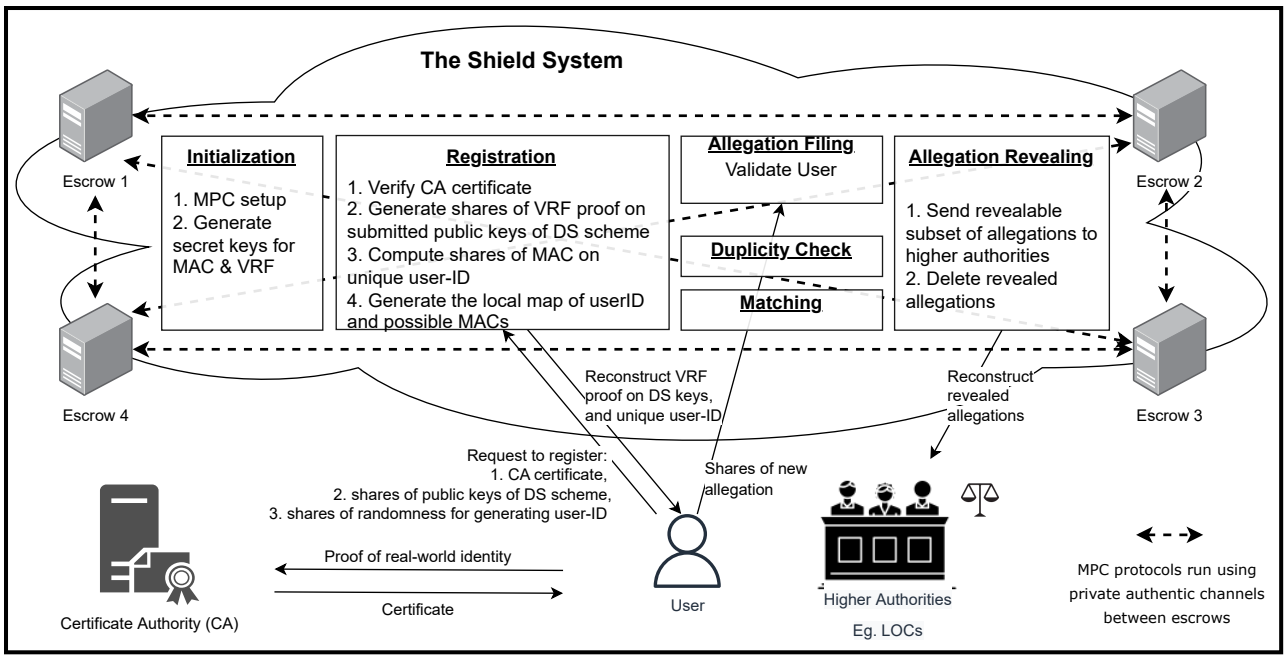


Figure 3: Schematic Diagram of Shield

real-world identity and supplies a certificate, c. Escrows register the user in the system after verifying c.

(ii) To empower only registered users to file an allegation, the system relies on digital signature schemes. To file an allegation later (with accountability), a user U needs to authenticate its allegation using a digital signature (DS) secret key sk^{U} and the escrows verify an allegation's authenticity using the corresponding public verification key pk^{U}. Additionally, the escrows must be able to validate that the used pk^{U} belongs to a registered user. Hence, it is required that each registered user records its verification key with the escrows during registration itself. However, the knowledge of the verification key on clear allows the escrows to link an allegation to a user. To validate a user's allegation without linking to its identity, the escrows issue to the user a VRF proof on its verification key, without learning the latter. This is enabled by having the user secret-share its key, and the escrows (jointly) compute $[\![\pi_{\mathsf{sk}_\mathsf{v}}]\!] = \Pi_{\mathsf{vrf}}\left([\![\mathsf{sk}_\mathsf{v}]\!], [\![\mathsf{pk}^{\mathsf{U}}]\!]\right)$ and reconstruct $\pi_{\mathsf{sk}_\mathsf{v}}$ towards the user. During registration, since the escrows hold pk^{U} and $\pi_{\mathsf{sk}_\mathsf{v}}$ in shared-format, they cannot associate these to the user, even though they know the user identity via CA certificate in the current phase. Hence, when user presents pk^{U} and $\pi_{\mathsf{sk}_\mathsf{v}}$ on clear during allegation filing, user identity remains hidden while the escrows can validate the user. This mechanism also prevents an outsider from faking its registration as it cannot generate valid proof even while colluding with at most t escrows since no subset of them knows sk_v.

Next, we must allow a user to file multiple allegations[2], if required, without the escrows learning that the two allegations have originated from the same user. To tackle this, user U generates maxalg pairs of keys $(\mathsf{pk}_i^{\mathsf{U}}, \mathsf{sk}_i^{\mathsf{U}})$ corresponding to a DS scheme, where a unique pair is to be consumed for each allegation, and $[\![\cdot]\!]$-shares these towards the escrows. The escrows compute VRF proof $[\![\pi^i_{\mathsf{sk}_\mathsf{v}}]\!] = \Pi_{\mathsf{vrf}}\left([\![\mathsf{sk}_\mathsf{v}]\!], [\![\mathsf{pk}_i^{\mathsf{U}}]\!]\right)$ for $i \in \{1, \ldots, \mathsf{maxalg}\}$, which is reconstructed towards the user. Here, maxalg denotes the maximum number of allegations that a user is allowed to file.

(iii) The escrows must be able to identify if the same user is filing a *duplicate* allegation against the same perpetrator ID, pid. For this, the user and the escrows rely on a MAC to generate a unique unforgeable user identity, uid to be given in shares along with each allegation. This is achieved by the user $[\![\cdot]\!]$-sharing a random $\mathsf{r} \in \mathbb{G}$ among the escrows. The escrows compute $[\![\mathsf{uid}]\!] = \Pi_{\mathsf{mac}}([\![\mathsf{sk}_\mathsf{m}]\!], [\![\mathsf{r}]\!])$, and reconstruct it towards the user. Since the escrows hold $\mathsf{sk}_\mathsf{m}, \mathsf{r}$ in shared format, they learn nothing. Further, a user is unable to generate a valid uid for itself or forge another user's uid due to (i) r being unique and secret to each user, and (ii) lack of knowledge about sk_m. Details of how uid facilitates detection of duplicates appears in §4.3. We note that this step is a new addition and is required to detect duplicates.

(iv) The escrows must also be able to learn the user's identity when its submitted allegation needs to be revealed. For this, the escrows rely on the MAC. The escrows compute $[\![\mathsf{mac}_i^{\mathsf{U}}]\!] = \Pi_{\mathsf{mac}}([\![\mathsf{sk}_\mathsf{m}]\!], [\![\mathsf{pk}_i^{\mathsf{U}}]\!])$ and reconstruct it. This allows the escrows to store the association between a user and mac(s) generated on all its DS verification keys in a local map. Looking ahead, during

[2]Multiple allegations should be against different perpetrators. If user allegation is yet to be revealed, it must be disallowed to allege the same perpetrator (duplicate). However, a revealed alleger victimized by the same perpetrator at a later time must be allowed to file the allegation (non-duplicate).

allegation revealing, the escrows recompute the MAC on the DS verification key used in an allegation. The recomputed MAC is matched against entries in the local map to trace the user of the allegation. Observe the duality in the need for anonymization in (ii) and traceability in (iv).

Allegation filing. User U connects to all the escrows via an anonymous communication channel. It files i^{th} new allegation by submitting $\mathsf{pk}_i^{\mathsf{U}}, \pi_{\mathsf{sk_v}}(\mathsf{pk}_i^{\mathsf{U}})$ on clear, and shares of allegation denoted $[\![\mathsf{a_{new}}]\!] = ([\![\mathsf{uid}]\!], [\![\mathsf{r}]\!], [\![\mathsf{pid}]\!], [\![\mathsf{t}]\!], [\![\mathsf{Text}]\!])$, all signed using $\mathsf{sk}_i^{\mathsf{U}}$. That is, j^{th} escrow receives $\mathsf{pk}_i^{\mathsf{U}}, \pi_{\mathsf{sk_v}}(\mathsf{pk}_i^{\mathsf{U}})$ and signed jth share of $\mathsf{a_{new}}$. Escrows check that the submitted $\mathsf{pk}_i^{\mathsf{U}}$ was not used previously, followed by verifying the proof $\pi_{\mathsf{sk_v}}(\mathsf{pk}_i^{\mathsf{U}})$. Upon success, they proceed to verify signature on $\mathsf{a_{new}}$ components using $\mathsf{pk}_i^{\mathsf{U}}$. If any verification fails, escrows ignore $\mathsf{a_{new}}$. Else, escrows proceed to next phase.

Duplicity check. The duplicity check protocol (§4.3) is run by the escrows to discard a new allegation if it is a duplicate.

Allegation matching. This involves running our new matching protocol (§4.4) to identify a revealable set $\mathcal{S}$ of matching allegations.

Allegation revealing. If a revealable set $\mathcal{S}$ is found during matching, this phase reveals $\mathcal{S}$ (together with identity of the allegers) to the escrows[3]. Since the pk^{U} (submitted during allegation filing) associated with allegations in $\mathcal{S}$ is known to the escrows, they recompute $[\![\mathsf{mac}^{\mathsf{U}}]\!] = \Pi_{\mathsf{mac}}([\![\mathsf{sk_m}]\!], [\![\mathsf{pk}^{\mathsf{U}}]\!])$, and reconstruct it. Using the local map of valid mac(s) generated during registration, they can de-anonymize the user.

A schematic representation of Shield appears in Fig. 3, and the security proof of our protocol is deferred to §B.

Theorem 4.1. *Let* VRF *be a secure VRF protocol, let* MAC *be a secure MAC, and let the employed signature scheme be strongly existentially unforgeable. Then our* Shield *protocol realizes* $\mathcal{F}_{\mathsf{Shield}}$ *(Fig. 2) with computational security in the threat model of the underlying MPC.*

For ease of reading the upcoming sections, we request readers to refer to Table 3 which enlists commonly-used notations.

Notation	Description
N	Number of allegations in the system
maxalg	Maximum number of allegations that can be filed by a user
maxths	Upper bound on the reveal threshold
a	Allegation with attributes (uid, r, pid, t, Text)*
a.x	Refers to attribute x of a
A	List of non-duplicate allegations filed in the system: $\{\mathsf{a}_1, \ldots, \mathsf{a}_\mathsf{N}\}$
$\mathsf{P} = \{\mathsf{p}_1, \mathsf{p}_2, ..\}$	List of revealed perpetrators, $\mathsf{p}_i = (\mathsf{pid}, \mathsf{ac})$†
$\mathcal{S}$	Set of revealable allegations
$\mathsf{sk_m}$	Secret key for MAC
$\mathsf{pk_v}, \mathsf{sk_v}$	Public key and secret key for VRF
$(\mathsf{pk}_i^{\mathsf{U}}, \mathsf{sk}_i^{\mathsf{U}})$	i^{th} public key and secret key of user U for digital signature

* uid, r: unique id, pid: perpetrator id; t: reveal threshold, Text: crime description
† pid: perpetrator id; ac: count of allegations revealed against the perpetrator

Table 3: Table of notations

[3]Note that Shield supports an alternative model where the shared $\mathcal{S}$ is directly reconstructed to an external authority (designated to deliver justice) and thus hiding this crucial information from escrows, which may be simply hired for compute-service.

4.3 Duplicity check protocol

To prevent filing of duplicate allegations, two measures are taken. First, every user U is associated with a unique unforgeable user identity (uid) during the registration, which is verified when an allegation is filed. Second, the unique identity and the perpetrator identity of the new allegation is matched with that of the existing allegations. While the latter is the obvious test for duplicity, the former test ensures that a user cannot submit an allegation without registering and impersonate another user[4] due to the unforgeability of the unique identity. The protocol overview is given below.

Protocol $\Pi_{\mathsf{Dup}}(\{\mathsf{a}_1, \ldots, \mathsf{a}_\mathsf{N}\}, \mathsf{a_{new}}, \mathsf{sk_m})$

- $[\![\mathsf{uid}']\!] = \Pi_{\mathsf{mac}}([\![\mathsf{sk_m}]\!], [\![\mathsf{a_{new}.r}]\!])$ and $[\![\mathsf{chk}]\!]^{\mathbf{B}} = \Pi_{\mathsf{eq}}([\![\mathsf{uid}']\!], [\![\mathsf{a_{new}.uid}]\!])$
- Reconstruct chk. If chk = 1, then for each allegation $\mathsf{a_i} \in \mathsf{A}$,
  - $[\![\mathsf{chk}_1^i]\!]^{\mathbf{B}} = \Pi_{\mathsf{eq}}([\![\mathsf{a_i.pid}]\!], [\![\mathsf{a_{new}.pid}]\!])$
  - $[\![\mathsf{chk}_2^i]\!]^{\mathbf{B}} = \Pi_{\mathsf{eq}}([\![\mathsf{a_i.uid}]\!], [\![\mathsf{a_{new}.uid}]\!])$
- Reconstruct $\mathsf{chk}_3 = \Pi_{\mathsf{dotp}}(\{[\![\mathsf{chk}_1^i]\!]^{\mathbf{B}}, [\![\mathsf{chk}_2^i]\!]^{\mathbf{B}}\}_{i=1}^{\mathsf{N}})$.
- If $\mathsf{chk}_3 = 1$, ignore the processing of $\mathsf{a_{new}}$ and discard it.

Figure 4: Duplicity check

To check if a valid uid has been submitted, escrows recompute $[\![\mathsf{uid}']\!] = \Pi_{\mathsf{mac}}([\![\mathsf{sk_m}]\!], [\![\mathsf{a_{new}.r}]\!])$ using the secret-shared r submitted as part of the new allegation, $\mathsf{a_{new}}$, and check equality of uid′ and $\mathsf{a_{new}}$.uid. If the submitted uid is valid, escrows proceed to verify if there exists an allegation in the system $\mathsf{a_i} \in \mathsf{A}$ such that $\mathsf{a_i}$.uid = $\mathsf{a_{new}}$.uid and $\mathsf{a_i}$.pid = $\mathsf{a_{new}}$.pid. To determine this, for each allegation in the system, escrows invoke Π_{eq} (Table 2) protocol to check for equality of pids and equality of uids. To determine if $\mathsf{a_i}$ is a duplicate, escrows check if both equalities hold. Formal details appear in Fig. 4.

Finally, observe that escrows essentially perform a linear scan over the list of all allegations in the system. Hence duplicity check has the complexity of $O(\mathsf{N})$.

4.4 Matching protocol

Let $\mathsf{a_{new}}$ be a newly filed allegation. The objective of matching protocol is to determine the *largest* set $\mathcal{S}$, if any, of matching allegations against $\mathsf{a_{new}}$.pid, whose reveal criteria is met, i.e., reveal threshold of each allegation in $\mathcal{S}$ must be less than $|\mathcal{S}|$.

A cleartext algorithm for identifying $\mathcal{S}$ is as follows: (a) sort the matching allegations in A (associated with $\mathsf{a_{new}}$.pid) in decreasing order of threshold; let $\mathsf{A}' = \{\mathsf{a}_1', \mathsf{a}_2', \ldots, \mathsf{a}_\mathsf{h}'\}$ be the sorted list of matched allegations, (b) let $\mathcal{S}_i = \{\mathsf{a_i}', \mathsf{a}_{i+1}', \ldots, \mathsf{a}_\mathsf{h}'\}$, and check if $\mathsf{a_i}'.\mathsf{t} < |\mathcal{S}_i|$, starting from $i = 1$ to h, (c) the lowest value of i (i.e., largest $\mathcal{S}_i$) for which (b) is satisfied determines largest revealable subset $\mathcal{S}_i$.

To preserve privacy, it is *not* enough to do the above computation on secret-shared data (that include $\mathsf{a_{new}}$ and A). The sequence of operations carried out during the computation must also not leak private information. In particular, in the cleartext algorithm described above, the length of the list $|\mathsf{A}'| = \mathsf{h}$ leaks information on the number of matching allegations against $\mathsf{a_{new}}$.pid in the system. Further, during the construction of $\mathcal{S}_i$, the inclusion or exclusion

[4]User U cannot impersonate user U′ unless it obtains U′.uid by colluding with U′. Due to accountability property, such collusions are deterred.

of an allegation in $\mathcal{S}_i$ reveals information about the relative ordering of the allegations with respect to the threshold. Thus, the sequence of operations (selection, comparison, equality, etc.) on secret-shared data also needs to be carried out *obliviously*. A protocol is data-oblivious if the sequence of operations and memory accesses made during the protocol run are independent of the input. Thus, our objective is to design a data-oblivious matching protocol that operates on secret shared data. As an example, we note that this would mean we must operate on A rather than identifying A′ during matching. Our matching protocol thus consists of the following four steps.

Step 1: Updating $\mathsf{a_{new}}$*'s threshold based on perpetrators revealed so far.* When a new allegation, $\mathsf{a_{new}}$, is filed in the system, its threshold should be reduced by the number of allegations that have already been revealed against $\mathsf{a_{new}}$.pid. This ensures that the alleger of $\mathsf{a_{new}}$ only has to find this reduced number of supporters from the unrevealed ones in A. The first step checks if there exists an entry for perpetrator $\mathsf{a_{new}}$.pid in P. If true, then the threshold $\mathsf{a_{new}}$.t is reduced by p_i.ac, where $\mathsf{a_{new}}$.pid = p_i.pid. Note that these steps should be performed obliviously. For this, we first compute a secret-shared bit via Π_{eq} that determines if $\mathsf{a_{new}}$.pid = p_i.pid, for each $\mathsf{p}_i \in$ P. Then, the oblivious select protocol Π_{sel} (Table 2) is invoked to obliviously determine if the threshold remains unchanged ($\mathsf{a_{new}}$.t) or is updated ($\mathsf{a_{new}}$.t − p_i.ac).

Step 2: Searching for largest $\mathcal{S}$. Determining the *largest* set of revealable matching allegations, translates to finding the highest threshold $\mathsf{t_{max}}$ (among the filed allegations) such that the number of matching allegations $\mathsf{a_j}$ with $\mathsf{a_j}$.t ≤ $\mathsf{t_{max}}$ is greater than $\mathsf{t_{max}}$. Let maxths denote the maximum reveal threshold allowed in Shield. To determine such a $\mathsf{t_{max}}$, we examine each possible threshold from maxths down to 1 iteratively. The order ensures we stop with the largest set $\mathcal{S}$. Starting with i = maxths iteration, we scan through the list of all allegations to determine the count (= $|\mathcal{S}|$) of matching allegations having threshold less than $i+1$ (=$\mathsf{t_{max}}+1$). The i^{th} loop terminates if and when the determined count value is greater than i and a flag is set. In each iteration, to avoid repeated comparison between allegations $\mathsf{a_i}$ and $\mathsf{a_{new}}$ to determine if there is a match, we store this information, generated during the first scan, in a Boolean array M of length N+1 (including $\mathsf{a_{new}}$). The array entries are secret shared bits where the i^{th} entry, $[\![\mathsf{M}_i]\!]^{\mathbf{B}}$ is a 1 if $\mathsf{a_i}$.pid = $\mathsf{a_{new}}$.pid, and 0 otherwise. To prevent leaking information about whether an allegation satisfies the above threshold conditions or not, steps of updating count are performed obliviously via Π_{sel}.

Step 3: Update A. This step is executed only when flag is set in some iteration i in step 2, indicating existence of $\mathcal{S}$, with count number of matching allegations and with highest threshold equal to i. The goal here is to identify the allegations in this set, delete these from A, and update the threshold of non-revealable yet matching allegations to reflect the newly revealed number of allegations—all of these without leaking any additional information. To determine if an allegation belongs to $\mathcal{S}$, we scan through the list of all allegations $\mathsf{a_j} \in$ A, and identify if a matching $\mathsf{a_j}$ is revealable, i.e., $\mathsf{a_j}$.t ≤ i < count. This is done by reducing the thresholds of all matching allegations in A by count, followed by checking if the updated threshold is < 0 (which will be the case if it is a revealable allegation). This operation implicitly updates the threshold of non-revealable yet matching allegations. The operation of whether $\mathsf{a_j}$ is a match and consequently whether the threshold is updated or not, is made oblivious, as in step 1. That is, we compute a secret-shared bit that implies if $\mathsf{a_j}$.pid = $\mathsf{a_{new}}$.pid, and use it to update the threshold ($\mathsf{a_j}$.t − count or $\mathsf{a_j}$.t) via Π_{sel}. Next, delete $\mathcal{S}$ from A.

Step 4: Updating P *when* $\mathcal{S}$ *is found.* If $\mathsf{a_{new}}$.pid does not exist in P, then a new entry in P is added with its allegation count ac = count. Else, the corresponding entry, say, p_i is updated to reflect the number of newly revealed allegations, i.e., p_i.ac = p_i.ac + count. This step is performed via Π_{sel}, as described earlier. The formal matching protocol appears in Fig. 5. The matching protocol entails performing multiple scans over the list of all allegations. The number of times the scan is performed may vary. However, note that the protocol terminates after at most maxths iterations, where maxths is the upper bound on the reveal threshold. Hence, matching has a complexity of $O(\mathsf{N} \cdot \text{maxths})$.

Protocol $\Pi_{\mathsf{Mat}}(\{\mathsf{a}_1, \ldots, \mathsf{a}_\mathsf{N}\}, \mathsf{a_{new}}, \mathsf{P})$

STEP 1:

– **for** *each* p_i *in* P **do**:
- $[\![\mathsf{chk}_1]\!]^{\mathbf{B}} = \Pi_{\mathsf{eq}}([\![\mathsf{a_{new}.pid}]\!], [\![\mathsf{p}_i.\mathsf{pid}]\!])$
- $[\![\mathsf{a_{new}.t}]\!] = \Pi_{\mathsf{sel}}([\![\mathsf{a_{new}.t}]\!], [\![\mathsf{a_{new}.t} - \mathsf{p}_i.\mathsf{ac}]\!], [\![\mathsf{chk}_1]\!]^{\mathbf{B}})$

– Add $\mathsf{a_{new}}$ as the $\mathsf{N}+1^{\text{th}}$ allegation in the system

STEP 2:

– **for** $i = 1$ *to* N **do**: $[\![\mathsf{M}_i]\!]^{\mathbf{B}} = \Pi_{\mathsf{eq}}([\![\mathsf{a_{new}.pid}]\!], [\![\mathsf{a_i.pid}]\!])$

– Set $[\![\mathsf{M}_{\mathsf{N}+1}]\!]^{\mathbf{B}} = [\![1]\!]^{\mathbf{B}}$

– **for** *each possible threshold* i = maxths *to* 1 **do**:
- $[\![\mathsf{count}]\!] = [\![0]\!]$
- **for** *each allegation* $\mathsf{a_j}$ *for* $j = 1$ *to* N + 1 **do**
  - $[\![\mathsf{chk}_2]\!]^{\mathbf{B}} = \Pi_{\mathsf{comp}}([\![\mathsf{a_j.t}]\!], i+1)$
  - $[\![\mathsf{count}]\!] = [\![\mathsf{count}]\!] + \Pi_{\mathsf{bit2A}}([\![\mathsf{M}_j]\!]^{\mathbf{B}} \cdot [\![\mathsf{chk}_2]\!]^{\mathbf{B}})$
- Reconstruct flag = $\Pi_{\mathsf{comp}}([\![i]\!], [\![\mathsf{count}]\!])$. If flag is set to 1, **break** from the loop

STEP 3:

– If the flag is set to 0, Terminate the protocol. Else, **for** *each allegation* $\mathsf{a_j}$ *for* $j = 1$ *to* N + 1 **do**:
- $[\![\mathsf{a_j.t}]\!] = \Pi_{\mathsf{sel}}([\![\mathsf{a_j.t}]\!], [\![\mathsf{a_j.t} - \mathsf{count}]\!], [\![\mathsf{M}_j]\!]^{\mathbf{B}})$
- Reconstruct $\mathsf{chk}_3 = \Pi_{\mathsf{comp}}([\![\mathsf{a_j.t}]\!], [\![0]\!])$
- If $\mathsf{chk}_3 = 1$, include $\mathsf{a_j}$ in $\mathcal{S}$ and delete from A Else, **continue**.

STEP 4:

– Initialize $[\![\mathsf{flag}]\!]^{\mathbf{B}} = 0$

– **for** *each record* p_i *in* P **do**:
- $[\![\mathsf{chk}_1]\!]^{\mathbf{B}} = \Pi_{\mathsf{eq}}([\![\mathsf{a_{new}.pid}]\!], [\![\mathsf{p}_i.\mathsf{pid}]\!])$
- $[\![\mathsf{flag}]\!]^{\mathbf{B}} = \Pi_{\mathsf{sel}}([\![\mathsf{flag}]\!]^{\mathbf{B}}, [\![1]\!]^{\mathbf{B}}, [\![\mathsf{chk}_1]\!]^{\mathbf{B}})$
- $[\![\mathsf{p}_i.\mathsf{ac}]\!] = \Pi_{\mathsf{sel}}([\![\mathsf{p}_i.\mathsf{ac}]\!], [\![\mathsf{p}_i.\mathsf{ac}]\!] + [\![\mathsf{count}]\!], [\![\mathsf{chk}_1]\!]^{\mathbf{B}})$

– Reconstruct flag. If not set, then create a new record p_{i+1} in list defined as $[\![\mathsf{p}_{i+1}.\mathsf{pid}]\!] = [\![\mathsf{a_{new}.pid}]\!]$ and $[\![\mathsf{p}_{i+1}.\mathsf{ac}]\!] = [\![\mathsf{count}]\!]$.

Figure 5: Allegation matching

5 ADDITIONAL FEATURES

Our system can easily be extended to support modification and deletion of filed allegations. The protocols for the same follow on similar lines as that of the duplicity check protocol, and entail performing a linear scan of the allegations in the system. The exact details are as described next.

5.1 Allegation Modification

Unlike in Callisto, where any aspect of the allegation can be modified, we only allow U to modify the reveal threshold t and/or allegation text Text. Allowing modifications to other components such as the user-ID uid, randomness r and perpetrator-ID pid of an allegation a is absurd and hampers the functionality of the system. This is because modifying uid, r is equivalent to a user claiming to be someone else. Similarly, modifying the pid is equivalent to the user alleging a different perpetrator when it has already alleged someone else. Hence, there are several checks that the escrows must perform before modifying an allegation to restrict modification to the above components.

First, the escrows must verify that a registered user of the system is submitting the request. Second, the escrows must verify that the user, submitting the (updated) allegation, is not impersonating another user. The first check is addressed by the escrows and the user carrying out the same steps as in allegation filing phase, where the user is now expected to instead submit the updated allegation by consuming a new DS key pair $(\mathsf{pk}^{\mathsf{U}}, \mathsf{sk}^{\mathsf{U}})$. For the second check, to ensure the user is attempting to modify its own allegation, the escrows verify the validity of the submitted uid by recomputing it as done in duplicity check protocol. Once validated, the escrows obliviously identify and update the allegation $\mathsf{a} \in \mathsf{A}$ in question. Note that knowing which allegation was updated may leak sensitive data based on auxiliary information such as time of the filed allegation and time of the modification. Hence, similar to the duplicity check protocol, escrows scan through each allegation a_i in the system and check for equality of uid and pid of a_i and submitted a'. Whenever these components are a match, the escrows obliviously update the components t and Text via Π_{sel}. The escrows additionally maintain a flag to detect if an allegation was modified successfully or not. This is done to ensure that a valid yet malicious user does not burden the system with fake requests to modify an allegation. If all the checks pass and yet the flag is not set, it indicates that no allegation was modified. Hence, the submitted pk^{U} can be used to trace back the user, as done in allegation revealing phase, and penalize it accordingly. That is, the escrows compute the MAC on the submitted pk^{U} and de-anonymize the user through the local map of mac(s). The formal protocol for the same is provided in 6.

Note that each successful request of allegation modification must be followed by allegation matching phase since updating the threshold can trigger the possibility of having a revealable subset.

Protocol $\Pi_{\mathsf{Mod}}(\{\mathsf{a}_1, \ldots, \mathsf{a}_\mathsf{N}\}, \mathsf{a}', \mathsf{sk}_\mathsf{m})$

- Initiate *allegation filing* phase with a' as the submitted allegation. If any verification fails, ignore processing of a', discard it and halt. Else set $[\![\mathsf{flag}]\!] = 0$ and continue.
- $[\![\mathsf{uid}']\!] = \Pi_{\mathsf{mac}}([\![\mathsf{sk}_\mathsf{m}]\!], [\![\mathsf{a}'.\mathsf{r}]\!])$ and $[\![\mathsf{chk}]\!]^{\mathbf{B}} = \Pi_{\mathsf{eq}}([\![\mathsf{uid}']\!], [\![\mathsf{a}'.\mathsf{uid}]\!])$
- Reconstruct chk. If $\mathsf{chk} = 1$, then for each allegation $\mathsf{a}_\mathsf{i} \in \mathsf{A}$,
  - $[\![\mathsf{chk}_1^i]\!]^{\mathbf{B}} = \Pi_{\mathsf{eq}}([\![\mathsf{a}_\mathsf{i}.\mathsf{uid}]\!], [\![\mathsf{a}'.\mathsf{uid}]\!])$
  - $[\![\mathsf{chk}_2^i]\!]^{\mathbf{B}} = \Pi_{\mathsf{eq}}([\![\mathsf{a}_\mathsf{i}.\mathsf{pid}]\!], [\![\mathsf{a}'.\mathsf{pid}]\!])$
  - $[\![\mathsf{chk}_3^i]\!]^{\mathbf{B}} = [\![\mathsf{chk}_1^i \cdot \mathsf{chk}_2^i]\!]^{\mathbf{B}}$
  - $[\![\mathsf{a}_\mathsf{i}.\mathsf{t}]\!] = \Pi_{\mathsf{sel}}([\![\mathsf{a}_\mathsf{i}.\mathsf{t}]\!], [\![\mathsf{a}'.\mathsf{t}]\!], [\![\mathsf{chk}_3^i]\!]^{\mathbf{B}})$
  - $[\![\mathsf{a}_\mathsf{i}.\mathsf{Text}]\!] = \Pi_{\mathsf{sel}}([\![\mathsf{a}_\mathsf{i}.\mathsf{Text}]\!], [\![\mathsf{a}'.\mathsf{Text}]\!], [\![\mathsf{chk}_3^i]\!]^{\mathbf{B}})$
- Reconstruct $\mathsf{flag} = \Pi_{\mathsf{dotp}}(\{[\![\mathsf{chk}_1^i]\!]^{\mathbf{B}}, [\![\mathsf{chk}_2^i]\!]^{\mathbf{B}}\}_{i=1}^{\mathsf{N}})$. If flag = 1, report success. Else trace malicious user via map of mac(s) on verification keys.

Figure 6: Allegation modification

5.2 Allegation Deletion

Not only must a user U be allowed to modify an allegation, but the system must also facilitate U to delete the same. The designed system facilitates deletion and works on similar lines of allegation modification. U places the request for deletion by resubmitting the allegation to be deleted, albeit under a new pair of DS key pair $(\mathsf{pk}^{\mathsf{U}}, \mathsf{sk}^{\mathsf{U}})$. The escrows perform the same verification as done in allegation filing phase. If all the verification succeeds, the escrows recompute the uid' using r and match it against the submitted uid, as done in allegation modification. If the check passes, the escrows identify the allegation to be deleted. Unlike in the case of allegation modification protocol where it was necessary to hide the allegation being modified, we note that for deleting an allegation, the escrows must learn which allegation it is. Hence, the identification of the allegation in question need not be performed obliviously. This also avoids explicitly maintaining a flag variable. Thus, escrows perform a linear scan over the list of all allegations to determine the a_i that has uid (using chk_1^i) and pid (using chk_2^i) equal to that of the submitted allegation a'. The escrows determine if both the equalities hold (using chk_3^i) and delete their shares of a_i if this is the case. If no such allegation a_i is found, then the escrows determine the malicious user by tracing the identity of the user using the submitted pk^{U}. The formal protocol for the same is given in 7.

Protocol $\Pi_{\mathsf{Del}}(\{\mathsf{a}_1, \ldots, \mathsf{a}_\mathsf{N}\}, \mathsf{a}', \mathsf{sk}_\mathsf{m})$

- Initiate *allegation filing* phase with a' as the submitted allegation. If any verification fails, ignore processing of a', discard it and halt. Else continue.
- $[\![\mathsf{uid}']\!] = \Pi_{\mathsf{mac}}([\![\mathsf{sk}_\mathsf{m}]\!], [\![\mathsf{a}'.\mathsf{r}]\!])$ and $[\![\mathsf{chk}]\!]^{\mathbf{B}} = \Pi_{\mathsf{eq}}([\![\mathsf{uid}']\!], [\![\mathsf{a}'.\mathsf{uid}]\!])$
- Reconstruct chk. If $\mathsf{chk} = 1$, then for each allegation $\mathsf{a}_\mathsf{i} \in \mathsf{A}$,
  - $[\![\mathsf{chk}_1^i]\!]^{\mathbf{B}} = \Pi_{\mathsf{eq}}([\![\mathsf{a}_\mathsf{i}.\mathsf{uid}]\!], [\![\mathsf{a}'.\mathsf{uid}]\!])$
  - $[\![\mathsf{chk}_2^i]\!]^{\mathbf{B}} = \Pi_{\mathsf{eq}}([\![\mathsf{a}_\mathsf{i}.\mathsf{pid}]\!], [\![\mathsf{a}'.\mathsf{pid}]\!])$
  - $[\![\mathsf{chk}_3^i]\!]^{\mathbf{B}} = [\![\mathsf{chk}_1^i \cdot \mathsf{chk}_2^i]\!]^{\mathbf{B}}$
  - Reconstruct chk_3^i. If $\mathsf{chk}_3^i = 1$ delete a_i, report success, halt.
- If no a_i was deleted, then trace the malicious user using the local map of mac(s) on verification keys.

Figure 7: Allegation deletion

A possible optimization is to associate a public token with every allegation and enforce the user to also submit the corresponding token when requesting deletion. This allows the escrows to identify the requested allegation to be deleted without relying on MPC. However, the escrows are required to use MPC to verify that the requested allegation is, in fact, filed by the user. Hence, all checks performed previously remain the same, except they are performed

specific to the identified allegation directly. This avoids the linear scan across A.

We finally would like to note that our system can be extended to handle different types of crimes. Towards this, the allegation can include an additional field indicating the type. Note that this would affect the duplicity check and matching protocols and require careful re-definitions of the same.

6 DISCUSSIONS

Here we address some concerns that may arise with respect to the design of Shield.

Privacy vs. efficiency. The inception of SAE systems was to protect user privacy and thereby encourage user participation. Hence, for a user-centric system such as SAE, privacy should be given utmost importance and must not be compromised at any cost. Further, a user who filed an allegation is inherently bound to wait until matching allegations are found and can be revealed. Given that this waiting period can vary from days to months [6, 37], improving the efficiency of the system (complexity of processing a filed allegation) has an insignificant effect on the wait time of the user. Thus, choosing efficiency over privacy is ill-advised for an SAE system.

Pessimistic view of the world. Our design decisions are made considering the worst-case scenario. Specifically, we assume malicious capabilities of users and escrows and their collusion. The protocols in Shield allow the escrows to detect misbehaviour by a malicious user. Relying on the accountability property offered by Shield, the escrows (or higher authorities) can trace back the user's real-world identity and punish the same. Thus, the accountability property plays an important role in deterring malicious users. Further, the choice of variable threshold instead of a globally fixed one is made to cater to the victims' varying levels of vulnerability, thereby making the system more inclusive. Presence of flexible reveal threshold further demands empowering a victim to change the reveal threshold of its allegation, which may be necessary to ensure the allegation is revealed sooner. This is facilitated by the allegation modification protocol. Note that our design choices in no way make us loose-out on any feature provided by prior systems, rather only improve upon them.

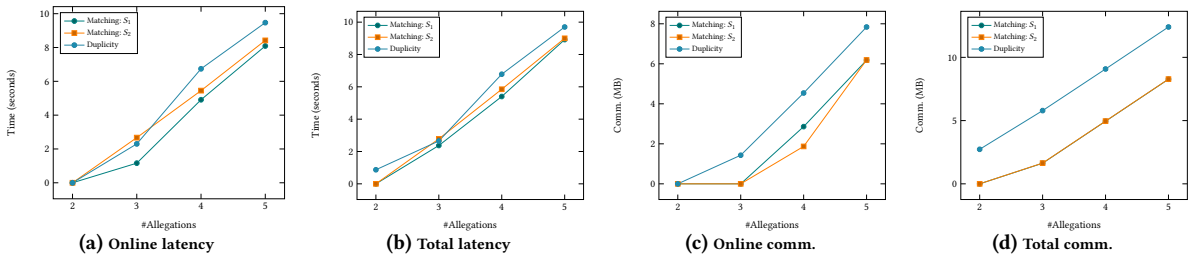
Choice of MPC . Note that an allegation escrow system has highly sensitive information and is required to run perpetually. Realization of the system cannot afford to cease providing the service due to malicious activities. Thus, employing any MPC protocol that enables the adversary to abort would be a misfit. However, previous works rely on MPC with identifiable abort (i.e., the protocol aborts and reveals the identity of the corrupt party upon misbehaviour) but restart the computation with one less party when a corrupt party is identified. Although this is a possibility, it comes at the cost of re-sharing the state of the corrupt party (escrow) that is thrown out of the computation, which is expensive. Recent protocols in the literature [12, 13, 26, 27] achieve the strongest security of *guaranteed output delivery* (i.e., protocol guarantees delivery of output irrespective of any misbehaviour) at the cost of weakest notion of abort security (i.e., the protocol may allow the adversary alone to obtain the output and abort the computation). Hence, aiming for guaranteed output delivery (GOD) does not come at any additional cost. Moreover, GOD prevents an adversary from wasting honest escrow's valuable compute resources by preventing repeated failures, which is otherwise possible in the weaker security notions that allow an adversary to abort. Thus, presence of GOD uplifts the trust in the system and encourages user participation. It is well known that an honest majority among the computing parties is necessary to achieve GOD [19]. Moreover, due to the challenges in identifying a large number of compute parties for real world deployments and efficiency reasons, MPC for small number of parties is gaining huge interest [4, 5, 11, 12, 15–18, 21, 32, 34, 36, 36]. Hence, to realize an SAE system, we focus on benchmarking honest majority MPC protocols with a small number of parties providing strongest security of GOD.

Other challenges. Several details such as requirement of anonymous communication channels for alleger anonymity, trust in the user's client-side software, and other deployment considerations, which apply to our solution, are not emphasized. These follow from prior works, as described in the deployment considerations of [6]. Our goal was to identify privacy breaches in the existing works and formalize the security desirable in such a system. Thus, we only provide an algorithmic solution that achieves the desired security. Addressing the system-level challenges that may arise in the actual deployment of the solution and designing user-friendly interface is the necessary next step and is left as future work.

7 BENCHMARKS

Shield, can be realized using any MPC protocol that provides the identified primitives in Table 2. Although we prioritize privacy over efficiency, we strive to achieve as efficient a solution as possible. Hence, we instantiate the MPC of Shield using state-of-the-art robust 3PC (with $n = 3, t = 1$) and 4PC (with $n = 4, t = 1$) frameworks of SWIFT [26] and Tetrad [27], respectively. Elaborately, the following are the reasons for our choice of MPC–(i) SWIFT and Tetrad are honest majority MPC ($t = 1$) protocols with a small population (3PC, 4PC) which have been shown to be more efficient than the dishonest majority counterpart, (ii) these allow operating over ring algebraic structure, which is more efficient than operating over fields [2], (iii) both these frameworks are designed in the preprocessing model, where input-independent computation is offloaded to a preprocessing phase, paving way for a fast online phase once the input is available. Since escrows are required to actively carry out computations only if an allegation is filed (which occurs sparsely); else, they can carry out the preprocessing computations leisurely, the preprocessing model becomes an apt fit for our application, (iv) SWIFT and Tetrad support all primitives (Table 2) except for equality protocol (Π_{eq}) which can be derived from the one in [35].

We implement all the protocols in python, including that of [27] and [26]. Our code accounts for multithreading. We instantiate the communication layer between the parties using PyTorch library. We use Crypto library for AES and hashlib for generating SHA256 hash. Our code, developed for benchmarking and not optimized for industry-grade use. We note that a C++ based implementation can give better performance. Our protocols are benchmarked over LAN, instantiated using n1-standard-64 instances of Google Cloud with 2.3 GHz Intel Xeon E5 v3 (Haswell) processors, and 240 GB of RAM. The machines have a bandwidth of 16Gbps. We use latency



(a) Online latency **(b)** Total latency **(c)** Online comm. **(d)** Total comm.

Figure 8: Variation in latency and communication for varying number of allegations for duplicity and matching protocol for 4PC. For matching, S_1 indicates the setting where #perpetrators revealed are sublinear in #allegations, while S_2 indicates setting where #perpetrators revealed is 10% of #allegations. Plots are log-log plots with x-axis logarithmic in base 10 and y-axis logarithmic in base 2.

(time taken for protocol to complete) and communication between escrows as the two parameters for benchmarks. We report values separately for online phase and total (= preprocessing + online). Our protocols are benchmarked over the ring $\mathbb{Z}_{2^\ell}$ ($\ell = 64$), except for computing MAC and VRF, security of which demands operating on a 1024-bit prime-order field.

Recall the six phases that comprise the Shield system. Observe that initialization is one-time process, and hence does not contribute to the cost of keeping the system running. However, the escrows may be challenged to register multiple users at once. Hence, we report the following–(i) cost for registering a single user, (ii) throughput which accounts for the number of users that can be handled in parallel per minute by the escrows. In this regard, Table 4 reports (i), where we fix on the number of digital signature keys being registered by the user as 50 following [6]. Further, for both 3PC and 4PC we observe a throughput of 128 registrations/min.

#Escrows	Online		Total	
	Latency (s)	Com (MB)	Latency (s)	Com (MB)
3	3.79	39.44	8.51	157.73
4	3.41	39.32	5.28	78.64

Table 4: Communication and latency for registering a user in 3PC, 4PC.

Allegation filing involves escrows locally performing operations (without any interaction). Hence, we do not explicitly report the cost of this phase. Duplicity check and allegation matching make up the compute-intensive phases for running the system. Unlike registration, these phases can only process one allegation at a time. Hence, we report costs for processing one filed allegation in these phases. These costs were not reported in [6] since it had a constant-time matching (and duplicity check was missing). Hence, the costs reported here capture the overhead in comparison to [6], which is the price paid for obtaining full privacy.

Complexity of our duplicity check is dependent on the number of allegations in the system and hence is benchmarked for varying number of allegations. For this, we consider a system where the number of filed allegations ranges from 100-100,000, which is sufficient to account for deployment in any institution (e.g., universities, private or government workplaces, etc.). We note that 10^5 filed (unrevealed) allegations account for a pessimistic view, so real-world deployment may be faster. The variations in latency and communication for online and preprocessing phases appears in Table 5. The reported online latency is within 12 minutes even in the presence of 10^5 allegations, which showcases its practicality. As expected, communication scales linearly with number of allegations.

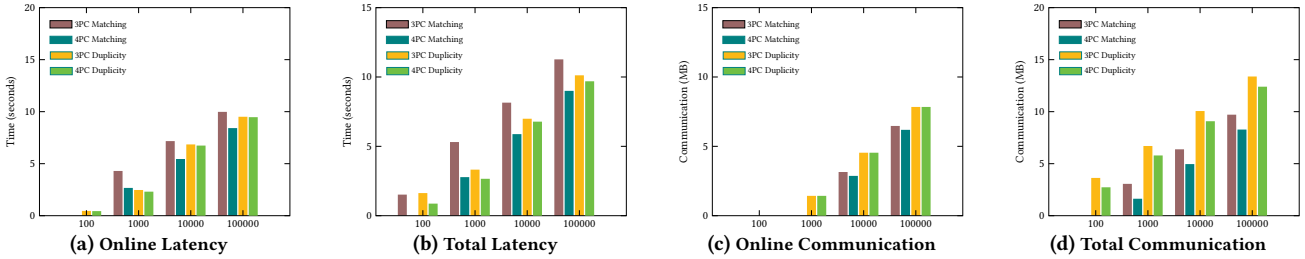
#Escrows	|A|	Online		Total	
		Latency (s)	Com (MB)	Latency (s)	Com (GB)
3	10^2	1.37	0.62	3.09	0.01
	10^3	5.56	2.69	10.02	0.10
	10^4	115.37	23.41	127.73	1.07
	10^5	731.48	230.57	1113.68	10.70
4	10^2	1.36	0.254	1.83	0.01
	10^3	4.96	2.32	6.30	0.05
	10^4	72.97	23.04	110.54	0.54
	10^5	709.96	230.199	831.66	5.42

Table 5: Communication and latency for duplicity check for 3PC, 4PC.

#Escrows	|A|	|P|	Online		Total	
			Latency (s)	Com (MB)	Latency (s)	Com (MB)
3	10^2	10	0.88	0.09	2.87	0.83
	10^3	30	12.24	0.87	27.03	7.81
		10^2	19.78	0.89	39.89	8.34
	10^4	10^2	95.76	8.71	234.12	83.42
		10^3	144.87	8.95	285.42	83.82
	10^5	300	867.26	87.18	2053.87	834.29
		10^4	1012.55	89.58	2477.32	838.26
4	10^2	10	0.50	0.073	0.55	0.31
	10^3	30	2.25	0.72	5.20	3.09
		10^2	6.38	0.73	6.89	3.12
	10^4	10^2	30.24	7.28	42.38	30.91
		10^3	43.75	7.32	59.09	31.23
	10^5	300	273.80	72.72	486.98	308.90
		10^4	342.81	73.27	513.26	312.39

Table 6: Overhead of matching for varying number of allegations and number of revealed perpetrators in 3PC, 4PC.

The above analysis also holds for matching since it too depends on the number of allegations in the system. Additionally, the matching protocol depends on the size of the revealed perpetrator list P, and the upper bound on the reveal threshold. To analyze the effect of this, we benchmark cases where |P| may be linear (1/10) or sub-linear ($\sqrt{\cdot}$) in the number of allegations and bound the reveal threshold by 10. These results are reported in Table 6. |P| is chosen to be linear and sub-linear in the number of allegations to account for a large band of variation in the possible number of revealed perpetrators. As evident from Table 6, in the presence of 10^5 allegations in the system, processing a new allegation requires < 6 minutes in 4PC and < 17 minutes in 3PC, in the online phase. Thus,



(a) Online Latency (b) Total Latency (c) Online Communication (d) Total Communication

Figure 9: Comparison of matching and duplicity check protocol for varying number of allegations with $|P| = 1/10 \cdot |A|$ and maximum threshold set to 10 for 3PC, 4PC. Note that all plots are log-log plots with x-axis logarithmic in base 10 and y-axis logarithmic in base 2. We do not report values < 1 as their log is negative.

the online run time of matching protocol showcases its practicality despite having dependence on the number of allegations in the system. This can be attributed to the use of preprocessing model. Moreover, for 10^5 allegations, changing $|P|$ from 300 to 10^4 has an overhead of not more than 3 minutes in online runtime, and less than 3 MB overhead in online communication. This shows the minimal effect $|P|$ has on complexity of matching. Further, to showcase the effect of threshold bound on complexity of matching, we vary the bound from 10 to 50 with fixed $|A| = 10^5$, $|P| = 10^4$ and report results in Table 7. An increase in bound from 10 to 50 results in an overhead of 2.41×, 2.61× in online latency and 4.96×, 4.66× in online communication for 3PC and 4PC, respectively.

#Escrows	Max. threshold	Online		Total	
		Latency (s)	Com (MB)	Latency (s)	Com (MB)
3	10	1012.55	89.58	2477.32	838.26
	20	1487.66	184.11	3003.68	842.75
	30	1932.45	271.01	3582.49	1236.51
	40	1981.40	357.92	3889.09	1630.26
	50	2442.85	444.82	4715.39	2024.01
4	10	342.81	73.27	513.26	312.39
	20	470.78	140.20	663.88	379.51
	30	635.99	207.52	741.78	446.64
	40	610.09	274.64	922.12	513.76
	50	894.77	341.77	976.97	580.89

Table 7: Communication and latency for matching with varying upper bounds on threshold in 3PC, 4PC for $|A| = 10^5$, $|P| = 10^4$.

As expected, from the values reported in Table 5, Table 6, 4PC fares better than 3PC. This effect can also be visualized from Fig. 9. Having established this, we compare the matching and duplicity check protocol in Fig. 8 under 4PC setting. As expected, matching fares better than duplicity since it works over a ring as opposed to duplicity, which works over a field.

Since the operations in allegation modification are similar to those in duplicity check, its cost is the same as that reported for duplicity check. The same holds true for deletion.

8 CONCLUSION

We identify the privacy issues and drawbacks in prior systems, and design a secure allegation escrow system that addresses these. This required incorporating a new duplicity check protocol to prevent prematurely revealing allegations due to filing of duplicates. We also incorporate a more secure matching protocol that addresses the privacy issues present in the prior systems. We also provide protocols for modifying and deleting filed allegations, that were missing in [6], making our system more comprehensive. In this way, our system retains the salient features of prior works and guarantees better privacy. Moreover, the proposed system is generic–can handle more than one type of crime and makes black-box use of the underlying MPC. To showcase the practicality of the designed system, we benchmark the same with state-of-the-art robust MPC protocols with 3 and 4 parties. We believe that the recent model of Friends-and-Foes (FaF) secure MPC [3] would be apt for designing SAE systems as opposed to traditional MPC protocols. We leave realising this as future work.

REFERENCES

[1] 2006. *Me Too Movement.* https://metoomvmt.org/ Accessed: 2021-11-18.
[2] Mark Abspoel, Anders Dalskov, Daniel Escudero, and Ariel Nof. 2021. An efficient passive-to-active compiler for honest-majority MPC over rings. In *ACNS*. Springer.
[3] Bar Alon, Eran Omri, and Anat Paskin-Cherniavsky. 2020. MPC with Friends and Foes. In *CRYPTO*.
[4] Toshinori Araki, Assi Barak, Jun Furukawa, Tamar Lichter, Yehuda Lindell, Ariel Nof, Kazuma Ohara, Adi Watzman, and Or Weinstein. 2017. Optimized Honest-Majority MPC for Malicious Adversaries - Breaking the 1 Billion-Gate Per Second Barrier. In *IEEE S&P*.
[5] Toshinori Araki, Jun Furukawa, Yehuda Lindell, Ariel Nof, and Kazuma Ohara. 2016. High-Throughput Semi-Honest Secure Three-Party Computation with an Honest Majority. In *ACM CCS*.
[6] Venkat Arun, Aniket Kate, Deepak Garg, Peter Druschel, and Bobby Bhattacharjee. 2020. Finding Safety in Numbers with Secure Allegation Escrows. In *NDSS*.
[7] Azer Bestavros, Andrei Lapets, and Mayank Varia. 2017. User-centric distributed solutions for privacy-preserving analytics. *Commun. ACM* (2017).
[8] Dan Bogdanov, Marko Jõemets, Sander Siim, and Meril Vaht. 2015. How the Estonian Tax and Customs Board Evaluated a Tax Fraud Detection System Based on Secure Multi-party Computation. In *FC*.
[9] Dan Bogdanov, Riivo Talviste, and Jan Willemson. 2012. Deploying Secure Multi-Party Computation for Financial Data Analysis - (Short Paper). In *FC*.
[10] Peter Bogetoft, Dan Lund Christensen, Ivan Damgård, Martin Geisler, Thomas Jakobsen, Mikkel Krøigaard, Janus Dam Nielsen, Jesper Buus Nielsen, Kurt Nielsen, Jakob Pagter, et al. 2009. Secure multiparty computation goes live. In *FC*.
[11] Dan Boneh, Elette Boyle, Henry Corrigan-Gibbs, Niv Gilboa, and Yuval Ishai. 2019. Zero-Knowledge Proofs on Secret-Shared Data via Fully Linear PCPs. In *CRYPTO*.
[12] Elette Boyle, Niv Gilboa, Yuval Ishai, and Ariel Nof. 2019. Practical Fully Secure Three-Party Computation via Sublinear Distributed Zero-Knowledge Proofs. In *ACM CCS*.
[13] Elette Boyle, Niv Gilboa, Yuval Ishai, and Ariel Nof. 2020. Efficient Fully Secure Computation via Distributed Zero-Knowledge Proofs. In *ASIACRYPT*. Springer.
[14] Ferdinand Brasser, Urs Müller, Alexandra Dmitrienko, Kari Kostiainen, Srdjan Capkun, and Ahmad-Reza Sadeghi. 2017. Software grand exposure:SGX cache attacks are practical. In *WOOT@USENIX*.
[15] Megha Byali, Harsh Chaudhari, Arpita Patra, and Ajith Suresh. 2020. FLASH: Fast and Robust Framework for Privacy-preserving Machine Learning. *PETS* (2020).

[16] Megha Byali, Arun Joseph, Arpita Patra, and Divya Ravi. 2018. Fast Secure Computation for Small Population over the Internet. In *ACM CCS*.

[17] Harsh Chaudhari, Ashish Choudhury, Arpita Patra, and Ajith Suresh. 2019. ASTRA: High Throughput 3PC over Rings with Application to Secure Prediction. In *ACM CCSW@CCS*. https://eprint.iacr.org/2019/429

[18] Harsh Chaudhari, Rahul Rachuri, and Ajith Suresh. 2020. Trident: Efficient 4PC Framework for Privacy Preserving Machine Learning. *NDSS* (2020). https://arxiv.org/abs/1912.02631

[19] Richard Cleve. 1986. Limits on the Security of Coin Flips when Half the Processors Are Faulty (Extended Abstract). In *ACM STOC*.

[20] Yevgeniy Dodis and Aleksandr Yampolskiy. 2005. A verifiable random function with short proofs and keys. In *International Workshop on Public Key Cryptography*. Springer.

[21] Jun Furukawa, Yehuda Lindell, Ariel Nof, and Or Weinstein. 2017. High-Throughput Secure Three-Party Computation for Malicious Adversaries and an Honest Majority. In *EUROCRYPT*.

[22] Oded Goldreich. 2007. *Foundations of cryptography: volume 1, basic tools*. Cambridge university press.

[23] Danny Harnik, Paula Ta-Shma, and Eliad Tsfadia. 2018. It takes two to# metoo-using enclaves to build autonomous trusted systems. *arXiv preprint arXiv:1808.02708* (2018).

[24] Alicia Iriberri, Gondy Leroy, and Nathan Garrett. 2006. Reporting on-campus crime online: User intention to use. In *Proceedings of the 39th Annual Hawaii International Conference on System Sciences (HICSS'06)*. IEEE.

[25] Mitsuru Ito, Akira Saito, and Takao Nishizeki. 1989. Secret sharing scheme realizing general access structure. *Electronics and Communications in Japan (Part III: Fundamental Electronic Science)* (1989).

[26] Nishat Koti, Mahak Pancholi, Arpita Patra, and Ajith Suresh. 2021. SWIFT: Super-fast and Robust Privacy-Preserving Machine Learning. In *USENIX Security*.

[27] Nishat Koti, Arpita Patra, Rahul Rachuri, and Ajith Suresh. 2022. Tetrad: Actively Secure 4PC for Secure Training and Inference. *To Appear In NDSS* (2022). https://ia.cr/2021/755.

[28] Benjamin Kuykendall, Hugo Krawczyk, and Tal Rabin. 2019. Cryptography for# metoo. *PETS* (2019).

[29] Sangho Lee, Ming-Wei Shih, Prasun Gera, Taesoo Kim, Hyesoon Kim, and Marcus Peinado. 2017. Inferring fine-grained control flow inside SGX enclaves with branch shadowing. In *USENIX*.

[30] Yehuda Lindell. 2017. How to simulate it–a tutorial on the simulation proof technique. In *Tutorials on the Foundations of Cryptography*. Springer.

[31] Payman Mohassel and Peter Rindal. 2018. ABY^3: A Mixed Protocol Framework for Machine Learning. In *ACM CCS*.

[32] Payman Mohassel, Mike Rosulek, and Ye Zhang. 2015. Fast and Secure Three-party Computation: The Garbled Circuit Approach. In *ACM CCS*.

[33] Payman Mohassel and Yupeng Zhang. 2017. SecureML: A System for Scalable Privacy-Preserving Machine Learning. In *IEEE S&P*.

[34] Peter Sebastian Nordholt and Meilof Veeningen. 2018. Minimising Communication in Honest-Majority MPC by Batchwise Multiplication Verification. In *ACNS*.

[35] Arpita Patra, Thomas Schneider, Ajith Suresh, and Hossein Yalame. 2021. ABY2.0: Improved mixed-protocol secure two-party computation. In *USENIX Security*.

[36] Arpita Patra and Ajith Suresh. 2020. BLAZE: Blazing Fast Privacy-Preserving Machine Learning. *NDSS* (2020).

[37] Anjana Rajan, Lucy Qin, David W Archer, Dan Boneh, Tancrede Lepoint, and Mayank Varia. 2018. Callisto: A cryptographic approach to detecting serial perpetrators of sexual misconduct. In *ACM SIGCAS*.

[38] Adi Shamir. 1979. How to share a secret. *Commun. ACM* (1979).

[39] Sameer Wagh, Shruti Tople, Fabrice Benhamouda, Eyal Kushilevitz, Prateek Mittal, and Tal Rabin. 2021. Falcon: Honest-Majority Maliciously Secure Framework for Private Deep Learning. *PoPETS* (2021).

[40] Wenhao Wang, Guoxing Chen, Xiaorui Pan, Yinqian Zhang, XiaoFeng Wang, Vincent Bindschaedler, Haixu Tang, and Carl A Gunter. 2017. Leaky cauldron on the dark land: Understanding memory side-channel hazards in SGX. In *ACM SIGSAC*.

A BUCKETING ALGORITHM OF [6]

The bucketing algorithm is used to identify a set of revealable allegations, if any, in the system when a new allegation is filed. For this, the allegations present in the system are grouped together as *collections* and are stored in a data-structure that consists of numbered *buckets*. The invariant maintained by the data structure is such that an allegation a, waiting for j more matching allegations before it can be revealed, is present in bucket j. Thus, when a new allegation a with threshold t is filed, it is included in bucket t[5]. Additionally, the SAE protocol ensures that the escrows can compare and match allegations only within the same bucket. This is achieved by the escrows associating a unique private key sk_j with respect to each bucket j where the key is known to them in $[\![\cdot]\!]$-shared form. The escrows jointly compute the verifiable pseudo-random function (PRF) value for each allegation in bucket j (using a distributed protocol). Specifically, the escrows invoke a PRF with shares of the bucket key $[\![\mathsf{sk}_j]\!]$ on the shares of the input $[\![\mathsf{H}(\mathsf{pid})]\!]$ for each allegation. The PRF output is revealed on clear to all escrows. This allows the escrows to locally determine the matching allegations within the bucket (two allegations match if they have the same PRF value implying that they have the same meta-data). Such matching allegations are grouped together as a collection. As a collection grows in size, it is copied onto lower buckets to ensure the above mentioned invariant is maintained. During this process, if two different (non-intersecting) collections C_1, C_2 with matching meta-data happen to overlap (i.e., they span across a common bucket), then the collections are coalesced into a single collection C. The collection C consists of the union of all the allegations in C_1, C_2 and is said to span the union of the buckets spanned by C_1 and C_2. Thus, a collection may span across many (consecutive) buckets. It is interesting to note that all the allegations in a collection have matching meta-data, but all allegations with matching meta-data need not belong to the same collection (as they may belong to different collections spanning non-overlapping buckets)! In the life cycle of an allegation (i.e. from its filing to its revelation)- it starts off as a singleton collection; a collection grows in size when more matching allegations are found within the same bucket; the collection is propagated to lower buckets when its size increases (i.e. to maintain the invariant); the collection is revealed when it reaches *bucket-0*, indicative of the fact that the allegations in the collection are waiting for no (0) more allegations before it can be revealed. The exact bucketing rules are given below.

Bucketing Rules: Apply the following rules repeatedly (in any order) till no further rules apply. Rules 2,3 and 4 only apply to collections that haven't been revealed.

1. When an allegation with threshold t is filed, it forms a singleton collection and is added to bucket t.
2. If $Min(A)$ is the smallest bucket occupied by a collection A and every allegation in A has a threshold $< Min(A) + |A| - 1$, 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, they coalesce into one collection.
4. When a collection reaches bucket-0, all of its allegations are revealed.
5. If a collection A is revealed, we make sure it occupies buckets-$1, \ldots, |A|$, even as A grows. This enables future matching allegations to be revealed.

[5]Note that the interpretation of threshold in [6] is slightly different and hence we modify the SAE protocol description to keep the interpretation of t consistent with ours.

B SECURITY OF OUR PROTOCOLS

We prove security using the real-world/ ideal-word simulation paradigm [22, 30]. Informally, a protocol is secure if whatever an adversary can do in the real world protocol can be done in an ideal world, where the latter is secure by definition. In the ideal world, each party sends its input to an incorruptible trusted party over a perfectly secure channel, where the latter computes the function based on these inputs and sends each party its respective output. The computations carried out in the ideal world are captured using an *ideal functionality* ($\mathcal{F}_f$), which the designed MPC protocol Π_f must securely emulate in the real world.

Let $\mathcal{A}$ denote the probabilistic polynomial time (PPT) real-world adversary corrupting at most t parties in $\mathcal{P}$ and $\mathcal{S}_{\mathcal{A}}$ denote the corresponding ideal world adversary. Let $\text{IDEAL}_{\mathcal{F}_f,\mathcal{S}_{\mathcal{A}}}$ denote the joint output of the honest parties and $\mathcal{S}_{\mathcal{A}}$ in the ideal execution. Similarly, let $\text{REAL}_{\Pi_f,\mathcal{A}}$ denote the joint output of the honest parties and $\mathcal{A}$ in the real world execution. We say that the protocol Π_f securely realizes $\mathcal{F}_f$ if for every PPT adversary $\mathcal{A}$ there exists an ideal world adversary $\mathcal{S}_{\mathcal{A}}$ corrupting the same parties such that $\text{IDEAL}_{\mathcal{F}_f,\mathcal{S}_{\mathcal{A}}}$ and $\text{REAL}_{\Pi_f,\mathcal{A}}$ are computationally indistinguishable. The ideal functionality for computing a function f with robustness appears in Fig. 10. The security of the designed protocols are also proved assuming a robust MPC.

Functionality $\mathcal{F}_f$

$\mathcal{F}_f$ interacts with the parties in $\mathcal{P}$ and the adversary $\mathcal{S}_{\mathcal{A}}$. Let f denote the function to be computed. Every honest party $P_i \in \mathcal{P}$ sends its input x_i to $\mathcal{F}_f$. Corrupted parties may send arbitrary inputs as instructed by the adversary.

Step 1: On message (Input, x_i) from P_i, do the following: if $(\text{Input}, *)$ already received from P_i, then ignore the current message. Otherwise, record $x'_i = x_i$ internally. If x_i is outside P_i's domain, consider x'_i to be some predetermined default value.

Step 2: $\mathcal{F}_f$ computes $(\{y_i\}_{i=1}^n) = f(\{x'_i\}_{i=1}^n)$ and sends (Output, y_i) to $P_i \in \mathcal{P}$.

Figure 10: Ideal functionality for function f

B.1 Simulations for layer 2 protocols

We give the ideal functionality for duplicity check and matching in Fig. 11 and Fig. 12, respectively.

Owing to the modular architecture (see Fig. 1), it is easy to observe that the duplicity and matching protocols in layer 2 build on top of protocols in layer 1 and layer 0. Hence, their simulation follows from simulation of underlying protocols.

Lemma B.1 (Security). *Protocol Π_{Dup} (Fig. 4) securely realizes $\mathcal{F}_{\text{Dup}}$ (Fig. 11) in the computational setting.*

Proof. Simulation proceeds via the simulation steps of the underlying protocols. Thus, indistinguishability of the simulated and real-world view of the adversary follows from the indistinguishability of the simulation steps of the underlying MPC protocols. Note that since the simulator receives from the ideal functionality the values that are reconstructed during the protocol execution, the simulator can also successfully reconstruct this towards the adversary. □

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

$\mathcal{F}_{\text{Dup}}$ interacts with the escrows in $\mathcal{P}$, the ideal world malicious adversary $\mathcal{S}_{\mathcal{A}}$.

– It receives as input the shares of the following from the parties: the newly filed allegation a_{new}, allegations in the system $\{a_1, \ldots, a_N\}$, and the MAC key sk_m.

– Reconstruct a_{new}, $\{a_1, \ldots, a_N\}$ and sk_m using the shares of honest parties.

– Recompute the user ID $\text{uid}' = \text{MAC}(sk_m, a_{\text{new}}.r)$ and check if $\text{uid}' = a_{\text{new}}.\text{uid}$.

– If it is not equal, send *ignore* message to escrows and terminate.

– Else check if there exists $a_i \in \{a_1, \ldots, a_N\}$ such that $a_i.\text{uid} = a_{\text{new}}.\text{uid}$ and $a_i.\text{pid} = a_{\text{new}}.\text{pid}$.

– If a_i as described above exists, output 1 to all the escrows. Else, it outputs 0.

Figure 11: Ideal functionality for duplicity check

Lemma B.2 (Security). *Protocol Π_{Mat} (Fig. 5) securely realizes $\mathcal{F}_{\text{Mat}}$ (Fig. 12) in the computational setting.*

Proof. Simulation proceeds via the simulation steps of the underlying protocols. Thus, indistinguishability of the simulated and real-world view of the adversary follows from the indistinguishability of the simulation steps of the underlying MPC protocols. Further, regarding the simulation of break, note that the simulator receives from the ideal functionality the values that are reconstructed during the protocol execution. This allows the simulator to reconstruct the values towards the adversary, thereby enabling the simulation of the break statements too. Further, note that the point at which protocol breaks discloses the number of allegations that can be revealed if any. Else, the loop breaks after a public number of iterations(equals maxths). In either case, escrows always learn this information depending on if they reveal/not reveal a set of allegations. Hence, it is not a breach of privacy. □

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

$\mathcal{F}_{\text{Mat}}$ interacts with the escrows in $\mathcal{P}$ and the ideal world malicious adversary $\mathcal{S}_{\mathcal{A}}$.

– Receive as input the shares of the following from the parties: a_{new}, $\{a_1, \ldots, a_N\}$, and all the entries p_i in P.

– Reconstruct a_{new}, $\{a_1, \ldots, a_N\}$ and P using received shares of honest parties.

– (**Step 1**) Check if there exists entry p_i in P such that $p_i.\text{pid}$ matches $a_{\text{new}}.\text{pid}$. If found, reduce $a_{\text{new}}.t$ by $p_i.\text{ac}$.

– Include a_{new} as the $N + 1^{th}$ allegation.

– (**Step 2**) Consider all subsets of matching allegations to a_{new} and check if any of them satisfy their reveal criteria. Let $\mathcal{S}$ denote such a set, if found.

– If no such set is found send *ignore* message to escrows and terminate.

– Else (**Step 3**) consider all those matching allegations in $\{a_1, \ldots, a_N\}$ but not in $\mathcal{S}$. For each such allegation a_i, update its threshold to $a_i.t - |\mathcal{S}|$.

– (**Step 4**) Check if there exists a record p_i such that $p_i.\text{pid} = a_{\text{new}}.\text{pid}$. If such a entry is found, update $p_i.\text{ac}$ by adding $|\mathcal{S}|$ to it. Else, create a new entry in P as $(\text{pid}, \text{ac}) = (a_{\text{new}}.\text{pid}, |\mathcal{S}|)$ and send message ("New P entry") to all escrows.

– Send the index i for all allegations a_i in $\{a_1, \ldots, a_N\}$ that are now included in $\mathcal{S}$ to the escrows. Delete these entries from the list of all allegations.

Figure 12: Ideal functionality for matching

B.2 Simulation for Shield protocol

We now prove the security of the Shield protocol.

Theorem B.3. *Let* VRF *be a secure VRF protocol, let* MAC *be a secure MAC, and let the employed signature scheme be strongly existentially unforgeable. Then our* Shield *protocol securely realizes* $\mathcal{F}_{\mathsf{Shield}}$ *(Fig. 2) in the computational setting.*

Proof. We provide the description of a simulator $\mathcal{S}_{\mathcal{A}}$ designed to simulate the view of the malicious adversary $\mathcal{A}$ in the real-world. We assume $\mathcal{A}$ can corrupt at most t escrows out of n and any number of users. We begin with steps followed by $\mathcal{S}_{\mathcal{A}}$.

The initialization phase involves interaction only among the escrows. $\mathcal{S}_{\mathcal{A}}$ emulates the distributed key generation[6] to generate the public key $\mathsf{pk_v}$ and secret key $\mathsf{sk_v}$ required to evaluate VRF. It also generates the secret key $\mathsf{sk_m}$ required to evaluate MAC. It sends the corrupt escrow's shares of $\mathsf{sk_v}$, $\mathsf{sk_m}$ to $\mathcal{A}$ while public key is sent in clear. Note that $\mathcal{S}_{\mathcal{A}}$ is aware of all the keys generated so far. The simulator also emulates the CA.

Since the registration and allegation filing phases require interaction between escrows and users, we consider the following two cases:

Case 1: Honest user and corrupt escrow

Upon receiving the message ("Register", c, ID) from $\mathcal{F}_{\mathsf{Shield}}$, $\mathcal{S}_{\mathcal{A}}$ generates maxalg new public keys $\mathsf{pk}_1^{\mathsf{U}}, \ldots, \mathsf{pk}_{\mathsf{maxalg}}^{\mathsf{U}}$ for a user U, a random r and provides the secret-shares of these, corresponding to the corrupt escrow, to $\mathcal{A}$. $\mathcal{S}_{\mathcal{A}}$ then participates in the distributed computation of the VRF on the received public keys and the computation of MAC on r to generate secret-shares of the VRF on the supplied public keys and shares of uid, respectively . This computation is simulated by running simulation steps with respect to Π_{vrf} and Π_{mac}, which outputs random shares of VRF on the public keys and random shares of uid. $\mathcal{S}_{\mathcal{A}}$ simulates generation and reconstruction of $\mathsf{mac}_i^{\mathsf{U}} = \mathsf{MAC}(\mathsf{sk_m}, \mathsf{pk}_i^{\mathsf{U}})$ for $i \in \{1, \ldots, \mathsf{maxalg}\}$ towards $\mathcal{A}$ ($\mathcal{S}_{\mathcal{A}}$ can simulate this since it has the key $\mathsf{sk_m}$ and $\mathsf{pk}_i^{\mathsf{U}}$). So far the view generated by $\mathcal{S}_{\mathcal{A}}$ is indistinguishable from $\mathcal{A}$'s real world view.

Upon receiving message ("Allege") from $\mathcal{F}_{\mathsf{Shield}}$ regarding the attempt to file a new allegation, $\mathcal{S}_{\mathcal{A}}$ chooses a pk^{U} which has been authenticated in the registration steps and chooses random shares for each component in the allegation, signs these under pk^{U} and sends it to $\mathcal{A}$. On receiving the message about duplicity check from $\mathcal{F}_{\mathsf{Shield}}$, $\mathcal{S}_{\mathcal{A}}$ emulates $\mathcal{F}_{\mathsf{Dup}}$ (the ideal functionality for Π_{Dup}) accordingly and proceeds to the matching phase. Similarly, depending on the message received from $\mathcal{F}_{\mathsf{Shield}}$ for the matching phase, $\mathcal{S}_{\mathcal{A}}$ emulates $\mathcal{F}_{\mathsf{Mat}}$ (ideal functionality for Π_{Mat}). Finally, to simulate revealing of an allegation depending on the user identifier ID received from $\mathcal{F}_{\mathsf{Shield}}$, $\mathcal{S}_{\mathcal{A}}$ picks a pk that was filed with respect to ID. Since $\mathcal{S}_{\mathcal{A}}$ knows the key $\mathsf{sk_m}$, it simulates steps of MAC computation such that it leads to reconstructing $\mathsf{MAC}(\mathsf{sk_m}, \mathsf{pk})$ towards $\mathcal{A}$. The allegation components received from $\mathcal{F}_{\mathsf{Shield}}$ are reconstructed towards $\mathcal{A}$. Observe that in all the above steps, since $\mathcal{S}_{\mathcal{A}}$ follows simulation steps of the underlying protocols, the indistinguishability of the simulation follows from the indistinguishability of the underlying simulations.

Case 2: Corrupt user and corrupt escrow

During registration, $\mathcal{A}$ sends the proof of ID, c obtained from the CA for a corrupt user U to $\mathcal{S}_{\mathcal{A}}$. It also sends the honest escrows' shares of the maxalg public keys $\mathsf{pk}_1^{\mathsf{U}}, \ldots, \mathsf{pk}_{\mathsf{maxalg}}^{\mathsf{U}}$ and a random r to $\mathcal{S}_{\mathcal{A}}$. If the proof of ID is invalid or the shares are inconsistent, $\mathcal{S}_{\mathcal{A}}$ sends $\perp$ to $\mathcal{A}$. Else, it sends ("Register", c, ID) to $\mathcal{F}_{\mathsf{Shield}}$ from the corrupted alleger's ID. Since $\mathcal{S}_{\mathcal{A}}$ knows the shares with respect to all honest escrows, it can reconstruct the underlying values ($\mathsf{pk}_i^{\mathsf{U}}$ and r). $\mathcal{S}_{\mathcal{A}}$ then participates in the distributed computation of the VRF and MAC on the keys and on r, submitted by $\mathcal{A}$, to generate shares of the VRF on the public keys and shares of uid. This computation is simulated by running simulation steps with respect to Π_{vrf} and Π_{mac}. Further, $\mathcal{S}_{\mathcal{A}}$ simulates generation and reconstruction of $\mathsf{mac}_i^{\mathsf{U}} = \mathsf{MAC}(\mathsf{sk_m}, \mathsf{pk}_i^{\mathsf{U}})$, for $i \in \{1, \ldots, \mathsf{maxalg}\}$ towards $\mathcal{A}$.

While filing an allegation, $\mathcal{A}$ sends honest escrow's shares of the allegation components to $\mathcal{S}_{\mathcal{A}}$, who checks if the submitted pk^{U} is valid and if had not been used earlier. If the check fails, $\mathcal{S}_{\mathcal{A}}$ sends $\perp$ to $\mathcal{A}$. Else, $\mathcal{S}_{\mathcal{A}}$ determines the ID with which pk^{U} is registered (recall that $\mathcal{S}_{\mathcal{A}}$ is able to do it since it can reconstruct all pk^{U}'s submitted during registration) and connects to $\mathcal{F}_{\mathsf{Shield}}$ on ID's channel.

Following this, on receiving the message about duplicity check from $\mathcal{F}_{\mathsf{Shield}}$, $\mathcal{S}_{\mathcal{A}}$ emulates $\mathcal{F}_{\mathsf{Dup}}$ accordingly and proceeds to the matching phase. Similarly, depending on the message received from $\mathcal{F}_{\mathsf{Shield}}$ for the matching phase, $\mathcal{S}_{\mathcal{A}}$ emulates $\mathcal{F}_{\mathsf{Mat}}$. On receiving a message from $\mathcal{F}_{\mathsf{Shield}}$ to reveal an allegation filed by a corrupt user U with identity ID, $\mathcal{S}_{\mathcal{A}}$ does the following. Since $\mathcal{S}_{\mathcal{A}}$ knows the key $\mathsf{sk_m}$, it simulates steps of Π_{mac} such that it leads to reconstructing $\mathsf{MAC}(\mathsf{sk_m}, \mathsf{pk}^{\mathsf{U}})$ towards $\mathcal{A}$, where pk^{U} is the key used by the corrupt user for filing the allegation. The allegation components received from $\mathcal{F}_{\mathsf{Shield}}$ are reconstructed towards $\mathcal{A}$. Observe that in all the above steps, since $\mathcal{S}_{\mathcal{A}}$ follows simulation steps of the underlying protocols, the indistinguishability of the simulation follows from the indistinguishability of the underlying simulations. □

[6] A distributed key generation functionality outputs $[\![\cdot]\!]$-shares of a secret key $\mathsf{sk_v}$, and the public key $\mathsf{pk_v}$, on clear, to the escrows. A secure protocol for this can be realized in the underlying MPC setting.