IACR Communications in Cryptology
ISSN 3006-5496, Vol. 1, No. 2, 43 pages.

https://doi.org/10.62056/ay11fhbmo

Check for updates

PACIFIC

Privacy-preserving automated contact tracing featuring integrity against cloning

Scott Griffy and Anna Lysyanskaya

Brown University, United States

Abstract. To be useful and widely accepted, automated contact tracing schemes (also called exposure notification) need to solve two seemingly contradictory problems at the same time: they need to protect the anonymity of honest users while also preventing malicious users from creating false alarms. In this paper, we provide, for the first time, an exposure notification construction that guarantees the same levels of privacy and integrity as existing schemes but with a fully malicious database (notably similar to [ACK+21]) without special restrictions on the adversary. We construct a new definition so that we can formally prove our construction secure. Our definition ensures the following integrity guarantees: no malicious user can cause exposure warnings in two locations at the same time and that any uploaded exposure notifications must be recent and not previously uploaded. Our construction is efficient, requiring only a single message to be broadcast at contact time no matter how many recipients are nearby. To notify contacts of potential infection, an infected user uploads data with size linear in the number of notifications, similar to other schemes. Linear upload complexity is not trivial with our assumptions and guarantees (a naive scheme would be quadratic). This linear complexity is achieved with a new primitive: zero knowledge subset proofs over commitments which is used by our "no cloning" proof protocol. We also introduce another new primitive: set commitments on equivalence classes, which makes each step of our construction more efficient. Both of these new primitives are of independent interest.

1 Introduction

In 2020, the COVID-19 virus spread across the world claiming over 5 million lives in two years [Org24]. Countries have responded with a number of measures such as: distributing masks and vaccines, requiring testing, and restricting travel. One of these measures is contact tracing. Contact tracing is the process of discovering who a person interacted with while they were contagious with a virus. This can help inform others so that they can quarantine themselves to stop the spread of the virus. Contact tracing can be performed by interviewing patients who have been exposed to the virus and notifying those who the patient claims to have come into contact with. Automated contact tracing (also known as automated exposure notification) skips the need for an in-person interview, instead, relying on devices to track users' locations and then notify them if they've come into contact with an infected person. This makes the contact tracing process more efficient allowing contact tracing to scale to cover a nation's population. Unfortunately, automated contact tracing often requires officials to record the locations of users, thus opening up the possibility for governments and organizations to use this data for malicious purposes. Privacy concerns also affect adoption [HMM+21, TN21] and thus, schemes with strong privacy guarantees also make contact tracing more effective. Since 2020, many schemes

E-mail: scott_griffy@brown.edu (Scott Griffy), anna_lysyanskaya@brown.edu (Anna Lysyanskaya)

Received: 2024-04-08 Accepted: 2024-06-03

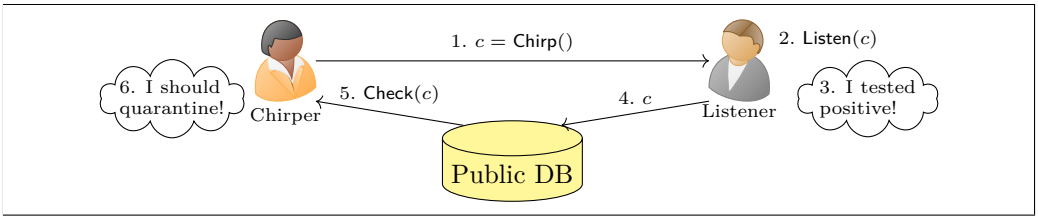

Figure 1: A basic contact tracing scheme

have been proposed [TPH+20, GA20, CKL+20, ACK+21, GKK20] to allow people to use contact tracing schemes (notifing others and checking if they've been exposed) while remaining anonymous. A problem emerges when anonymity is added to a contact tracing scheme as malicious actors could use the anonymity provided by the scheme to create fake outbreaks in targeted areas or at events [GKK20]. The fake contacts could cause users to quarantine themselves when they have not actually been exposed to the virus. Automating contact tracing allows such an attack to cause damage on a large scale. This leaves us with the question: Can a contact tracing scheme provide anonymity while simultaneously ensuring that malicious users cannot abuse the anonymity to create fake exposures?

1.1 Automated contact tracing

To give context to our discussion in the rest of this introduction, we first describe a basic automated contact tracing scheme. This basic contact tracing scheme uses a person's phone to broadcast messages over bluetooth (we'll call these messages "chirps"). Then, if a user tests positive for COVID-19, they upload all of the chirps they heard (or some function of the chirps) to a public database which others can check to see if they were exposed to the virus. We show this process in Figure 1 which shows 6 steps. In steps 1 and 2, a user sends a chirp (c) to nearby listeners. A user later then realizes they've been infected and uploads the chirps they heard (or a function of the chirps) to a database (steps 3 and 4). Other users then check the database (step 5) and if they find that some threshold of the chirps in the database belong to them, they should quarantine and take a viral test to ensure they don't spread the virus.

In a naive scheme, these chirps can simply be random strings and can be uploaded as-is to the database. Because other users can check if their chirps are in the database, this will correctly notify users who were in contact with others. This naive implementation suffers from a multitude of problems, but gives the basic blueprint for a contact tracing scheme. In this naive scheme, malicious users could upload chirps heard from multiple devices to create a fake outbreak, or inspect the database to link the chirps to bluetooth messages they heard, thus potentially learning who was infected or learning someone's location. To provide privacy and integrity in our construction, we need to ensure that chirps become unlinkable to any user or interaction when uploaded to the database, while at the same time requiring uploaders to prove that they haven't acted maliciously.

Upload-what-you-heard Our naive scheme in Figure 1 follows the "upload-what-you-heard" model used by each scheme in [CKL+20], where users upload chirps that they heard. Some existing schemes instead use a "upload-what-you-sent" model where instead, users upload chirps that they sent and users instead check the database for chirps they heard. This model is used by DP-3T, Google and Apple's contact tracing scheme (sometimes referred to as "GAPPLE"), and ReBabbler [TPH+20, GA20, CKL+20]. Upload-what-you-sent models can be more efficient (allowing for constant-sized uploads with respect

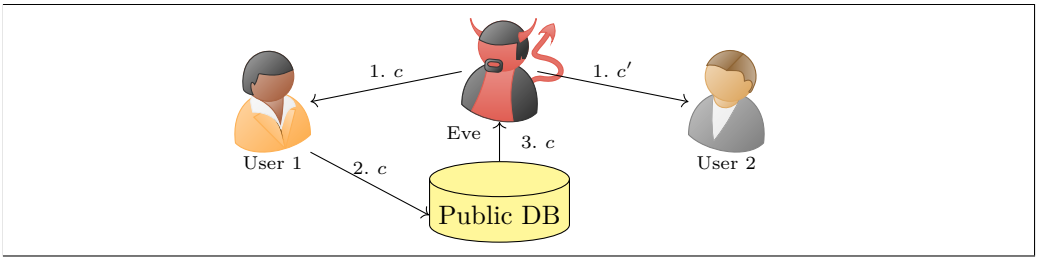

Figure 2: Upload privacy attack

to the number of chirps heard). These models necessarily sacrifice privacy [CKL+20] (with a malicious database). Intuitively, this is because, if an upload is only dependent on messages that the uploader sent, and chirps are broadcasted with no response, any user that can check the database can try different subsets of chirps they heard against the database and use the resulting number of exposures to link uploads to specific chirps. Thus, any privacy-preserving contact tracing scheme (that keeps chirp complexity as a single broadcast) must use the upload-what-you-heard model.

1.2 Attacks on contact tracing schemes

We'll now discuss a few schemes and attacks that are related to contact tracing schemes. There is a much larger body of work on contact tracing which we reference here: [RAC+20, CFG+20, BRS20, BDH+21, PR21, ABIV23, Vau20b, RBS21, WL20, CBB+20, GVhP+22, RBS20, TSS+20, Tra20]. To keep the discussion concise, we'll only highlight a few schemes and attacks in this section.

Privacy attacks There has been a lot of work attacking the privacy of contact tracing schemes [Vau20a, ABIV23, CKL+20]. Some examples are the "chirp attack", the "matrix attack" (also known as the "upload privacy attack"), the "Sybill attack", and the "Brutus attack". The chirp attack involve collecting chirps and linking them together to track users. Our basic scheme in Figure 1 is secure against chirp attacks as each chirp is randomly generated. The matrix attack considers a malicious database that uses data that users uploaded to de-anonymize them (also known as the "upload privacy attack"). The Sybill and Brutus attacks are more advanced and attack countermeasures of the upload privacy attack and so we'll explain them after an example upload privacy attack in the next paragraph.

Example of an upload privacy attack In Figure 2, we show an example attack on privacy. In this figure, the adversary broadcasts chirps to two different users (step 1). Then, we see that user 1 finds they are infected and uploads to the database in step 2. The adversary can then match what they see in the public database with the chirps they broadcasted to learn that it is user 1 that was infected and not user 2. Ideally, we'd like to avoid this type of attack to keep the uploader anonymous.

One way to mitigate upload privacy attacks is to ensure the database only indicates whether a user should quarantine or not, having users learn nothing more, even when the user is colluding with a malicious database. This countermeasure can be circumvented with a Sybill attack where a malicious adversary creates many fake identities to chirp in different locations and thus learns more fine-grained information by learning which of their fake users is recommended to quarantine. Any countermeasure to the Sybill attack would involve registering users to ensure they cannot create many fake identities. Registering

users introduces a new attack which involves an authority using the mapping between registrations and users to de-anonymize users (dubbed the Brutus attack).

In our paper, we'll formalize definitions that address these attacks which we call "chirp privacy" and "upload privacy".

While not applicable to our paper, a common real-world attack [Vau20a] in the literature is against the DP-3T scheme [TPH+20]. The DP-3T scheme diverges from our basic model in Figure 1 and instead follows the upload-what-you-sent model (described in Section 1.1). In DP-3T, a user uploads a single seed which generates all of their chirps they broadcasted during the last week. This opens up an attack (to which all upload-what-you-sent schemes are subject to) where a malicious database owner listens to chirps during that week in different locations and reconstructs a users' locations for the last week.

Integrity attacks Vaudenay [Vau20a] and Avitabile et al. [AFV21] consider the idea that an automated contact tracing scheme could be used to launch a sort of "terrorist attack" where a large area such as a city could be falsely notified of being in contact with an infected person thus needlessly causing them to quarantine. The authors [Vau20a] also propose a solution to this attack. Gennaro et al. [GKK20] explain how a more targeted attack could create fake outbreaks in politically partisan neighborhoods during the weeks leading up to an election to sway the outcome. By targeting a specific neighborhood known to vote a certain way with fake exposures, a malicious attacker could cause those voters to needlessly quarantine during the election, potentially preventing them from reaching their voting location. We can see that our basic scheme in Figure 1 is susceptible to these attacks as the server does not do any checking to ensure that uploaders are honest. Thus, a malicious user could disperse listening devices in areas across a city and upload all chirps they heard.

Another attack is where a malicious device replays and relays chirps to other users. This is known as the "chirp replay/relay attack" [CKL+20]. A chirp replay/relay attacker can listen to a chirp, then duplicate it at another time/place to fool honest users into believing that the replayed chirp was honestly created and including it in their upload, thus falsely exposing an honest user.

Another potential attack on existing schemes is the "upload replay/delay attack." This is partially described in [ACK+21]. In this attack, a malicious user listens to a chirp, then uploads the chirp multiple times and/or waits to upload the chirp at a later time. Reusing a single chirp for multiple exposures could falsely increase the severity of contacts, cause needless quarantines months after the real exposure, or overwhelm the system with false notifications to mask real notifications. This attack becomes more severe when uploaders are anonymous and existing schemes are vulnerable to this attack [CKL+20].

The most novel attack on integrity that we solve in this paper is the "clone attack" which we discuss in the next paragraph.

Clone attacks The mitigation to any integrity attack involves registering users so that the server can verify that the user is acting honestly during chirping and uploading. In this paper, we'll refer to the authority registering users as the "registration party". Many schemes [CKL+20, BCK+20] leverage a registration party in order to provide privacy guarantees, such as imposing "upload limits", ensuring no user uploads too many times per week. Even with upload limits, an adversary could falsely expose attendees at single event. While upload limits stop a user from uploading a massive number of notifications, it does not stop an adversary from using a single device to concentrate a week's worth of notifications into a few hours. This type of attack could be used to target a rally or a political convention which takes place in a shorter time (hours) rather than the period where an honest user would normally upload data for (days). This attack could be more appealing for an adversary to suppress votes as they would only need to attend a single

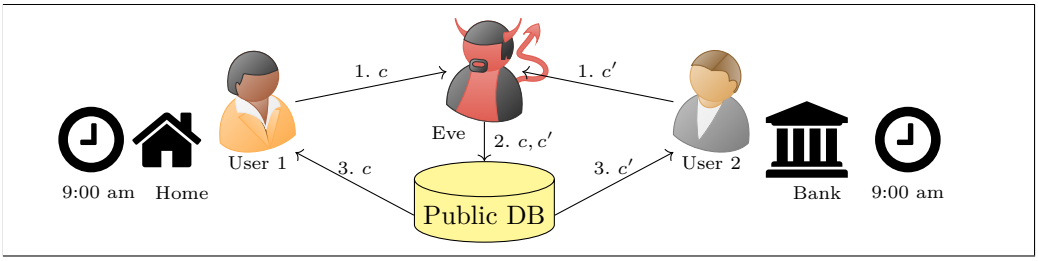

Figure 3: Clone attack

event where they know all attendees will be voting for one candidate, rather than patrolling a neighborhood for weeks to find enough contacts to upload. In order to blanket an event with notifications, an attacker would need to clone their registered device in order to use the contact tracing scheme in multiple locations simultaneously, thus we call it the "clone attack". We describe the clone attack in Figure 3 where Eve (the adversary) is able to clone her device to listen in two locations and upload chirps heard by both devices to notify users in different locations at the same time. This is the most novel attack that our construction aims to prevent. In step 1 of this figure, the adversary listens for chirps c and c' in two different locations at the same time (user 1 is at home and user 2 is at the bank). In step 2 the adversary uploads these chirps, which then expose users in step 3. To our knowledge, the only other scheme that solves this attack while also providing privacy is [ACK+21]. To achieve this notion, [ACK+21] assumes that adversarial devices cannot communicate with each other, which is a strong assumption that our construction avoids.

1.3 Contributions

In this work, we define the requirements for a system that provides the functionality of a contact tracing scheme while simultaneously providing privacy for users and integrity for notifications. We call such a system a ***p**rivacy-preserving **a**utomated **c**ontact trac**i**ng scheme **f**eaturing **i**ntegrity against **c**loning* (shortened to "PACIFIC"). We also introduce a new construction, "ProvenParrot," which satisfies this definition. Our definition addresses all the privacy attacks mentioned in this introduction. To make our scheme efficient, we introduce two primitives: set commitments on equivalence classes and zero knowledge subset proofs over commitments. We use zero knowledge subset proofs over commitments to construct a *no-cloning* proof protocol. By "cloning" we mean the malicious cloning of keys to multiple devices. Preventing the cloning of devices has been studied in previous work [CHK+06].

Threat model and assumptions We operate in a model where we assume an honest registration party. Our integrity definitions are impossible if the adversary has unrestricted access to the registration secret key as then they can register any number of users, so an honest registration party is a necessary assumption for the security we provide. We assume a fully malicious database owner who can corrupt users to learn more information or broadcast malicious chirps. We also assume a semi-honest health authority who verifies that uploaders are infected. By "semi-honest" we mean the adversary can passively attempt to use any messages they see to de-anonymize users, but otherwise acts honestly (i.e. follows our protocol). We note that this is a strong threat model as many schemes leak information when there is collusion between devices and the database [RBS21, ACK+21, GA20, TPH+20].

Out of scope problems Our security definition does not capture attacks where the database deletes uploaded data. Preventing this attack would require a semi-honest database. Our security definition also does not capture an attack where the database owner creates snapshots to learn batches of information uploaded from a single user. Preventing this attack would require users to mix their data with others' data before uploading it to the server. This could potentially be added on top of our scheme in future work using a mix net [Cha81, HM20]. Like many other contact tracing schemes [ABIV23, CKL+20], we do not consider privacy attacks on the BLE protocol itself, e.g., using power analysis or linking a bluetooth identifier. Bluetooth protocols (like BLE) often include metadata which is prohibitive to any privacy-preserving scheme performed over bluetooth [BLS19]. Unless phone makers add an option to remove this metadata, any scheme's chirp privacy will be broken [ABIV23], but we note that even with this limitation, our scheme still retains upload privacy, since uploading does not rely on bluetooth. It is a separate problem to verify that users really have COVID, which we trust the health authority to do. Any data uploaded by users to the database is implicitly signed by the health authority and the uploader.

Time and location We rely on location data to determine if users have been in contact with one another. Like in [CKL+20], our scheme can utilize location-specific "measurements" like GPS data, background noise, nearby wifi networks, or application-specific beacons. In this case, the measurements are hashed to one value which represents the location of the chirper. These measurements are used in addition to GPS data since GPS isn't always reliable and using unpredictable data in the area also provides some integrity against users who might try to broadcast chirps to an area without being present there.

Tying attacks to resources It's impossible to prevent a user from infecting themselves and going to an event to expose many people there. A real world adversary could also bribe those with COVID to include extra chirps in their upload [AFV21]. Thus, instead of entirely stopping these attacks, we need to instead limit the amount of damage that cheating adversaries can do by tying the effectiveness of their attacks to real-world costs such as paying for a phone or bribing "COVID mules" to upload maliciously crafted batches. This is why we must have an honest registration party, to enforce that registration is costly.

Comparison to other works In Table 1 we compare our work with similar schemes. Because in DP-3T, the database learns all chirps that uploaders broadcasted, the database must be honest to achieve privacy. We can see that our scheme achieves clone protection without any non-standard assumption about the adversary. Instead, our construction requires bilinear pairings (BP) in order to provide this privacy guarantee. Specifically, our construction uses mercurial signatures [CL19] and subset commitments [FHS19] which both use BP (reviewed in Section 2). By "non-coordination" in the "special assumptions" column, we mean that the scheme assumes that an adversary can't communicate quickly between their devices. By "DH KEX" in the "chirp efficiency" column, we mean the scheme uses a Diffie-Hellman key exchange. Each scheme achieves chirp privacy (i.e. that an adversary seeing two chirps cannot tell if they're from the same user) but we contrast these schemes in the "upload privacy" column by determining if they require an honest database to provide upload privacy. In the "integrity" column, "clone protection" means the scheme ensures that users cannot upload chirps that were collected in two locations at the same time. "Registration" means that only authorized users can upload, thus preventing some integrity attacks. A scheme with clone protection implies that it also has registration. Linear/constant upload complexity is in respect to the number of notifications uploaded.

Table 1: Comparison of contact tracing schemes.

Scheme	Special assumptions	Upload privacy	Integrity	Chirp efficiency	Upload complexity
Pronto-C2 [ACK+21]	Non-coordination	Malicious DB	Clone protection	DH KEX	Linear
CertifiedClever-Parrot [CKL+20]	None	Malicious DB	Registration	BP	Linear
DP-3T [TPH+20]	None	Honest DB	Registration	PRF	Constant
PACIFIC (This paper)	None	Malicious DB	Clone protection	BP	Linear

Paper roadmap The paper is structured with preliminaries in Section 2, new definitions in Section 3, and new constructions in Section 4. We prove our construction secure in Appendix C. We define our scheme, PACIFIC, in Section 3.1, defining the protocol and our definition of integrity and privacy including security games. We then introduce the definition of set commitments on equivalence classes (CoECs) in Section 3.2 which we construct in Section 4.2. We then present our construction, named "ProvenParrot", in Section 4.1 which meets our PACIFIC security definitions from Section 3.1. Our construction relies on CoECs as well as a "no-cloning" relation which is described alongside the ProvenParrot construction. A proof system for this relation is constructed in Section 4.3.1. Our no-cloning proof system relies on zero knowledge subset proofs over commitments, which we construct in Section 4.3.2. We give more details on preliminaries in Appendix A. A proof of security for our construction is given in Appendix C.1. We also prove our other constructions (CoECs, the no-cloning protocol, and zero knowledge subset proofs over commitments) in Appendices C.2, C.3, and C.4 respectively.

2 Notation and preliminaries

By $(m, *) \in S$ we mean there is a tuple in set S such that the first element of the tuple is m and the second element is another value which could be anything. $\{(m, *) \in S : A(m)\}$ is the set of all tuples in S with m as their first element meeting condition A. Applying a function, f, onto a set of inputs results in a set of outputs. I.e. if $f : X \to Y$, then for $S \subseteq X$, $f(S) = \{f(s) : s \in S\}$. If a set is given to an adversary, it gives the adversary no information about the ordering of the elements in the set (one can imagine a set as a vector that is shuffled whenever a challenger gives it to an adversary). We use $r \leftarrow_{\$} S$ to denote a random choice from a set. For a set (or vector), S, the operation: $k * S$ or S^k defines a new set (or vector) with with k multiplied or exponentiated over each element of the set. By "PPT $\mathcal{A}$" we mean that $\mathcal{A}$ is a probabilistic polynomial-time algorithm. If we have a vector of values, $\mathbf{M}$ (which we denote with bold font), we'll reference the i-th element of that vector with: $\mathbf{M}[i]$. We denote the size of a set as $\#S$. We use $(a, b) \sim (a', b')$ to indicate that these two distributions are indistinguishable to any PPT $\mathcal{A}$ where a, b, a', b' are random variables. We use λ to denote the security parameter and use *negl* to denote a negligible function.

2.1 Cryptographic bilinear pairings

A bilinear pairing [GPS08] is a set of groups, $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$, along with a pairing function, e where $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. We call $\mathbb{G}_T$ the "target group" and call $\mathbb{G}_1$ and $\mathbb{G}_2$ the "source groups". In this work, we use Type III pairings, which means that there is no efficient, non-trivial homomorphism between $\mathbb{G}_1$ and $\mathbb{G}_2$. The pairing function is efficiently computable and has a bilinearity property such that if $\langle P \rangle = \mathbb{G}_1$ and $\langle \hat{P} \rangle = \mathbb{G}_2$, then for $a, b \in \mathbb{Z}_p^*$, $e(P^a, \hat{P}^b) = e(P, \hat{P})^{ab}$. Our groups satisfy $|\mathbb{G}_1| = |\mathbb{G}_2| = |\mathbb{G}_T| = p$ for some prime p. In our pairing groups, the Diffie-Hellman assumptions hold in the related groups, such that for $a, b \leftarrow_\$ \mathbb{Z}_p$, $(P^a, P^b, P^{ab}) \sim (P^a, P^b, P^c)$. and given (P^a, P^b) it is difficult to compute P^{ab}.

2.2 Commitments

A commitment scheme is composed of three functions, $\mathsf{Setup}, \mathsf{Commit}$, and Open where Setup takes in a security parameter λ and outputs some public parameters pp, Commit takes in some mesage m and outputs a commitment c along with some opening information o, and Open takes in a commitment, a message, and the opening information and determines if the commitment opens to the given message. We require both the hiding and binding properties for our commitments. A hiding commitment scheme implies that no adversary can distinguish between commitments to different messages. A binding commitment scheme means that no adversary can open a single commitment to two different messages. For brevity, we omit the formal definitions of foundational primitives like commitments and instead refer the reader to [KL14].

2.3 FHS Commitments [FHS19]

A set-commitment scheme from [FHS19] has the functions: $\mathsf{Setup}(1^k, 1^s) \to pp, \mathsf{Commit}(pp, S) \to (C, O), \mathsf{Open}(pp, C, S, O) \to \{0, 1\}, \mathsf{OpenSubset}(pp, C, S, O, T) \to W$, $\mathsf{VerifySubset}(pp, C, T, W) \to \{0, 1\}$ where S is a set of attributes (of size s), C is a commitment, O is an opening, T is a set of attributes such that $T \subset S$, and W is a witness. FHS commitments have the properties: binding, hiding, and subset soundness. Binding and hiding function exactly as regular commitments described above, except that hiding challenges the adversary to distinguish commitments to two sets S_1 and S_2 and allows the adversary to see witnesses for subsets which wouldn't trivially distinguish the sets i.e. seeing a witness that the challenge commitment ($C = \mathsf{Com}(S_b)$) contains some value that is not in the other set (S_{1-b}). Subset soundness ensures that an adversary cannot produce a witness that a commitment is committed to a subset, T, while also opening the commitment to a set, S, that does not contain T. We formally define these properties in Appendix A.3.

2.4 Non-interactive zero knowledge proofs of knowledge (NIZKs)

A NIZK scheme ($\Phi = (\mathsf{S}, \mathsf{P}, \mathsf{V})$) allows a prover to prove knowledge of a witness that satisfies a verifiable relationship R without revealing any further information. We use the following notation to describe relations where A is some condition: $R((\mathsf{witness}), (\mathsf{statement})) = 1$ iff $A(\mathsf{witness}, \mathsf{statement}) = 1$. If a NIZK is extractable, a witness can be extracted using some secret information or extra power (e.g. in the random oracle model and/or with some black-box access to the prover). To define NIZKs, we use a definition of correctness, zero knowledge and extractability. These definitions assume the scheme uses a hash function as a random oracle.

Definition 1 (NIZK completeness)**.** A NIZK scheme, $\Phi = (\mathsf{S}, \mathsf{P}, \mathsf{V})$, is complete if, given a random oracle, H, the following holds: For all λ, $(w, x) \in \mathsf{R}$, $pp \in \mathsf{S}(1^\lambda)$, $\Pr[\mathsf{V}^{\mathsf{H}}(x, \pi) =$

$1] \geq 1 - negl(\lambda)$ where $\pi \leftarrow \mathsf{P}^{\mathsf{H}}(w, x)$.

Definition 2 (NIZK zero-knowledge)**.** A NIZK scheme, $\Phi = (\mathsf{S}, \mathsf{P}, \mathsf{V})$, is zero-knowledge under the random oracle model if, for all PPT $\mathcal{A}$, $pp \leftarrow \mathsf{S}(1^\lambda)$, $\exists$ simulators SimS, SimP, and SimH with shared state such that if $pp_{\mathcal{S}} \leftarrow \mathsf{SimS}(1^\lambda)$, then

$$|\Pr[\mathcal{A}^{\mathcal{O}^{\mathsf{SimP}},\mathcal{O}^{\mathsf{SimH}}}(pp_{\mathcal{S}}) = 1] - \Pr[\mathcal{A}^{\mathsf{P},\mathsf{H}}(pp) = 1]| \leq negl(\lambda)$$

where $\mathcal{O}^{\mathsf{SimP}}$ ensures that the adversary has given a valid statement and witness, then gives the statement to SimP and returns the simulated proof to the adversary. These oracles also ensure that both SimP and SimH can receive and update the simulator's state.

Definition 3 (NIZK simulation-extractability)**.** A NIZK scheme, $\Phi = (\mathsf{S}, \mathsf{P}, \mathsf{V})$, is simulation-extractable if, for all PPT $\mathcal{A}$, $\exists$ a simulator SimP, random oracle SimH, and extractor, $\mathcal{E}$ with black-box access to $\mathcal{A}$ (labeled as $\mathsf{BB}(\mathcal{A})$) such that, when $pp \leftarrow \mathsf{SimS}(1^\lambda)$, then:

$$\Pr\left[\mathsf{V}(x,\pi) = 1 \wedge (w,x) \notin \mathsf{R} \quad : \quad \begin{array}{l}(x,\pi) \leftarrow \mathcal{A}^{\mathcal{O}^{\mathsf{SimP}},\mathcal{O}^{\mathsf{SimH}}}(pp);\\ w \leftarrow \mathcal{E}^{\mathsf{BB}(\mathcal{A})}(pp);\end{array}\right] \leq negl(\lambda)$$

In Appendix A.1, we define and describe a protocol to prove arbitrary relations between discrete logs of elements in a cyclic prime order group. We then use this protocol implicitly with Camenisch-Shoup notation, e.g., $NIZK[a : h = g^a]$.

2.5 Mercurial Signatures

The first three functions of a mercurial signature scheme ($\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Verify}$) act as a regular digital signature scheme [GMR88]. But a mercurial signature scheme has additional functions ($\mathsf{ConvertSK}, \mathsf{ConvertSig}, \mathsf{ConvertPK}, \mathsf{ChangeRep}$) which allow for signatures, messages, and public keys to be randomized. Randomized signatures remain verifiable for their randomized messages under randomized public key. Signatures and public keys can be randomized independently of each other (e.g. a signature can be randomized while remaning verifiable under the original public key). This allows for the creation of privacy-preserving schemes e.g. if a signature is converted using $\mathsf{ConvertSig}$, no PPT adversary can determine which signature it was randomized from, known as "origin-hiding". A similar property holds for randomized public keys and messages with $\mathsf{ConvertPK}$ and $\mathsf{ChangeRep}$) (known as "public key class-hiding" and "message class-hiding". The $\mathsf{ConvertSK}$ function allows a signer to compute a new signature under a randomized public key.

The unforgeabiliy definition for mercurial signatures is similar to traditional digital signatures, but holds over *equivalence classes*. This means that forgeries only count in the unforgeability game if the adversary outputs a message outside of any equivalence class which they have already queried signatures for. Equivalence classes allow for messages to be randomized while still retaining their underlying message. Mercurial signatures define equivalence classes using a relation R_M. An equivalence class is denoted as $[M]_{\mathsf{R}_M}$ so that the equivalence classes of two messages M and M' are in the same equivalence class (noted as: $[M]_{\mathsf{R}_M} = [M']_{\mathsf{R}_M}$) if $(M, M') \in \mathsf{R}_M$. We call a message such as M a representative of the equivalence class $[M]_{\mathsf{R}_M}$, which is the set of all representatives of that equivalence class. Public keys similarly have equivalence classes denoted by $[pk]_{\mathsf{R}_{pk}}$. The construction of mercurial signatures in [CL19] defines a message as any ℓ-sized vector of group elements $M = \{M_1, ..., M_\ell\} \in \mathbb{G}_1^\ell$ (where ℓ is a parameter that can be tuned and $\mathbb{G}_1$ is a source group of a bilinear pairing described in Section 2.1). Two messages M, M' are in the same equivalence class if $\exists \rho$ such that $M^\rho = M'$ for some $\rho \in \mathbb{Z}$ (each group element in the vector is exponentiated by ρ). We see that in this case, the discrete logs between messages remains the same (i.e., for M, M' such that $[M]_{\mathsf{R}_M} = [M']_{\mathsf{R}_M}$, $\log_{M_i}(M_{i+1}) = \log_{M'_i}(M'_{i+1})$. Public keys are identical to messages except in the second

source group of the bilinear pairing that [CL19] uses: $pk = \{\hat{X}_1, ..., \hat{X}_\ell\} \in \mathbb{G}_2^\ell$. We formally define mercurial signatures in Appendix A.2.

We also need a special function, $\mathsf{Recognize}(sk, pk)$, which operates on public keys from a mercurial signature scheme, determining if they are in the corresponding equivalence class for a given secret key. This function does not exist in the literature, but is clear to construct after realizing that the holder of a secret key knows the discrete log between elements for any representation of their public key.

We discuss mercurial signatures further, providing formalizations, in Appendix A.2.

2.6 Verifiable random functions

Verifiable random functions (introduced by [MRV99]) produce a pseudorandom output on a public input, while verifiably using the secret part of an asymmetric key pair as the key for the PRF. For example, given the output of a PRF, $Y = \mathsf{PRF}_{sk}(x)$, and a Schnorr public key, $pk = P^{sk}$, one can be efficiently convinced by a proof, π (created by the holder of the secret key, sk), that Y was computed on sk and x (without knowing sk). There are many well established constructions of VRFs [DY05, CHK+06]. We will label the computation of these proofs using Camenisch-Stadler notation, e.g. $\pi_Y \leftarrow NIZK[sk : Y = \mathsf{PRF}_{sk}(x) \wedge pk = P^{sk}]$ and $b \leftarrow \mathsf{VerifyVRF}(pk, x, Y, \pi_Y)$. This x can be hidden inside a commitment as well, while maintaining the proof, e.g. $NIZK[sk, x, O : \mathsf{Open}(C, x, O) = 1 \wedge Y = \mathsf{PRF}_{sk}(x) \wedge pk = P^{sk}]$. We also describe VRFs using the keys from mercurial signatures in Appendix A.2.1.

3 Definitions

3.1 PACIFIC Definitions

In this section, we provide our definition of PACIFIC. The functions that make up a PACIFIC scheme are described in Definition 4.

Usage At a high-level, our PACIFIC scheme functions similar to Figure 1 where users repeatedly chirp and listen and then upload when they test positive. Users broadcast chrips at a set interval on the order of seconds or minutes. This interval could coincide with the interval in which a device maintains the same bluetooth MAC address before sampling a new random MAC address. In order to ensure that users cannot violate integrity, users must register their devices to receive a certificate which they use in the chirp function. Some trusted party such as a government agency acts as the registration party. This party creates a registration key pair using RegPartyKeyGen and users trust this public key to verify certificates. Users generate a key pair via UserKeyGen and give their public key to the registration party for registration. Users who test positive for the virus would be verified to upload by a health authority. We model this process by assuming that any uploaders truely have been infected. The infected user would then proceed to compute a "batch" of "notifications" generated by computing Notify on the chirps that they heard while they were possibly infectious. They then upload this batch to a public database so that other users can check if they were exposed using: VerifyBatch and CountExposures. If a user counts a number of exposures higher than some threshold (e.g. indicating they were in contact with infected users long enough to possibly be infected) then they are recommended to quarantine. Users are only meant to upload at most once per "epoch." This epoch should model the length of time that users are infectious before they test positive, so it should be much larger than the interval between chirps, spanning days or weeks, whereas a chirp "interval" (the time between chirps) is measured in seconds or minutes. In our scheme, a database, labeled DB, is simply a set of batches. A database owner receives batches and appends them to the database.

Definition 4 (A PACIFIC scheme)**.**
A privacy-preserving automated contact tracing featuring integrity against cloning scheme consists of a set of algorithms: Setup, RegPartyKeyGen, UserKeyGen, RegisterUser, Chirp, Listen, Notify, VerifyBatch, and CountExposures.

- $\mathsf{Setup}(1^\lambda, e) \to (pp)$: The parameter generation function takes as input: a security parameter, 1^λ, and an epoch length, e, and outputs public parameters: pp which includes an epoch function, Epoch, which is $t \mapsto \lfloor t/e \rfloor$.
- $\mathsf{RegPartyKeyGen}(pp) \to (sk_{rp}, pk_{rp})$: The registration party key generation function takes the public parameters as input and outputs a public/private key for the registration party that users trust to ensure integrity of the scheme. The registration party's key will be used to sign users' public keys.
- $\mathsf{UserKeyGen}(pp, pk_{rp}) \to (sk_{\mathcal{U}}, pk_{\mathcal{U}})$: The user key generation function generates a user's public/private keys, which they use to register as well as sign chirps and uploads.
- $\mathsf{RegisterUser}(pp, sk_{rp}, pk_{\mathcal{U}}) \to (cert)$: This function takes the registration party's keys and registers a user's public key resulting in a certificate which allows the user to chirp and upload.
- $\mathsf{Chirp}(pp, sk_{\mathcal{U}}, pk_{rp}, cert, t, l) \to (\mathsf{c})$: The chirp function outputs a chirp using a user's secret key and certificate. This chirp corresponds to the given time and location: (t, l).
- $\mathsf{Listen}(pp, sk_{\hat{\mathcal{U}}}, pk_{rp}, t, l, \mathsf{c}) \to (0 \text{ or } 1)$: The listen function verifies that a chirp is fresh (matches the given t, l and is not a replay) and valid (signed by the registration party pk_{rp}). After verifying, the user stores the chirp (and metadata) for a potential upload later.
- $\mathsf{Notify}(pp, sk_{\mathcal{U}}, S, d) \to (B)$: The Notify function creates a batch of notifications which indicate to other users that they should quarantine. The function accepts a set of chirps, S. These notifications are related to chirps that were heard within the epoch, d.
- $\mathsf{VerifyBatch}(pp, pk_{rp}, DB, B, d) \to (0 \text{ or } 1)$: This ensures that the batch of notifications includes interactions in a single epoch, d, and was uploaded by a registered user that hasn't already uploaded for epoch d. This function also ensures that clone protection was not violated.
- $\mathsf{CountExposures}(pp, sk_{\mathcal{U}}, pk_{rp}, DB, d) \to (\perp \text{ or } \lambda)$: This function allows any user to check a database to determine how many contacts they've had with infected users.

PACIFIC **security definitions** In the rest of this section, we'll describe *clone integrity*, *chirp privacy*, and *upload privacy* in Defs. 6, 7, and 8 respectively. Our clone integrity definition also serves as our correctness definition as it ensures that the correct number of exposures is counted. In these definitions, the adversary will interact with a challenger through a number of oracles which we describe in Def. 5. These oracles allow the adversary to create honest users (RegisterHonest), create corrupted users (RegisterCorrupt), send their own chirps to honest users (SendChirp), request chirps from honest users (RecvChirp and HonestInteraction), have honest users upload batches (HonestUpload), upload malicious batches (Upload), and control the time (IncrementTime). We describe these oracles in Def. 5. These oracles share some global state. This global state consists of a map from handles to honest secret keys ($\mathsf{HU}_{sk}(\cdot)$), a map from handles to certificates ($\mathsf{Hcert}(\cdot)$), a current time t_{now}, and lists of chirps sent by the adversary (SC) or received by the adversary (RC) which contain tuples of the form: (chirper secret key, listener secret key, time, location). The handles for the maps $\mathsf{HU}_{sk}(\cdot)$ and $\mathsf{Hcert}_{sk}(\cdot)$ can be thought of as integers. We use the notation HU_{sk} (without an input) to represent the entire set of honest secret keys and use similar notation for Hcert.

Summary of integrity game In the integrity game (Game 1), we give the adversary access to the oracles defined in the set $\mathcal{O}_{\mathsf{int}}$. After the adversary exits, we are left with a resulting global state that records all the chirps sent and batches uploaded. The challenger then uses this global state to create a set of possible interactions (PI) that we expect to be in the database due to correctness, assuming the adversary uploads every interaction they could've been a part of. The integrity definition requires that after the game, the interactions extracted from the database (EI) satisfy three properties: (1) that users must count no more exposures from the database than what the extracted set indicates (Equation 1), (2) that the extracted interactions are a subset of those possible interactions (in Equation 2), and (3) that the extracted interactions do not violate clone protection (in Equation 3) (i.e. no uploader was in different places at the same time). Using the extractors in this way ensures that *the truth (that defines what the users draw their counted exposures from) maps to some subset of the possible interactions that doesn't violate clone protection.* Thus, there exists an adversary that could produce this exact set of counted exposures simply by acting as a number of honest users equal to the number of registrations the adversary makes. This ensures that no user can do more damage than if they were to purchase devices and use those devices like an honest user would. This is a strong guarantee and gives us upload replay/delay protection as well as clone protection since any attacks would require the adversary to act dishonestly.

Summary of privacy games To prove the privacy of a PACIFIC scheme, we need two simulators: $\mathcal{S}_{\mathsf{c}}$ for simulating chirps, and $\mathcal{S}_B$ for simulating batches. We describe the functions that call these simulators in Def. 5. In both chirp privacy and upload privacy, we simulate chirps in the same way, using $\mathsf{RecvChirp}^{sim}$.

In the chirp privacy game (but not upload privacy), we allow the adversary to create a corrupted registration party, ensuring that even a malicious registration party cannot de-anonymize using only their chirps. This means the challenger needs to call $\mathsf{RegisterHonest}^{mal}$ instead of having the challenger register honest users itself. The challenger also extracts the adversary's secret registration key to be used in the $\mathsf{RecvChirp}^{sim}$ oracle. In our chirp privacy game (Game 2), we challenge the adversary to distinguish between a simulator and a real chirp. We allow the chirp simulator ($\mathcal{S}_{\mathsf{c}}$) to know the real time and location, but critically not the identity of the honest user. Instead, the simulator is given a freshly generated key and certificate, independent of the user that the adversary has selected to chirp. We allow the time and location to leak since this chirp is broadcasted locally, so any nearby adversary would know this information.

To define upload privacy we present Game 8, where (in addition to using a simulator for chirps) the upload privacy game uses a simulator for batches uploaded by honest users, $\mathsf{HonestUpload}^{sim}$. Intuitively, an adversary will learn the size of the batch, along with the result of calling CountExposures using each of their registered keys. Thus, the simulator, $\mathcal{S}_B$, accepts a freshly generated uploader key and certificate along with a set of adversarial keys (with the associated number of times the user accepted a chirp from this adversarial key), as well as a number of random keys and certificates to pad the batch to the correct size. The batch simulator returns a batch, B. The simulated batches are created independent of the uploader's identity, as well as with random times and locations. Thus the returned batches are unlinkable to any time, user, or location. Passing the adversarial keys to the simulator is necessary to ensure the adversary counts the correct number of exposures. But we omit all honest keys, times and locations. The adversary can learn this data on an honest batch by computing CountExposures with each of his or her corrupted identities. This is because of correctness: the batch must reveal the number of exposures corresponding to each of the corrupted users. But, critically, *the amount of information that the adversary gets through the simulated honest upload oracle scales with the number of corrupted users that the adversary creates.* This ensures that the adversary

must expend resources by registering devices to get any advantage in the privacy game, thus limiting attacks. We also ensure cloned devices can be detected and excluded from the database.

Definition 5 (Oracles for PACIFIC security games)**.**

- $\mathsf{RegisterHonest}(1^\lambda) \to (i, pk_{\mathcal{U}})$: **Register a new honest user.** Generate a user key pair: $(sk_{\mathcal{U}}, pk_{\mathcal{U}}) \leftarrow \mathsf{UserKeyGen}(pp, pk_{rp})$. and the user's certificate: $cert \leftarrow \mathsf{RegisterUser}(pp, sk_{rp}, pk_{\mathcal{U}})$. Return a handle for this honest user, i and the public key, $pk_{\mathcal{U}}$.
- $\mathsf{RegisterCorrupt}(pk_{\mathcal{U}}) \to (cert)$: **Register a corrupted user.** Return a certificate on $pk_{\mathcal{U}}$: $cert \leftarrow \mathsf{RegisterUser}(pp, sk_{rp}, pk_{\mathcal{U}})$.
- $\mathsf{IncrementTime}(1^\lambda) \to (\perp)$: **Increment the current time.** Set the global time variable, $t_{\mathsf{now}} = t_{\mathsf{now}} + 1$. If, after the increment, we enter a new epoch, reset the database by forgetting all previous batches.
- $\mathsf{RecvChirp}(i, l) \to (\mathsf{c})$: **Receive a chirp from an honest user.** Takes a user handle, i (received by the adversary from $\mathsf{RegisterHonest}$), and a location, l. If this user already chirped at the time, t_{now} (the current time), abort. Otherwise, compute the chirp: $\mathsf{c} \leftarrow \mathsf{Chirp}(pp, \mathsf{HU}_{sk}(i), pk_{rp}, \mathsf{Hcert}(i), t_{\mathsf{now}}, l)$ where $\mathsf{HU}_{sk}(i)$ is honest user i's secret key, and $\mathsf{Hcert}(i)$ is the honset user i's certificate. Add this chirp to the list of received chirps, RC. Return c.
- $\mathsf{SendChirp}(i, l, \mathsf{c}) \to (\perp)$: **Send a chirp to an honest user.** If user i already listened to a chirp at a different location at this same time, abort. Compute: $\mathsf{Listen}(pp, \mathsf{HU}_{sk}(i), pk_{rp}, t_{\mathsf{now}}, l, \mathsf{c})$ and, if this succeeds, add this chirp to the list of sent chirps, SC.
- $\mathsf{HonestInteraction}(i, j, l) \to (\perp)$: **Have two honest users interact.** Run $\mathsf{c} \leftarrow \mathsf{RecvChirp}(i, l)$ and $\mathsf{SendChirp}(j, l, \mathsf{c})$.
- $\mathsf{HonestUpload}(i) \to (B)$: **Have an honest user compute and upload a batch.** Compute a batch computed by the honest user: $B = \mathsf{Notify}(pp, pk_{rp}, \mathsf{HU}_{sk}(i), S, d_{\mathsf{now}})$ where S is the set of all chirps they heard and verified during the game and $d_{\mathsf{now}} = \mathsf{Epoch}(t_{\mathsf{now}})$. Add this batch to the database, DB, and return B to the adversary.
- $\mathsf{Upload}(B) \to (\perp)$: **A batch is uploaded to the database.** Run $\mathsf{VerifyBatch}(pp, pk_{rp}, DB, B, \mathsf{Epoch}(t_{\mathsf{now}}))$ where DB is the current set of batches. If this outputs 1, add B to the database, DB.
- $\mathsf{RecvChirp}^{sim}(i, l) \to (\mathsf{c})$: First generate a new key pair: $(sk_{\mathcal{U}}, pk_{\mathcal{U}}) \leftarrow \mathsf{UserKeyGen}(pp, pk_{rp})$. Then register this user: $cert \leftarrow \mathsf{RegisterUser}(sk_{rp}, pk_{\mathcal{U}})$. Then call the chirp simulator ($\mathcal{S}_{\mathsf{c}}$) with this new user's keys, certification, the current time, t_{now}, and given location, l, returning the simulated chirp.
- $\mathsf{RegisterHonest}^{mal}(1^\lambda) \to (i, pk_{\mathcal{U}})$: Generate a key pair, $(sk_{\mathcal{U}}, pk_{\mathcal{U}}) \leftarrow \mathsf{UserKeyGen}(pp, pk_{rp})$. Call the adversary with $pk_{\mathcal{U}}$ to receive $cert$. $sk_{\mathcal{U}}, cert$ now constitute a new honest user i. Return the handle for this honest user (i) and the public key, $pk_{\mathcal{U}}$.
- $\mathsf{HonestUpload}^{sim}(i) \to (B)$: First, the challenger computes a new uploader keypair, $(sk_{\mathcal{U}}, pk_{\mathcal{U}}) \leftarrow \mathsf{UserKeyGen}(pp, pk_{rp})$ and registers this user: $cert_{\mathcal{U}} \leftarrow \mathsf{RegisterUser}(pp, sk_{rp}, pk_{\mathcal{U}})$. Next, the challenger extracts a set of secret keys of corrupted users (CU_{sk}) with an extractor: $\mathsf{CU}_{sk} = \mathcal{E}_{sk}(\mathsf{CU})$ where CU is the corrupted public keys provided by the adversary to $\mathsf{RegisterCorrupt}$. Next, the challenger constructs a set containing the secret keys of corrupted users along with the number of times that user chirped to each honest user: $K_{\mathcal{A}} = \{(sk, k) : (sk \in \mathsf{CU}_{sk}) \wedge (k = \#\{(sk, \mathsf{HU}_{sk}(i), *, *) \in \mathsf{SC}\}) \wedge (k > 0)\}$ where SC is the "sent chirps" during the game and has tuples of the form: $(sender, listener, time, location)$. Next, the challenger generates and registers a random secret key for each chirp that user i accepted from another honest user, yielding a set $K_H = \{(sk_j, cert_j)\}_{j \in [h]}$ where $\forall j \in [h], (sk_j, pk_j) \leftarrow \mathsf{KeyGen}(pp)$, $cert_j \leftarrow \mathsf{RegisterUser}(pp, sk_{rp}, pk_j)$ and h is the number of interactions between this user and other honest users: $h = \#\{(sk, \mathsf{HU}_{sk}(i), *, *) \in \mathsf{SC}$ s.t. $sk \in$

$\mathsf{HU}_{sk}\}$. The challenger then passes $sk_{\mathcal{U}}$, $cert_{\mathcal{U}}$, $K_{\mathcal{A}}$, and K_H, to the simulator, $\mathcal{S}_B$, and returns the simulated batch to the adversary.

Game 1 (Clone integrity game)**.** *The challenger runs* $(sk_{rp}, pk_{rp}) \leftarrow \mathsf{RegPartyKeyGen}(1^\lambda)$ *and* $\mathcal{A}^{O_{\mathsf{int}}}(pp, pk_{rp}, 1^\lambda)$*, where* $O_{\mathsf{int}} = \{\mathsf{RegisterHonest}, \mathsf{RegisterCorrupt}, \mathsf{RecvChirp}, \mathsf{SendChirp}, \mathsf{HonestInteraction}, \mathsf{IncrementTime}, \mathsf{HonestUpload}, \mathsf{Upload}\}$*. When* $\mathcal{A}$ *exits, the challenger uses the resulting global state (including the chirps sent by honest users,* SC*) from the game to determine if the adversary won, described below:*

- *The challenger computes the set of* ***possible interactions*** *(*PI*) which contains each chirp emitted by honest users (in the final epoch) paired with each corrupted secret key registered as well as any interactions between honest users. The size of this set is:* (# *honestly emitted chirps*) × (# *corrupted users*) + (# *honest interactions*)*. This set also includes every possible interaction between two malicious users. The tuples in this set include a chirper secret key, a listener secret key, a time, and location in that order. The times and locations are read from the chirps accumulated during the game in the set,* SC*. The secret keys are extracted from public keys during registration using an extractor,* $\mathcal{E}_{sk}$*.*
- *The challenger computes the set of* ***extracted interactions*** *(*EI*) from the database (using the extractor* $\mathcal{E}_{DB}$*) which would notify an honest user. The tuples of this set have the same form as* PI*.*

We now check conditions and output 1 *if any fail, indicating that the adversary wins. Otherwise, output* 0*.*

1. ***Correct exposure count.*** *Ensure the extracted interactions match the exposures counted by honest users:*
$$\forall i \in \mathsf{HU},\\ \mathsf{CountExposures}(pp, \mathsf{HU}_{sk}(i), pk_{rp}, DB, t_{\mathsf{now}}) = \#\{(*, \mathsf{HU}_{sk}(i), *, *) \in \mathsf{EI}\} \quad (1)$$
Where HU *is the set of honest users and* $\mathsf{HU}_{sk}(i)$ *is the secret key of the honest user,* i*.*
2. ***Database contains a subset of possible interactions.*** *Ensure these extractions are within the set of possible interactions:* $\mathsf{EI} \subseteq \mathsf{PI}$ (2)
3. ***Clone protection.*** *Ensure that no uploader was in two locations at the same time:*
$$\nexists \big((*, sk_{\mathcal{U}}, t, l), (*, sk_{\mathcal{U}}{}', t', l')\big) \in \mathsf{EI} \text{ s.t. } \big((sk_{\mathcal{U}} = sk_{\mathcal{U}}{}') \wedge (t = t') \wedge (l \neq l')\big) \quad (3)$$

Game 2 (Chirp privacy game)**.** *Run* $(pk_{rp}, st) \leftarrow \mathcal{A}(pp, 1^\lambda)$ *and use the* pk_{rp} *for oracles in the game. Extract* $sk_{rp} = \mathcal{E}_{pk_{rp}}(pk_{rp})$ *for registering new users in the simulator. Sample a random bit,* $b \leftarrow_{\$} \{0, 1\}$*. If* $b = 0$*, run* $b' \leftarrow \mathcal{A}^{O_{real}}(pp, 1^\lambda, st)$*, otherwise, if* $b = 1$*, run* $b' \leftarrow \mathcal{A}^{O_{sim}}(pp, 1^\lambda, st)$*, where:* $O_{real} = \{\mathsf{RegisterHonest}^{mal}, \mathsf{RecvChirp}, \mathsf{SendChirp}, \mathsf{HonestInteraction}, \mathsf{IncrementTime}\}$ *and* $O_{sim} = \{\mathsf{RegisterHonest}^{mal}, \mathsf{RecvChirp}^{sim}, \mathsf{SendChirp}, \mathsf{HonestInteraction}, \mathsf{IncrementTime}\}$*. The adversary wins if* $b = b'$*. The* HonestInteraction *function uses* $\mathsf{RecvChirp}^{sim}$ *instead of* RecvChirp *if* $b = 1$*.*

Game 3 (Upload-privacy game)**.**

Run $(sk_{rp}, pk_{rp}) \leftarrow \mathsf{RegPartyKeyGen}(1^\lambda)$*. Sample a random bit,* $b \xleftarrow{\$} \{0, 1\}$*. If* $b = 0$*, run* $b' \leftarrow \mathcal{A}^{O_{real}}(pp, pk_{rp}, 1^\lambda)$*, otherwise, if* $b = 1$*, run* $b' \leftarrow \mathcal{A}^{O_{sim}}(pp, pk_{rp}, 1^\lambda)$*, where:* $O_{real} = \{\mathsf{RegisterHonest}, \mathsf{RegisterCorrupt}, \mathsf{RecvChirp}, \mathsf{SendChirp}, \mathsf{HonestInteraction}, \mathsf{IncrementTime}, \mathsf{HonestUpload}\}$ *and* $O_{sim} = \{\mathsf{RegisterHonest}, \mathsf{RegisterCorrupt}, \mathsf{RecvChirp}^{sim}, \mathsf{SendChirp}, \mathsf{HonestInteraction}, \mathsf{IncrementTime}, \mathsf{HonestUpload}^{sim}\}$*. The adversary wins if* $b = b'$*.*

Definition 6 (Clone integrity)**.** A PACIFIC scheme, Π, has clone integrity if there exists a set of extractors, $\mathcal{E} = \{\mathcal{E}_{sk}, \mathcal{E}_{DB}\}$, such that no PPT adversary can win Game 1 with probability greater than negligible for all valid epoch values.

Definition 7 (Chirp privacy)**.** A PACIFIC scheme, Π, is privacy-preserving with respect to chirps if there exists a set of extractors, $\mathcal{E} = \{\mathcal{E}_{sk}, \mathcal{E}_{DB}\}$, and a simulator, $\mathcal{S}_{\mathsf{c}}$, such that

no PPT adversary has greater than $\frac{1}{2} + negl(\lambda)$ advantage in Game 2 for all valid epoch values.

Definition 8 (Upload privacy)**.** A PACIFIC scheme, Π, is privacy-preserving if there exists a set of extractors, $\mathcal{E} = \{\mathcal{E}_{sk}, \mathcal{E}_{DB}\}$ and a set of simulators, $\mathcal{S} = \{\mathcal{S}_{\mathsf{c}}, \mathcal{S}_B\}$, such that no PPT adversary has greater than $\frac{1}{2} + negl(\lambda)$ advantage in Game 3 for all valid epoch values.

3.2 Set Commitments on equivalence classes (CoECs)

We can already see how mercurial signatures (described in Section 2.5) could be useful to construct a PACIFIC scheme as public key class-hiding allows for keys to be signed and later randomized so that a verifier can only tell that a user is authenticated and not who the user is. Unfortunately, mercurial signatures give no notion of hiding randomized messages if an adversary is allowed to construct the initial message. We define CoECs in this section to remedy this. CoECs are commitments that are hiding and binding across equivalence classes. Traditional commitments ensure that an adversary cannot produce a commitment with openings to distinct messages. CoECs provide a stronger binding property, *class-binding* (in Def. 11), which prevents adversaries from opening two commitments to distinct sets of messages as long as the commitments are in the same equivalence class (i.e. one commitment is a randomization the other). This ensures that even if the adversary randomizes the commitment, it is still binding to the original value. This notion of class-binding for CoECs allows them to compose well with mercurial signatures (discussed in Section 2). Mercurial signatures are unforgeable with respect to an equivalence class. This means that, using a CoEC committed to attributes, a mercurial signature enforces unforgeability of those attributes, while still allowing the commitment to be randomized. We also include a property: *class-hiding* (in Def. 12) which ensures that even if an adversary creates the commitment, it is still hiding after it has been randomized. This allows our PACIFIC construction to be efficient by allowing uploaders to reveal randomized commitments which other users and authorities can verify directly without learning the attributes that the commitments are committed to. Uploaders can then compute proofs over these commitments to enforce clone protection. This scheme uses a relation to describe equivalence classes, which we'll label as R_C. We label the equivalence class of commitments as $[C]_{\mathsf{R}_C}$ for some representative, C. This equivalence class is the set of all representatives which are in the same class as C. In our construction, we ensure that these commitments are compatible with the construction of mercurial signatures in [CL19]. This means that commitments from our CoEC scheme are within the message space of mercurial signatures and the equivalence classes are the same. We define a CoEC scheme in Def. 9. We define class-binding and class-hiding in Definitions 12 and 11. These class-* definitions imply their traditional counterparts (which are defined in Section 2.2). In Section 4.2, we construct a CoEC scheme in Definition 13.

Definition 9 (CoECs scheme)**.**

- $\mathsf{Setup}_{\mathsf{com}}(1^\lambda, s, \mathbb{G}) \to pp$: Initialize the commitment scheme for a number of attributes, s, outputting public parameters, pp.
- $\mathsf{Commit}(pp, M = \{m_0, m_1, ...\}) \to (C, O)$: Commit to a set of attributes, M, where $|M| = s$. Output the commitment, C (which is also a representation of the equivalence class, $[C]_{\mathsf{R}_C}$. Also output the opening information O.
- $\mathsf{RandomizeCom}(pp, C, O; \mu) \to (C', O')$: Randomizes the commitment using randomizer, μ, so that C and C' are unlinkable but still a commitment to the same set.
- $\mathsf{Open}(pp, C, M, O) \to (0 \text{ or } 1)$: Use opening O to verify that C is committed to M.

Definition 10 (CoEC correctness)**.** A commitment scheme is correct if $\forall\ s, \lambda$ and attributes M where $|M| = s$, then given $pp \leftarrow \mathsf{InitializeCoEC}(1^\lambda, s)$, $(C, O) \leftarrow \mathsf{Commit}(pp, M)$, and $(C', O') = \mathsf{RandomizeCom}(C, O)$, the following holds:
$\Pr[\mathsf{Open}(pp, C, M, O) = 1] = 1$
And:
$\Pr[\mathsf{Open}(pp, C', M, O') = 1] = 1$

Definition 11 (CoEC class-binding)**.** A commitment scheme is binding if for all s, λ and for any PPT adversary, $\mathcal{A}$, then given $pp \leftarrow \mathsf{InitializeCoEC}(1^\lambda, s)$, the following probability is negligible:
$\Pr[(C, C', M, M', O, O') \leftarrow \mathcal{A}(pp); \mathsf{Open}(pp, C, M, O) = 1 \wedge \mathsf{Open}(pp, C', M', O') = 1 \wedge M \neq M' \wedge [C]_{\mathsf{R}_C} = [C']_{\mathsf{R}_C}] \leq negl(\lambda)$

Definition 12 (CoEC class-hiding)**.** A commitment scheme is class-hiding if for all s, λ and for any PPT adversary, $\mathcal{A}$, $pp \leftarrow \mathsf{InitializeCoEC}(1^\lambda, s, \mathbb{G})$, $(C, M, O, , C', M', O') \leftarrow \mathcal{A}(pp, 1^\lambda)$ where $\mathsf{Open}(C, M, O) = 1$ and $\mathsf{Open}(C', M', O') = 1$, then $C_0 \leftarrow \mathsf{RandomizeCom}(C)$ is indistinguishable from $C_1 \leftarrow \mathsf{RandomizeCom}(C')$.

We construct CoECs in Section 4.2 and prove them secure in Appendix C.2.

4 Constructions

4.1 PACIFIC construction

We name our construction: "ProvenParrot" as it is similar to CertifiedCleverParrot from [CKL+20] but requires the uploader to *prove* properties of their upload (that their notifications do not violate clone protection, are from the latest epoch, and contain no duplicates). We've split our scheme into 3 figures (Figures 4, 5, and 6) that group similar functions together, specficially describing initialization, interactions, and uploading respectively. We prove that this construction meets Definitions 6, 7, and 8 (which are integrity, chirp privacy, and upload privacy, respectively) in Appendix C.1.

We describe the efficiency of this construction in Appendix B, but to summarize here, we find that our uploads size is $15|\mathbb{G}| + 48n|\mathbb{G}| + 4|\mathbb{Z}_p| + 30n|\mathbb{Z}_p| + 4n|\text{range proof}|$ where n is the number of interactions. This ends up being about five kilobytes per interaction. Our chirps total ten group elements and one element of $\mathbb{Z}_p$ which is less than 400 bytes. Checking the database only requires recognizing a public key in the database, which only requires one multiplication for each entry in the database. Verifying chirps requires two mercurial signature verifications but can be delayed until an upload is required. Computing a chirp requires only elliptic curve multiplication and exponentiation.

Overview of the scheme In Fig. 4, we describe the initialization of the protocol, with the registration party calling RegPartyKeyGen to generate their keys and users calling UserKeyGen to generate keys. The users then interact with the registration party through RegisterUser to receive their certificate. In Fig. 5, we describe both halves of an interaction (Chirp and Listen). Note that chirps do not depend on the listener, so they can be broadcasted. We continue with the Notify, VerifyBatch, and CountExposures functions in Fig. 6 which allow for a user to notify other users they previously interacted with that they should quarantine. This notification is facilitated through the database. In order to register users without allowing the registration party or database to de-anonymize users (as in the Brutus and matrix attacks in Section 1.2) we will use mercurial signatures. We reviewed mercurial signatures in Section 2. Specifically, we use the mercurial signature construction from [CL19] because it uses the same equivalence class for messages as our

$\mathsf{Setup}(1^\lambda) \to pp$

Initialize three schemes:
the mercurial signature scheme:
$pp_{\mathsf{MS}} \leftarrow \mathsf{Setup}_{\mathsf{MS}}(1^\lambda)$
the commitment scheme:
$pp_{\mathsf{com}} \leftarrow \mathsf{Setup}_{\mathsf{com}}(1^\lambda)$
and the no-cloning scheme:
$pp_{\mathsf{clone}} \leftarrow \mathsf{SetupNoCloning}(1^\lambda, 0, 1)$
Output $pp = (pp_{\mathsf{MS}}, pp_{\mathsf{com}}, pp_{\mathsf{clone}})$

$\mathsf{UserKeyGen}(pp, pk_{rp}) \to (sk_{\mathcal{U}}, pk_{\mathcal{U}})$

Generate a mercurial keypair for the user: $(sk_{\mathcal{U}}, pk_{\mathcal{U}}) \leftarrow \mathsf{KeyGen}(pp_{\mathsf{MS}})$.

$\mathsf{RegPartyKeyGen}(pp) \to (sk_{rp}, pk_{rp})$

Generate a mercurial keypair for the registration party: $(sk_{rp}, pk_{rp}) \leftarrow \mathsf{KeyGen}(pp_{\mathsf{MS}})$.
Construct a NIZK proving that sk_{rp} is known: π_{rp}.
Output (sk_{rp}, pk_{rp}) where π_{rp} is implicitly part of the public key.

$\mathsf{RegisterUser}(pp, sk_{rp}, pk_{\mathcal{U}}) \to (cert)$

Sign $pk_{\mathcal{U}}$ with sk_{rp}: $\sigma \leftarrow \mathsf{Sign}(sk_{rp}, pk_{\mathcal{U}})$ and return σ as the certificate, $cert$.
The user also proves that they know $sk_{\mathcal{U}}$

Figure 4: ProvenParrot construction - Initialization

commitments (discussed in Section 4.2). We will use the chains of mercurial signatures (similar to the anonymous credential construction in [CL19]) to allow registered users to sign commitments to the current time and location for chirping, thus preventing Sybil and relay/replay attacks, while also randomizing users' signatures so that they cannot be de-anonymized. We implicitly use mirrored mercurial signature schemes like in [CL19] to allow the registration party to sign user keys while at the same time allowing those user keys to sign further messages. In RegisterUser (Fig. 4), the registration party will use their key (sk_{rp}, pk_{rp}) to sign the user's key $(sk_{\mathcal{U}}, pk_{\mathcal{U}})$, yielding a certificate $(cert)$ with the function Sign. To chirp (function Chirp in Fig. 5) a user will randomize their key and certificate with ChangeRep and ConvertSK, yielding $sk_{\mathcal{U}}{}', pk_{\mathcal{U}}{}', cert'$. We use $\mathbb{Z}_p$ as the randomization space from which ρ (and other randomizers) are drawn from as this is the randomization space used in [CL19]. The user then creates a CoEC to the current time (t), location (l), and epoch (d) along with a random nonce (r) yielding commitment C and opening O. The user then signs this commitment with their randomized mercurial secret key $sk_{\mathcal{U}}{}'$ using Sign, yielding signature σ_{c}. The user then combines all of this information into a "chirp" (c) which they can then broadcast. This creates a chain of keys and signatures $(pk_{rp}, pk_{\mathcal{U}}{}', cert', \sigma_{\mathsf{c}})$ such that a listener can use $cert'$ to ensure that the chirper (known to listener by $pk_{\mathcal{U}}{}'$) is registered and then use σ_{c} to verify that this registered chirper signed the commitment C. The listener can then use the opening information O to verify that this commitment opens to the current time, location, and epoch. The listener then remembers the commitment C, opening O, signature σ_{c}, chirper key $pk_{\mathcal{U}}{}'$, and attributes t, l, d, r for possibly notifying them later in an upload. If a user determines they were infected, they iterate over the keys, commitments, and attributes they heard in the last epoch (week) and randomize them to ensure they cannot be used to de-anonymize the uploader. They also compute a proof over the commitments to prove to the other users that their batch does not include any attacks on integrity such as replay or clone attacks. Other users will then download the database, verify the proofs, and count their number of exposures in the database. This process is described in Fig. 6.

Preventing multiple batches from the same user We want to prevent uploaders from uploading multiple times. Naively, with mercurial signatures, this poses a problem as

$\mathsf{Chirp}(sk_{\mathcal{U}}, t, l) \to \mathsf{c}$

Compute the current epoch:
$d \leftarrow \mathsf{Epoch}(t)$
Randomize credentials and keys:
$\rho \leftarrow_{\$} \mathbb{Z}_p^*$
$(pk_{\mathcal{U}}{}', cert')$
$\quad \leftarrow \mathsf{ChangeRep}(pk_{rp}, pk_{\mathcal{U}}, cert, \rho)$
$sk_{\mathcal{U}}{}' \leftarrow \mathsf{ConvertSK}(sk_{\mathcal{U}}, \rho)$
Commit to time, location, epoch, and a nonce:
$r \leftarrow_{\$} \mathbb{Z}_p^*$
$C, O = \mathsf{Commit}(pp_{\mathsf{Com}}, (t, l, d, r))$
Sign the commitment:
$\sigma_{\mathsf{c}} \leftarrow \mathsf{Sign}(sk_{\mathcal{U}}{}', C)$
$c = (pk_{\mathcal{U}}{}', cert', C, O, \sigma_{\mathsf{c}}, r)$

$\mathsf{Listen}(pk_{rp}, t, l, \mathsf{c}, S) \to S$

Parse c as $(pk_{\mathcal{U}}{}', cert', C, O, \sigma_{\mathsf{c}}, r)$
$d = \mathsf{Epoch}(t)$
Verify commitment and chirper's key
if $\mathsf{Open}(pp_{\mathsf{Com}}, C, (t, l, d, r), O) = 1$
$\quad \wedge\, \mathsf{Verify}(pk_{rp}, pk_{\mathcal{U}}{}', cert') = 1$
$\quad \wedge\, \mathsf{Verify}(pk_{\mathcal{U}}{}', C, \sigma_{\mathsf{c}}) = 1$ **then**
$\quad S = S \cup \{(t, l, d, r, C, O, pk_{\mathcal{U}}{}', \sigma_{\mathsf{c}})\}$

Figure 5: ProvenParrot construction - Interaction

uploaders can simply randomize their pseudonym, potentially allowing them to upload multiple batches in the same epoch. We solve this problem using *verifiable random functions* (VRFs) [MRV99] (reviewed in Section 2). This allows an uploader to compute a pseudorandom function of the current epoch ($d = \mathsf{Epoch}(t_{\mathsf{now}})$) using a VRF key derived from the user's secret key which remains constant across all pseudonyms, $Y = \mathsf{PRF}_{sk_{\mathcal{U}}}(d)$. We ensure that the key used for the VRF across all representations of a user's public key is constant using a technique described in Appendix A.2. This ensures that even if a malicious uploader randomizes their public key to upload another batch in the same epoch, then if the proof π_Y verifies, then Y will match the existing Y for the previously uploaded batch. Thus, if a user only uploads at most once per epoch (as honest users do) the output will look random as this is only time they will never recompute the VRF on this epoch.

Randomizing uploads An honest uploader will randomize their key and certificate in a batch. They will also randomize every chirp they heard to create the list of notifications. This is where we use CoECs which we describe in Section 3.2. CoECs (or set commitments on equivalence classes) are designed to be randomized over the same equivalence classes as mercurial signatures. This ensures that their hiding and binding properties hold when randomized to verify with a similarly randomized signature and public key. Randomizing in this way allows for a checker of the database to recognize their own public key using the function $\mathsf{Recognize}$ and verify their own signature but prevents them from linking the commitment to any time, location, or specific chirp. If we use the mercurial signature construction from [CL19], the message equivalence class will match the equivalence class for our commitments, ensuring that calling $\mathsf{ChangeRep}$ on C has the same effect as calling $\mathsf{RandomizeCom}$ on C.

Preventing integrity attacks The uploader will also create a proof that they did not violate clone protection. Naively, in quadratic size relative to the number of chirps, proving the batch does not contain a clone can be done with a NIZK for each pair of commitments proving that the uploader knows the opening and that each pair of attributes does not have the same time at different locations. We introduce zero knowledge subset proofs over

$\mathsf{Notify}(pk_{rp}, sk_{\hat{\mathcal{U}}}, S, d) \rightarrow B$

Randomize the uploader key:
$\psi \leftarrow_\$ \mathbb{Z}_p$
$(pk'_{\hat{\mathcal{U}}}, cert') \leftarrow \mathsf{ChangeRep}(pk_{rp}, pk_{\hat{\mathcal{U}}}, cert; \psi)$
$(sk'_{\hat{\mathcal{U}}}) \leftarrow \mathsf{ConvertSK}(sk_{\hat{\mathcal{U}}}; \psi)$
Compute a VRF for this epoch:
$Y = \mathsf{PRF}_{sk'_{\hat{\mathcal{U}}}}(d)$
Compute a proof for this VRF:
$\pi_Y = NIZK[sk'_{\hat{\mathcal{U}}} : Y = \mathsf{PRF}_{sk'_{\hat{\mathcal{U}}}}(d)$
$\wedge (pk'_{\hat{\mathcal{U}}}, sk'_{\hat{\mathcal{U}}}) \in \mathsf{KeyGen}(pp)]$
Iterate over heard chirps:
$\forall((t, l, d, r), C, O, pk_{\mathcal{U}}, \sigma_{\mathsf{c}}) \in S :$
Randomize the chirp and key:
$\mu \leftarrow_\$ \mathbb{Z}_p$
$O^* = O * \mu$
$(C^*, \sigma'_{\mathsf{c}}) \leftarrow \mathsf{ChangeRep}(pk_{\mathcal{U}}, C, \sigma_{\mathsf{c}}; \mu)$
$\rho \leftarrow_\$ \mathbb{Z}_p$
$\sigma^*_{\mathsf{c}} = \mathsf{ConvertSig}(pk_{\mathcal{U}}, C^*, \sigma'_{\mathsf{c}}, \rho)$
$pk_{\mathcal{U}}{}^* = \mathsf{ConvertPK}(pk_{\mathcal{U}}{}', \rho)$
Prove properties of the batch:
$\pi_{\mathsf{clone}} = \mathsf{ProveNoCloning}(pp_{\mathsf{clone}},$
$\{C^*_i, O^*_i, (t_i, l_i, d_i, r_i\}_{i\in[|S|]}))$
$S^* = \{C^*_i, pk_{\mathcal{U}i}{}^*, \sigma^*_{\mathsf{c},i}\}_{i\in[|S|]}$
$B = (Y, \pi_Y, pk'_{\hat{\mathcal{U}}}, cert', \pi_{\mathsf{clone}}, S^*)$

$\mathsf{VerifyBatch}(pk_{rp}, DB, B, d_{\mathsf{now}}) \rightarrow e$

Parse B as $(Y, \pi_Y, pk_{\mathcal{U}}{}', cert', \pi_{\mathsf{clone}}, S^*)$
if $\mathsf{Verify}(pk_{rp}, pk_{\mathcal{U}}{}', cert') = 0$
$\vee\ \mathsf{VerifyNoCloning}(pp, B, d_{\mathsf{now}}, \pi_{\mathsf{clone}}) = 0$
then return $\perp$
$\forall B^\dagger \in DB$
Parse $B^\dagger$ as $(Y^\dagger, \pi^\dagger_Y, pk_{\mathcal{U}}{}^\dagger, cert^\dagger, \pi^\dagger_{\mathsf{clone}}, S^\dagger)$
if $Y = Y^\dagger$ **then return** $\perp$

$\mathsf{CountExposures}(pk_{rp}, sk_{\mathcal{U}}, DB, d_{\mathsf{now}}) \rightarrow e$

Iterate through each batch (B)
in the database (DB)
$e = 0$
$\forall B \in DB$
Parse B as $(Y, \pi_Y, pk_{\mathcal{U}}{}', cert', \pi_{\mathsf{clone}}, S^*)$
$\forall(C^*, pk_{\mathcal{U}}{}^*, \sigma^*_{\mathsf{c}}) \in S^*$
if $\mathsf{Recognize}(pp_{\mathsf{MS}}, sk_{\mathcal{U}}, pk_{\mathcal{U}}{}^*)$
$\wedge\ \mathsf{Verify}(pp_{\mathsf{MS}}, pk_{\mathcal{U}}{}^*, C^*, \sigma^*_{\mathsf{c}})$
then $e = e + 1$
return e

Figure 6: ProvenParrot construction - Upload

commitments (zk-SPoCs) in Section 4.3.2 to reduce this to a linear complexity. These subset proofs are used in our "no cloning" proof scheme in Section 4.3.1. The uploader then combines all proofs, signatures, commitments (discarding the openings) and public keys in a batch to send to the database.

While checking the database (in functions VerifyBatch and CountExposures), a user will first verify that the uploader was registered using $\mathsf{Verify}(pk_{rp}, pk'_{\hat{\mathcal{U}}}, cert')$. They will then verify each VRF output, Y to ensure that this matches $pk'_{\hat{\mathcal{U}}}$ and that no identical Y appears in any other batch in the database. They will then verify the proof of clone protection in each batch and count the number of public keys they recognize as their own in the database with verifying signatures on a commitment. This number is exactly the number of times that the user was exposed and will indicate the user's risk factor, possibly suggesting that they quarantine.

As part of our construction, we need a NIZK to prove that a batch is correct. This requires us to prove relations over all of the attributes to which the commitments in the batch are committed to. Specifically, we need to prove that this batch doesn't include two commitments whose attributes indicate the uploader was in different places at the same time. We also need to prove that each nonce is unique and that each time is in the correct epoch. Formally, this relation is defined as:

$$
\begin{aligned}
&R((\{t_i, l_i, d_i, r_i, o_i\}_{i\in[n]}), (d, B = \{C_i\}_{i\in[n]})) = 1 \\
&\quad \text{iff } (\forall i \in [n], \mathsf{Open}(C_i, \{t_i, l_i, d_i, r_i\}, o_i) = 1 \wedge d_i = d) \\
&\quad \wedge (\forall i, j \in [n], (l_i = l_j \vee t_i \neq t_j) \wedge (r_i \neq r_j))
\end{aligned} \tag{4}
$$

where d is the current epoch and size of the batch is $n = |B|$ (the notation for R is described in Section 2.4). This proof system contains a setup function, prove function, and verify function, $\{\mathsf{SetupNoCloning}, \mathsf{ProveNoCloning}, \mathsf{VerifyNoCloning}\}$. We construct this proof system in Section 4.3.

We present the following theorems and prove them in Appendix C.1 along with the sets of simulators and extractors needed for the proofs.

Theorem 1. *The ProvenParrot scheme described in Fig. 4 has clone integrity (meeting Definition 6) assuming that the underlying schemes are secure. Specifically, we assume that the mercurial signature scheme is unforgeable, the NIZKs are simulation-extractable, and the CoECs are binding.*

Theorem 2. *The ProvenParrot scheme described in Fig. 4 is chirp-private (meeting Definition 7) assuming that the underlying schemes are secure. Specifically, we assume that the mercurial signature scheme is class and origin hiding, the NIZKs are zero knowledge, and the CoECs are hiding.*

Theorem 3. *The ProvenParrot scheme described in Fig. 4 is upload-private (meeting Definition 8) assuming that the underlying schemes are secure. Specifically, we assume that the mercurial signature scheme is class and origin hiding, the NIZKs are zero knowledge, and the CoECs are hiding.*

4.2 Set commitments on equivalence classes (CoEC) construction

Here we construct a scheme which satisfies the definitions in Section 3.2.

CoEC scheme intuition In $\mathsf{InitializeCoEC}$, we generate a number of random points (we'll call them "bases"). Then, given a vector of messages, M, a committer will construct a commitment (C) composed of two group elements such that the discrete log between the group elements is dependent on the messages as well as the discrete log of the random points. Because the discrete log between the elements of the commitment is dependent on the discrete log of the bases generated in setup, we will see that an attempt to forge or distinguish commitments reduces to the computational and decisional Diffie-Hellman problems in the proofs in Appendix C.2. Because mercurial signatures use type III bilinear pairings, a hash function exists for at least one of the bilinear groups [GPS08, Cos12]. Thus, in practice, we can create these random bases verifiably using the hash function. Though we can also use a CRS to generate these.

Definition 13 (CoEC construction)**.** A commitment scheme on equivalence classes has the following functions:

- $\mathsf{InitializeCoEC}(1^\lambda, 1^s) \to pp$: Compute pairs of random points: $(A, B_0, B_1, ..., B_{s-1}) \xleftarrow{\$} (\mathbb{G}_1)^{s+1}$. Output these points as pp.
- $\mathsf{Commit}(pp, M) \to (C, O)$: Let $M = \{m_0, m_1, ..., m_{s-1}\}$. Generate a random $\mu \leftarrow \mathbb{Z}_p^*$. Compute a vector of size 2: $C = (C_0, C_1)$, $C_0 = A^\mu$ and $C_1 = \prod_{i=0}^{s-1} {B_{i,1}}^{\mu m_i}$ and $O = \mu$.
- $\mathsf{RandomizeCom}(pp, C, O; \mu)$: $C' = C^\mu$, $O' = \mu * O$. If μ is omitted, it is sampled randomly from $\mathbb{Z}_p$.
- $\mathsf{Open}(pp, C, M, O)$: Compute $C' = \mathsf{Commit}(pp, M; O)$. Ensure that $C' = C$.

We prove Theorem 4 in Appendix C.2.

Theorem 4. *The CoEC scheme in Definition 13 is correct, origin-hiding, binding, and hiding as defined in Definitions 10, 11, 12, and Section 2.2 as long as the DDH and CDH assumptions (from Section 2) holds in the underlying bilinear pairing.*

There may be other constructions of CoECs for other equivalence classes or that provide other functionalities or efficiencies. Messages from [FHS19, CL19] could theoretically be considered a CoEC for high-entropy attributes (similar to how public keys can be considered "commitments" to secret keys). In our PACIFIC construction, our attributes are high-entropy, so we cannot simply use the messages classes from [FHS19, CL19] and instead need this more secure construction.

4.3 ZKP for batch uploads

4.3.1 Clone protection for a set of commitments to vectors of attributes

To construct our scheme such that we can detect clones, we'll need to prove certain properties of the user's uploaded batch, i.e. we need to prove the relation in Eq. 4 in Section 4.1. A batch will include signed commitments to attributes of transactions. These attributes are time, location, epoch, and a random nonce unique to each contact. Labeling these values, a batch will contain commitments to vectors of messages $\{\mathbf{M}_i\}$ such that $\mathbf{M}_i[\mathsf{idx}_\mathsf{t}]$ is the time of the contact, $\mathbf{M}_i[\mathsf{idx}_\mathsf{l}]$ is the location, $\mathbf{M}_i[\mathsf{idx}_\mathsf{d}]$ is the epoch, and $\mathbf{M}_i[\mathsf{idx}_\mathsf{r}]$ is the nonce (this notation is described in Section 2). These indices can be aribtrary integers as long as they are consistent across the scheme. Thus, to prevent cloning, we need to prove that, across a batch, no two attribute vectors in the set have the same time value at a different location value. Proving this ensures that the device was not cloned to be used in two different locations. We also need to prove that the epoch is correct with respect to the current epoch (remember, the epoch is a longer period of time than an interval length and no honest user would upload twice in one epoch). Because the commitments of chirps include the epoch, we do not need to prove computation of the epoch function, $\mathsf{Epoch}(t)$, in zero knowledge, thus avoiding costly range proofs. And lastly, we need to prove that each nonce is distinct (thus, proving that the user did not duplicate any contacts).

Thus, in this section, we construct a proof of the relation in Eq 4.

Proving this relation trivially, we can see that it would take communication quadratic in n as we need to compare each possible pair of attribute vectors. To prove this in linear time, we find that a NIZK scheme of the following relation is useful: $R_\subset = (\{A_i, O_i, L_i\}_{i\in[n]}, \{B_i, P_i, M_i\}_{i\in[m]}), (A_i, B_i)) = 1$ iff $\forall i \in [n], \mathsf{Open}(A_i, L_i, O_i) = 1$ and $\forall i \in [m], \mathsf{Open}(B_i, P_i, M_i) = 1$ and $\forall i \in [n], \exists j \in [m]$ s.t. $L_i = M_j$. The last condition $(\forall i \in [n], \exists j \in [m]$ s.t. $L_i = M_j)$ represents a subset condition with the set $\{L_i\}_{i\in[n]}$ being a subset of $\{M_i\}_{i\in[m]}$. We write this explicitly without the $\subset$ operator for clarity as $\{A_i, O_i, L_i\}_{i\in[n]}$ may contain duplicate L_i's which still need to be proven to be openable. We construct this NIZK in Section 4.3.2, calling the scheme a *zero knowledge subset proof over commitments* (zk-SPoC). We label the functions of the proof scheme for $R_\subset$ as $\{\mathsf{InitZKSPoC}, \mathsf{ProveSubset}, \mathsf{VerifySubset}\}$.

With our description of $R_\subset$, we are ready to construct our NIZK for relation R in Fig. 7.

Theorem 5. *The construction in Fig. 7 is complete with respect to definition 1 and relation R.*

Theorem 6. *The construction in Fig. 7 is zero knowledge with respect to definition 2 and relation R as long as* eqrep *and zk-SPoCs are zero knowledge.*

Theorem 7. *The construction in Fig. 7 is simulation-extractable with respect to definition 3 and relation R as long as* eqrep *and zk-SPoCs are simulation extractable.*

We prove Theorems 5, 6, and 7 in Appendix C.3.

4.3.2 Zero knowledge subset proofs over commitments (zk-SPoCs) construction

In this section we construct a scheme to prove the relation: $R_{\subset} = ((\{A_i, L_i, O_i\}_{i\in[n]}, \{B_i, M_i, P_i\}_{i\in[m]}), (A_i, B_i)) = 1$ iff $\forall i \in [n], \mathsf{Open}(A_i, L_i, O_i) = 1$ and $\forall i \in [m], \mathsf{Open}(B_i, M_i, P_i) = 1$ and $\forall i \in [n], \exists j \in [m]$ s.t. $L_i = M_j$. This is easily accomplished in $n * m$ communication using a generic NIZK OR proof: for every commitment in A, prove that it is equivalent to (committed to the same values as) one of the commitments in B. Because quadratic complexity would be considered inefficient, instead, we construct proofs in time linear to n. To do this, we'll use the FHS commitment scheme from [FHS19] for this scheme as well as the concatenation function (c) from [CHK+06]. This concatenation function maps vectors of attributes to a single message for the FHS commitment. Thus, each "message" in the FHS commitment scheme will represent a vector of attributes. This allows us to construct an FHS commitment to a set of attribute vectors. Following [CHK+06], this concatenation function works by partitioning the message space of the FHS commitment scheme into bits and assigning sets of these bits to the spaces of different attributes. This requires that the bit-length of the FHS message space is the sum of the bit-lengths of all attributes in the vector. We describe this further in Appendix A.3. The proof consists of creating FHS commitments, C and D to our two sets $\{c(L_i)\}_{i\in[n]}$ and $\{c(M_i)\}_{i\in[n]}$ and then doing an efficient subset proof ($\pi_{C\subset D}$) that the set committed to by C is a subset of the set committed to by D. The proof, $\pi_{C\subset D}$, requires only a single group element. But, we still need $O(n)$ proofs to prove that our FHS commitments C and D are correct with respect to $\{A_i\}_{i\in[n]}$ and $\{B_i\}_{i\in[n]}$. We show this scheme in Fig. 8.

We prove Theorems 8, 9, and 10 in Appendix C.4.

Theorem 8. *The construction in Fig. 8 is complete with respect to definition 1.*

Theorem 9. *The construction in Fig. 8 is zero knowledge with respect to definition 2 as long as FHS set commitments are perfectly hiding and* eqrep *(from Section A.1) is zero knowledge.*

Theorem 10. *The construction in Fig. 8 is simulation-extractable with respect to definition 3 as long as* eqrep *is simulation-extractable and FHS set commitments are subset sound.*

5 Acknowledgments

We are very appreciative of the anonymous reviewers for their time and effort in reading and commenting on this work. We would also like to acknowledge Octavio Pérez Kempner for his helpful comments on this paper. Anna Lysyanskaya and Scott Griffy are supported by NSF Awards 2247305, 2154941 and 2154170, as well as funding from the Peter G. Peterson Foundation and Meta.

- $\mathsf{SetupNoCloning}(1^\lambda, pp_{\mathsf{com}}, t, l) \rightarrow pp$: Initialize a zk-SPoC scheme for relation $R_\subset$: $pp_{\mathsf{SPoC}} = \mathsf{InitZKSPoC}(1^\lambda)$. Initialize a VRF scheme: pp_{VRF}. Output $pp = (pp_{\mathsf{VRF}}, pp_{\mathsf{SPoC}}, pp_{\mathsf{com}}, t, l)$.
- $\mathsf{ProveNoCloning}(pp, \{C_i, O_i, \mathbf{M}_i\}_{i\in[n]}) \rightarrow \pi$: We split the computation of this proof up into three separate relations:
  1. To prove the $(\forall i \in [n], \mathsf{Open}(C_i, \{t_i, l_i, d_i, r_i\}, O_i) = 1 \wedge d_i = d)$ part of the relation, R, the prover first proves that they can open all C_i and then proves that each $d_i = d$ in zero knowledge, yielding NIZK proof, π_d.
  2. To prove that $\forall i, j \in [n], (r_i \neq r_j)$, the prover computes a random VRF key pair, $k \leftarrow_\$ \mathbb{Z}_p, K = P^k$ and then computes VRF outputs and proofs: $\forall i \in [n], Y_{r,i} = \mathsf{PRF}_k(r_i)$ and $\pi_{r,i} = NIZK[r_i, O_i : \mathsf{Open}(C_i, \{t_i, l_i, d_i, r_i\}, O_i) = 1 \wedge Y_{r,i} = \mathsf{PRF}_k(r_i)]$.
  3. To prove the relation: $\forall i, j \in [n], (l_i = l_j \vee t_i \neq t_j)$, the prover needs to use zk-SPoCs (described earlier in this section and constructed in Section 4.3.2). The prover first computes commitments to the time and location attributes of vectors in $\{M_i\}$, yielding $\{C'_i, O'_i, \mathbf{M}'_i\}_{i\in[n]}$ such that $\forall i \in [n]$, $\mathbf{M}'_i = (t_i, l_i) \wedge \mathbf{M}_i = (t_i, l_i, d_i, r_i)$. They then prove this relation correct where $\{\mathbf{M}_i, \mathbf{M}'_i\}_{i\in[n]}$ are the messages to which the commitments $\{C'_i, C_i\}_{i\in[n]}$ are committed to. This yields the proof $\pi_{t,l}$ which can be constructed using eqrep described in Appendix A.1. The prover then computes a set of vectors, $\{\mathbf{L}_i\}_{i\in[m]}$, containing all distinct vectors in $\{\mathbf{M}'_i\}_{i\in[n]}$. The prover then pads $\{\mathbf{L}_i\}_{i\in[m]}$ with vectors of random values† so that its length is the same as $\{\mathbf{M}'\}_{i\in[n]}$, resulting in a set of vectors, $\{\mathbf{L}'_i\}_{i\in[n]}$. Commit to $\{\mathbf{L}'\}_{i\in[n]}$, yielding, $\{(B'_i, P'_i) = \mathsf{Com}_{CoEC}(\mathbf{L}'_i)\}_{i\in[n]}$. Compute $\pi_\subset = \mathsf{ProveSubset}(\{C'_i, O'_i, \mathbf{M}'_i, B'_i, P'_i, \mathbf{L}'_i\}_{i\in[n]})$ using the zk-SPoC scheme described in Section 4.3.2. Next, the prover computes a second key pair for the VRF scheme (k', K') and iterates through the commitments, $\{B'_i, \mathbf{L}'_i, P'_i\}_{i\in[n]}$, and computes a VRF on each of the time values in each vector $\mathbf{L}_i$: $\{Y_{t,i} = \mathsf{PRF}_{k'}(\mathbf{L}_i[\mathsf{idx_t}])\}_{i\in[n]}$ along with a proof that the VRF is correct with respect to B_i and K': $\{\pi_{t,i} = NIZK[\mathbf{L}_i, P_i, k' : \mathsf{Open}(B_i, \mathbf{L}_i, P_i) = 1 \wedge Y_{t,i} = \mathsf{PRF}_{k'}(\mathbf{L}_i[\mathsf{idx_t}]) \wedge K' = P^{k'}\}_{i\in[n]}$.

  Finally, the prover outputs all of their proofs along with new commitments, VRF outputs, and public keys: $\pi = (\pi_d, K, \{Y_{r,i}, \pi_{r,i}\}_{i\in[n]}, \pi_{t,l}, \{C'_i, B'_i\}_{i\in[n]}, \pi_\subset, K', \{Y_{t,i}, \pi_{t,i}\}_{i\in[n]})$.
- $\mathsf{VerifyNoCloning}(pp, \{C_i\}_{i\in[n]}, \pi) \rightarrow \{0, 1\}$: Parse π as $\pi = (\pi_d, K, \{Y_{r,i}, \pi_{r,i}\}_{i\in[n]}, \pi_{t,l}, \{C'_i, B'_i\}_{i\in[n]}, \pi_\subset, K', \{Y_{t,i}, \pi_{t,i}\}_{i\in[n]})$. Ensure that $\mathsf{VerifSubset}(pp, \{C'_i\}_{i\in[n]}, \{B'_i\}_{i\in[n]}, \pi_\subset) = 1$. Verify all other proofs in π. Iterate through $\{Y_{r,i}, Y_{t,i}\}_{i\in[n]}$ and verify that $\forall i, j \in [n], (i = j \vee (Y_{r,i} \neq Y_{r,j} \wedge Y_{t,i} \neq Y_{t,j}))$.

† Sampling these random values from an exponential space ensures that there is a negligible probability of collisions with existing values in vectors.

Figure 7: Clone protection scheme

- $\mathsf{InitZKSPoC}(1^\lambda) \to (pp)$: Initialize a commitment scheme with efficient subset openings, pp_{com}. Output $pp = pp_{\mathsf{com}}$.
- $\mathsf{ProveSubset}(pp, \{(A_i, O_i, L_i), (B_i, P_i, M_i)\}_{i\in[n]}) \to (\pi)$:
  1. First, create new FHS commitments, A'_i, B'_i, to $L'_i = c(L_i)$ and $M'_i = c(M_i)$ respectively with openings O'_i, P'_i (where c from [CHK+06] is a function that concatenates a vector of messages into a single message). Compute NIZKs to prove equivalence of these commitments i.e. $\pi' = NIZK[\{L_i, L'_i, M_i, M'_i, O_i, O'_i, P_i, P'_i\}_{i\in[n]} : \forall i \in [n], \mathsf{Open}(A_i, L_i, O_i) = 1 \wedge \mathsf{Open}(B_i, M_i, P_i) = 1 \wedge \mathsf{Open}(A'_i, L'_i, O'_i) = 1 \wedge \mathsf{Open}(B'_i, M'_i, P'_i) = 1 \wedge L'_i = c(L_i) \wedge M'_i = c(M_i)]$. We describe the computation of this concatenation function along with its proof in Appendix A.3.
  2. Compute FHS set commitments, C, D, to each set $\{L'_i\}_{i\in[n]}$ and $\{M'_i\}_{i\in[n]}$, $(C, O_C) = \mathsf{Commit}(\{L_i\}_{i\in[n]})$, $(D, O_D) = \mathsf{Commit}(\{M_i\}_{i\in[n]})$. Duplicates should be removed in this set to ensure that we can prove that C is committed to a subset of the values that D is committed to.
  3. Prove that each A'_i is a commitment to a subset of the set committed to by C. This can be done using the $\mathsf{ProveSubset}_{\mathsf{FHS}}$ function described in Appendix A.3, yielding proofs $\{\pi_{A_i\subset C}\}_{i\in[n]}$. Specifically, for a commitment, A'_i, this proof involves constructing a witness for the $\mathsf{OpenSubset}$ function, $\pi_{W,i} = (\prod_{m\in\{M'_j\}_{j\in[n]}\setminus\{M'_i\}} \hat{P}^{(\alpha-m)})^{O_C/O'_i}$ using the public parameters for the FHS set commitment scheme, $\{\hat{P}^{\alpha^i}\}_{i\in[n]}$. Computing $\pi_{W,i}$ in this way ensures $\pi_{W,i}$ is a valid witness for proving that A_i is committed to a subset of the set committed to by C, i.e. that $e(A'_i, \pi_{W,i}) = e(C, \hat{P})$. The prover then proves knowledge of opening of C. Then, the prover randomizes this witness with a $\mu \leftarrow_\$ \mathbb{Z}_p$, yielding π'_W such that $e(A'_i, (\pi'_W)^\mu) = e(C, \hat{P})$. Then, the prover produces a proof that μ is known, $\pi_{\mu,j}$ and outputs $\pi_{A_i\subset C} = (\pi_{\mu,i}, \pi_{W,i})$.
  4. Prove that C is committed to a subset of the messages committed to by D using $\mathsf{ProveSubset}_{\mathsf{FHS}}$ (described in Appendix A.3), yielding a proof: $\pi_{C\subset D}$, which can be included in the proof.
  5. Prove the relation $NIZK[O_D, \{O_{B'_i}\}_{i\in[n]} : \mathsf{Open}(D, \{M'_i\}_{i\in[n]}, O_D) = 1 \wedge \forall i \in [n], \mathsf{Open}(B'_i, M'_i, P'_i) = 1]$. This can be done using the eqrep protocol described in Appendix A.1. Label this proof as $\pi_{\{B_i\}\approx D}$.
  6. Output all commitments along with their proofs as the subset proof: $\pi_\subset = (C, D, \pi_{\{B_i\}\approx D}, \pi', \{A'_i, B'_i, \pi_{A_i\subset C}\}_{i\in[n]}, \pi_{C\subset D})$.
- $\mathsf{VerifSubset}(pp, \{A_i, B_i\}_{i\in[n]}, \pi) \to (1 \text{ or } 0)$: Verify the zero knowledge proofs in π to ensure that the messages committed to by the A_i values are a subset of the messages committed to by B_i.

Figure 8: Our zk-SPoC construction

References

[ABIV23] Gennaro Avitabile, Vincenzo Botta, Vincenzo Iovino, and Ivan Visconti. Privacy and integrity threats in contact tracing systems and their mitigations. *IEEE Internet Computing*, 27(2):13–19, 2023. Full version: https://eprint.iacr.org/2020/493. doi:10.1109/MIC.2022.3213870.

[ACK+21] Benedikt Auerbach, Suvradip Chakraborty, Karen Klein, Guillermo Pascual-Perez, Krzysztof Pietrzak, Michael Walter, and Michelle Yeo. Inverse-sybil attacks in automated contact tracing. In Kenneth G. Paterson, editor, *CT-RSA 2021*, volume 12704 of *LNCS*, pages 399–421. Springer, Heidelberg, May 2021. doi:10.1007/978-3-030-75539-3_17.

[AFV21] Gennaro Avitabile, Daniele Friolo, and Ivan Visconti. Terrorist attacks for fake exposure notifications in contact tracing systems. In Kazue Sako and Nils Ole Tippenhauer, editors, *Applied Cryptography and Network Security*, pages 220–247, Cham, 2021. Springer International Publishing. doi:10.1007/978-3-030-78372-3_9.

[BBB+18] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 315–334, 2018. doi:10.1109/SP.2018.00020.

[BCK+20] Jean-François Biasse, Sriram Chellappan, Sherzod Kariev, Noyem Khan, Lynette Menezes, Efe Seyitoglu, Charurut Somboonwit, and Attila Yavuz. Trace-Σ: a privacy-preserving contact tracing app. IACR ePrint, 2020. https://eprint.iacr.org/2020/792.

[BD19] Razvan Barbulescu and Sylvain Duquesne. Updating key size estimations for pairings. *Journal of Cryptology*, 32(4):1298–1336, October 2019. doi:10.1007/s00145-018-9280-5.

[BDH+21] Wasilij Beskorovajnov, Felix Dörre, Gunnar Hartung, Alexander Koch, Jörn Müller-Quade, and Thorsten Strufe. Contra corona: Contact tracing against the coronavirus by bridging the centralized–decentralized divide for stronger privacy. In Mehdi Tibouchi and Huaxiong Wang, editors, *Advances in Cryptology – ASIACRYPT 2021*, pages 665–695, Cham, 2021. Springer International Publishing. doi:10.1007/978-3-030-92075-3_23.

[BLS19] Johannes K. Becker, David Li, and David Starobinski. Tracking anonymized bluetooth devices. *PoPETs*, 2019(3):50–65, July 2019. doi:10.2478/popets-2019-0036.

[BRS20] Samuel Brack, Leonie Reichert, and Björn Scheuermann. Caudht: Decentralized contact tracing using a dht and blind signatures. In *2020 IEEE 45th Conference on Local Computer Networks (LCN)*, pages 337–340, 2020. doi:10.1109/LCN48667.2020.9314850.

[CBB+20] Claude Castelluccia, Nataliia Bielova, Antoine Boutet, Mathieu Cunche, Cédric Lauradoux, Daniel Le Métayer, and Vincent Roca. DESIRE: A third way for a european exposure notification system leveraging the best of centralized and decentralized systems. *CoRR*, abs/2008.01621, 2020. URL: https://arxiv.org/abs/2008.01621, arXiv:2008.01621.

[CBC+24] Miranda Christ, Foteini Baldimtsi, Konstantinos Kryptos Chalkias, Deepak Maram, Arnab Roy, and Joy Wang. SoK: Zero-knowledge range proofs. Cryptology ePrint Archive, Paper 2024/430, 2024. https://eprint.iacr.org/2024/430.

[CFG+20] Justin Chan, Dean Foster, Shyam Gollakota, Eric Horvitz, Joseph Jaeger, Sham Kakade, Tadayoshi Kohno, John Langford, Jonathan Larson, Puneet Sharma, Sudheesh Singanamalla, Jacob Sunshine, and Stefano Tessaro. Pact: Privacy sensitive protocols and mechanisms for mobile contact tracing, 2020. URL: https://arxiv.org/abs/2004.03544, arXiv:2004.03544.

[CGH+23] Sofía Celi, Scott Griffy, Lucjan Hanzlik, Octavio Perez Kempner, and Daniel Slamanig. Sok: Signatures with randomizable keys. Cryptology ePrint Archive, Paper 2023/1524, 2023. https://eprint.iacr.org/2023/1524.

[Cha81] David L. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Commun. ACM*, 24(2):84–90, feb 1981. doi:10.1145/358549.358563.

[CHK+06] Jan Camenisch, Susan Hohenberger, Markulf Kohlweiss, Anna Lysyanskaya, and Mira Meyerovich. How to win the clonewars: Efficient periodic n-times anonymous authentication. In Ari Juels, Rebecca N. Wright, and Sabrina De Capitani di Vimercati, editors, *ACM CCS 2006*, pages 201–210. ACM Press, October / November 2006. doi:10.1145/1180405.1180431.

[CKL+20] Ran Canetti, Yael Tauman Kalai, Anna Lysyanskaya, Ronald L. Rivest, Adi Shamir, Emily Shen, Ari Trachtenberg, Mayank Varia, and Daniel J. Weitzner. Privacy-preserving automated exposure notification. Cryptology ePrint Archive, Report 2020/863, 2020. https://eprint.iacr.org/2020/863.

[CL19] Elizabeth C. Crites and Anna Lysyanskaya. Delegatable anonymous credentials from mercurial signatures. In Mitsuru Matsui, editor, *CT-RSA 2019*, volume 11405 of *LNCS*, pages 535–555. Springer, Heidelberg, March 2019. doi:10.1007/978-3-030-12612-4_27.

[Cos12] Craig Costello. Pairings for beginners. Online, 2012. https://www.craigcostello.com.au/s/PairingsForBeginners.pdf.

[CS97] Jan Camenisch and Markus Stadler. Proof systems for general statements about discrete logarithms. Report, Zürich, 1997. Technical Reports D-INFK. doi:10.3929/ethz-a-006651937.

[DY05] Yevgeniy Dodis and Aleksandr Yampolskiy. A verifiable random function with short proofs and keys. In Serge Vaudenay, editor, *PKC 2005*, volume 3386 of *LNCS*, pages 416–431. Springer, Heidelberg, January 2005. doi:10.1007/978-3-540-30580-4_28.

[FHS19] Georg Fuchsbauer, Christian Hanser, and Daniel Slamanig. Structure-preserving signatures on equivalence classes and constant-size anonymous credentials. *Journal of Cryptology*, 32(2):498–546, April 2019. doi:10.1007/s00145-018-9281-4.

[GA20] Google and Apple. Privacy-preserving contact tracing, 2020. https://covid19.apple.com/contacttracing.

[GKK20] Rosario Gennaro, Adam Krellenstein, and James Krellenstein. Exposure notification system may allow for large-scale voter suppression. Real World Crypto, 2020.

[GMR88] Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on Computing*, 17(2):281–308, April 1988.

[GPS08] Steven D. Galbraith, Kenneth G. Paterson, and Nigel P. Smart. Pairings for cryptographers. *Discrete Appl. Math.*, 156(16):3113–3121, sep 2008. doi:10.1016/j.dam.2007.12.010.

[GVhP+22] Giuseppe Garofalo, Tim Van hamme, Davy Preuveneers, Wouter Joosen, Aysajan Abidin, and Mustafa A. Mustafa. Pivot: Private and effective contact tracing, 2022. doi:10.1109/JIOT.2021.3138694.

[HM20] Thomas Haines and Johannes Müller. SoK: Techniques for verifiable mix nets. In Limin Jia and Ralf Küsters, editors, *CSF 2020 Computer Security Foundations Symposium*, pages 49–64. IEEE Computer Society Press, 2020. doi:10.1109/CSF49147.2020.00012.

[HMM+21] Katie Hogan, Briana Macedo, Venkata Macha, Arko Barman, and Xiaoqian Jiang. Contact tracing apps: Lessons learned on privacy, autonomy, and the need for detailed and thoughtful implementation. *JMIR Med Inform*, 9(7):e27449, July 2021. doi:10.2196/27449.

[KL14] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography*. CRC Press, second edition, 2014.

[KLN23] Markulf Kohlweiss, Anna Lysyanskaya, and An Nguyen. Privacy-preserving blueprints. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part II*, volume 14005 of *LNCS*, pages 594–625. Springer, Heidelberg, April 2023. doi:10.1007/978-3-031-30617-4_20.

[MRV99] Silvio Micali, Michael O. Rabin, and Salil P. Vadhan. Verifiable random functions. In *40th FOCS*, pages 120–130. IEEE Computer Society Press, October 1999. doi:10.1109/SFFCS.1999.814584.

[MSBM23] Omid Mir, Daniel Slamanig, Balthazar Bauer, and Rene Mayrhofer. Practical delegatable anonymous credentials from equivalence class signatures. *Proceedings on Privacy Enhancing Technologies*, 2023:488–513, 06 2023. doi:10.56553/popets-2023-0093.

[Org24] World Health Organization. Who coronavirus (covid-19) dashboard, 2024. https://covid19.who.int/.

[PR21] Benny Pinkas and Eyal Ronen. Hashomer – privacy-preserving bluetooth based contact tracing scheme for hamagen. *Proceedings 2021 Innovative Secure IT Technologies against COVID-19 Workshop*, 2021. URL: https://api.semanticscholar.org/CorpusID:232073728, doi:10.14722/coronadef.2021.23011.

[RAC+20] Ronald L. Rivest, Hal Abelson, Jon Callas, Ran Canetti, Kevin Esvelt, Daniel Kahn Gillmor, Louise Ivers, Yael Tauman Kalai, Anna Lysyanskaya, Adam Norige, Bobby Pelletier, Ramesh Raskar, Adi Shamir, Emily Shen, Israel Soibelman, Michael Specter, Vanessa Teague, Ari Trachtenberg, Mayank Varia, Marc Viera, Daniel Weitzner, John Wilkinson, and Marc Zissman. The

pact protocol specification, 2020. https://pact.mit.edu/wp-content/uploads/2020/11/The-PACT-protocol-specification-2020.pdf.

[RBS20] Leonie Reichert, Samuel Brack, and Björn Scheuermann. Privacy-preserving contact tracing of covid-19 patients. Poster session, IEEE Symposium on Security and Privacy, 2020. https://www.ieee-security.org/TC/SP2020/poster-abstracts/hotcrp_sp20posters-final10.pdf.

[RBS21] Leonie Reichert, Samuel Brack, and Björn Scheuermann. Ovid: Message-based automatic contact tracing. CoronaDef Workshop, 2021. https://www.ndss-symposium.org/wp-content/uploads/coronadef2021_23010_paper.pdf.

[TN21] Cong Duc Tran and Tin Trung Nguyen. Health vs. privacy? the risk-risk tradeoff in using COVID-19 contact-tracing apps. *Technol Soc*, 67:101755, September 2021. doi:10.1016/j.techsoc.2021.101755.

[TPH+20] Carmela Troncoso, Mathias Payer, Jean-Pierre Hubaux, Marcel Salathé, James R. Larus, Edouard Bugnion, Wouter Lueks, Theresa Stadler, Apostolos Pyrgelis, Daniele Antonioli, Ludovic Barman, Sylvain Chatel, Kenneth G. Paterson, Srdjan Capkun, David A. Basin, Jan Beutel, Dennis Jackson, Marc Roeschlin, Patrick Leu, Bart Preneel, Nigel P. Smart, Aysajan Abidin, Seda F. Gürses, Michael Veale, Cas Cremers, Michael Backes, Nils Ole Tippenhauer, Reuben Binns, Ciro Cattuto, Alain Barrat, Dario Fiore, Manuel Barbosa, Rui Oliveira, and José Pereira. Decentralized privacy-preserving proximity tracing. *CoRR*, abs/2005.12273, 2020. URL: https://arxiv.org/abs/2005.12273, arXiv:2005.12273.

[Tra20] Tracetogether. Government of Singapore, 2020. https://www.tracetogether.gov.sg/. URL: https://www.tracetogether.gov.sg/.

[TSS+20] Ni Trieu, Kareem Shehata, Prateek Saxena, Reza Shokri, and Dawn Song. Epione: Lightweight contact tracing with strong privacy, 2020. URL: https://arxiv.org/abs/2004.13293, doi:10.48550/ARXIV.2004.13293.

[Vau20a] Serge Vaudenay. Analysis of dp3t. IACR ePrint, 2020. https://eprint.iacr.org/2020/399.

[Vau20b] Serge Vaudenay. Centralized or decentralized? the contact tracing dilemma. IACR ePrint, 2020. https://eprint.iacr.org/2020/531.

[WL20] Zhiguo Wan and Xiaotong Liu. Contactchaser: A simple yet effective contact tracing scheme with strong privacy. IACR ePrint, 2020. https://eprint.iacr.org/2020/630.

A Additional Preliminaries

A.1 DL Representations Proof in Prime Order Groups

Using known techniques [CS97, KLN23], a Σ-protocol can be constructed that proves the following relation in Def. 14 in prime order cyclic groups where the DDH and CDH problems are hard. We describe a σ-protocol that satisfies Def. 15.

Definition 14 (Relation for proof of equality of discrete logarithm representations in cyclic groups of prime order)**.** Let $R_{\mathsf{eqrep}-p}$ be the following relation: $R_{\mathsf{eqrep}-p}(\mathsf{w},\mathsf{x})$ accepts if $\mathsf{x} = (\mathbb{G}, \{x_i, \{g_{i,1}, \ldots, g_{i,m}\}\}_{i=1}^{n})$ where $\mathbb{G}$ is the description of a group of order p, and all the x_is and $g_{i,j}$s are elements of $\mathbb{G}$, and witness $\mathsf{w} = \{w_j\}_{j=1}^{m}$ such that $x_i = \prod_{j=1}^{m} g_{i,j}^{w_j}$.

Definition 15 (σ-protocol for equality of DL representations)**.** **P→V** On input the $(\mathsf{w}, \mathsf{x}) \in R_{\mathsf{eqrep}-p}$, the Prover chooses $e_j \leftarrow \mathbb{Z}_p$ for $1 \le j \le m$ and computes $d_i = \prod_{j=1}^{m} g_{i,j}^{e_j}$ for $1 \le i \le k$. Finally, the Prover sends to the Verifier the values $\mathsf{com} = (d_1, \ldots, d_n)$.

P←V On input x and com, the Verifier responds with a challenge $\mathsf{chal} = c$ for $c \leftarrow \mathbb{Z}_p$.

P→V The Prover receives $\mathsf{chal} = c$ and computes $s_i = e_i + cw_i \bmod p$ for $1 \le i \le m$, and sends $\mathsf{res} = (s_1, \ldots, s_m)$ to the Verifier.

Verification The Verifier accepts if for all $1 \le i \le n$, $d_i x_i^c = \prod_{j=1}^{m} g_{i,j}^{s_j}$; rejects otherwise.

Simulation On input x and $\mathsf{chal} = c$, the simulator chooses $s_j \leftarrow \mathbb{Z}_p$ for $1 \le j \le m$, and sets $d_i = (\prod_{j=1}^{m} g_{i,j}^{s_j})/x_i^c$ for $1 \le i \le k$. He then sets $\mathsf{com} = (d_1, \ldots, d_n)$ and $\mathsf{res} = (s_1, \ldots, s_m)$.

Extraction On input two accepting transcripts for the same $\mathsf{com} = (d_1, \ldots, d_n)$, namely $\mathsf{chal} = c$, $\mathsf{res} = (s_1, \ldots, s_m)$, and $\mathsf{chal}' = c'$, $\mathsf{res}' = (s'_1, \ldots, s'_m)$, output $w_j = (s_j - s'_j)/(c - c') \bmod p$ for $1 \le j \le m$.

We can see from the construction in Def. 15, that each NIZK will contain a group element for each element in the statement, as well as an element of $\mathbb{Z}_p$ for each witness.

A.2 Mercurial signatures

We formally define the inputs and outputs for a mercurial signature scheme [CL19] in Def. 16. We then formally define unforgeability in Def. 17 and define privacy definitions in Defs. 18, 19, 20, and 21. Mercurial signature schemes are defined over a set of equivalence relations which we described in Section 2.5. The randomizers for keys as well as the randomizers for messages (ρ and μ respectively in Def. 16) are drawn from $\mathbb{Z}_p$ in the construction in [CL19].

Definition 16 (A mercurial signature scheme)**.**

- $\mathsf{Setup}(1^\lambda) \to pp_{\mathsf{MS}}$: Outputs public parameters for the scheme
- $\mathsf{KeyGen}(pp_{\mathsf{MS}}, \ell) \to (pk, sk)$: Outputs a pair of keys for signing and verification.
- $\mathsf{Sign}(sk, M) \to \sigma$: Signs the message M.
- $\mathsf{Verify}(pk, M, \sigma) \to 0/1$: Verifies the signature on the message, M.
- $\mathsf{ConvertSK}(sk, \rho) \to sk'$: Transforms the secret key to work with a similarly randomized public key, i.e, if $\mathsf{Sign}(\mathsf{ConvertSK}(sk, \rho), M) = \sigma$, then $\mathsf{Verify}(\mathsf{ConvertPK}(pk, \rho), M, \sigma) = 1$.
- $\mathsf{ConvertPK}(pk, \rho) \to pk'$: Transforms the public key to work with a similarly randomized secret key, or a similarly randomized signature.
- $\mathsf{ConvertSig}(pk, M, \sigma, \rho) \to \sigma'$: Transforms a signature to work with a randomized public key, i.e, if $\mathsf{Verify}(pk, M, \sigma) = 1$, then $\mathsf{Verify}(\mathsf{ConvertPK}(pk, \rho), M, \mathsf{ConvertSig}(pk, M, \sigma, \rho)) = 1$
- $\mathsf{ChangeRep}(pk, M, \sigma, \mu) \to (M', \sigma')$: Transforms a message and signature into an equivalent message and a randomized signature, i.e., if $\mathsf{Verify}(pk, M, \sigma) = 1$ and $(M', \sigma') \leftarrow \mathsf{ChangeRep}(pk, M, \sigma, \mu)$, then $\mathsf{Verify}(pk, M', \sigma') = 1 \wedge [M']_{\mathcal{R}_M} = [M]_{\mathcal{R}_M}$.

Definition 17 (Mercurial Signatures Unforgeability)**.** A mercurial signature scheme (Setup, KeyGen, Sign, Verify, ConvertSK, ConvertPK, ConvertSig, ChangeRep) for parameterized equivalence relations $\mathcal{R}_{\mathcal{M}}$, $\mathcal{R}_{pk}$, and $\mathcal{R}_{sk}$ is unforgeable if for all polynomial-length parameters (λ) and all probabilistic, polynomial-time (PPT) algorithms $\mathcal{A}$ having access to a signing oracle Sign, then if $pp \leftarrow \mathsf{Setup}(1^\lambda)$, $(pk, sk) \leftarrow \mathsf{KeyGen}(pp, \ell(\lambda))$, and $(pk^*, M^*, \sigma^*) \leftarrow \mathcal{A}^{\mathsf{Sign}(sk,\cdot)}(pp, pk)$, the following probability holds:

$$\Pr\left[\begin{array}{l}(\forall M \in Q, [M^*]_{\mathcal{R}_M} \neq [M]_{\mathcal{R}_M}) \wedge [pk^*]_{\mathcal{R}_{pk}} = [pk]_{\mathcal{R}_{pk}} \\ \wedge\, \mathsf{Verify}(pk^*, M^*, \sigma^*) = 1\end{array}\right] \leq negl(\lambda)$$

where Q is the set of all queries to the signing oracle that $\mathcal{A}$ made

Definition 18 (**Origin-Hiding for** ConvertSig [CL19])**.** A mercurial signature scheme is origin-hiding for ConvertSig if, given any tuple (pk, σ, M) that verifies, and given a random key randomizer ρ, $\mathsf{ConvertSig}(\sigma, pk, \rho)$ outputs a new signature σ' such that σ' is a uniform random signature in the set of all verifying signatures for M, i.e., $\sigma' \in \{\sigma^* | \mathsf{Verify}(\mathsf{ConvertPK}(pk, \rho), M, \sigma^*) = 1\}$.

Definition 19 (**Origin-Hiding for** ChangeRep [CL19])**.** A mercurial signature scheme is origin-hiding for ChangeRep if, given any tuple (pk, σ, M) that verifies, and given a random message randomizer μ, $\mathsf{ChangeRep}(pk, M, \sigma, \mu)$ outputs a new message and signature M', σ' such that (M', σ') is a uniform random message/signature pair in $(M', \sigma') \in \{(M^*, \sigma^*) | \mathsf{Verify}(pk, M^*, \sigma^*) = 1 \wedge [M^*]_{\mathcal{R}_M} = [M]_{\mathcal{R}_M}\}$.

Definition 20 (**Message Class-Hiding** [CL19])**.** A mercurial signature scheme has message class-hiding if the advantage of any PPT adversary $\mathcal{A}$ defined by $\mathbf{Adv}^{\mathsf{MCH}}_{\mathcal{A}}(\lambda)$ $:= 2 \cdot \Pr\left[\mathbf{Exp}^{\mathsf{MCH}}_{\mathcal{A}}(\lambda) \Rightarrow \mathsf{true}\right] - 1 = \epsilon(\lambda)$, where $\mathbf{Exp}^{\mathsf{MCH}}_{\mathcal{A}}(\lambda)$ is shown in Fig. 9a.

Definition 21 (**Public Key Class-Hiding** [CL19])**.** A mercurial signature scheme has public key class-hiding if the advantage of any PPT adversary $\mathcal{A}$ defined by $\mathbf{Adv}^{\mathsf{PKCH}}_{\mathcal{A}}(\lambda)$ $:= 2 \cdot \Pr\left[\mathbf{Exp}^{\mathsf{PKCH}}_{\mathcal{A}}(\lambda) \Rightarrow \mathsf{true}\right] - 1 = \epsilon(\lambda)$, where $\mathbf{Exp}^{\mathsf{PKCH}}_{\mathcal{A}}(\lambda)$ is shown in Fig. 9b.

$\mathbf{Exp}^{\mathsf{MCH}}_{\mathcal{A}}(\lambda)$

$pp \leftarrow_\$ \mathsf{Setup}(1^\lambda); b \leftarrow_\$ \{0,1\}; \rho \leftarrow_\$ \mathbb{Z}_p$

$M_1 \leftarrow_\$ \mathcal{M}; M_2^0 \leftarrow_\$ \mathcal{M}$

$M_2^1 \leftarrow_\$ [M_1]_{\mathcal{R}_M};$

$b' \leftarrow_\$ \mathcal{A}(M_1, M_2^b); \mathbf{return}\ b = b'$

(a) Message class-hiding experiment from [CL19, CGH+23].

$\mathbf{Exp}^{\mathsf{PKCH}}_{\mathcal{A}}(\lambda)$

$pp \leftarrow_\$ \mathsf{Setup}(1^\lambda); b \leftarrow_\$ \{0,1\}; \rho \leftarrow_\$ \mathbb{Z}_p$

$(sk_1, pk_1) \leftarrow_\$ \mathsf{KeyGen}(pp);$

$(sk_2^0, pk_2^0) \leftarrow_\$ \mathsf{KeyGen}(pp)$

$pk_2^1 \leftarrow \mathsf{ConvertPK}(pk_1, \rho); sk_2^1 \leftarrow \mathsf{ConvertSK}(sk_1, \rho)$

$b' \leftarrow_\$ \mathcal{A}^{\mathsf{Sign}(sk_1,\cdot),\mathsf{Sign}(sk_2^b,\cdot)}(pk_1, pk_2^b);$

$\mathbf{return}\ b = b'$

(b) Public key class-hiding experiment from [CL19, CGH+23].

Figure 9: Privacy games for mercurial signatures

A.2.1 Proof of VRF over a mercurial key

For mercurial keys in the construction from [CL19] where the length of the vector is $\ell = 2$, a single discrete log describes all representations of an equivalence class. To understand why this is, we quickly have to review the construction in [CL19]. In this construction, mercurial secret keys are $sk = \{x_1, x_2\} \in (\mathbb{Z}_p)^2$ and the corresponding public keys are $pk = \{\hat{X}_1, \hat{X}_2\} = \{\hat{P}^{x_1}, \hat{P}^{x_2}\} \in \mathbb{G}_2$ where $\mathbb{G}_2$ is the second source group of a bilinear

Definition 22 (Correctness)**.**
A FHS set-commitment scheme is correct if for all S,T where $T \subset S$, if $(C, O) \leftarrow \mathsf{Commit}(pp, S)$ and $(W) \leftarrow \mathsf{OpenSubset}(pp, C, S, O, T)$, then $\mathsf{Open}(pp, C, S, O) = 1$ and $\mathsf{VerifySubset}(pp, C, T, W) = 1$.

Definition 23 (Hiding)**.**
A FHS set-commitment scheme is hiding if for all security parameters, $\lambda \in \mathbb{Z}_{>0}$, and any polynomial adversary, $\mathcal{A}$, if $(S_0, S_1, st) \leftarrow \mathsf{Adv}(pp)$, $b \leftarrow \{0, 1\}$, $(C, O) \leftarrow \mathsf{Commit}(pp, S_b)$, and $(b') \leftarrow \mathsf{Adv}^{\mathsf{OpenSubset}(pp, \mathsf{C}, \mathsf{S_b}, \cdot \cap \mathsf{S_0} \cap \mathsf{S_1})}(pp, C, st)$ the probability that $b = b'$ is $\leq \frac{1}{2} + negl(\lambda)$.

Definition 24 (Binding)**.**
A FHS set-commitment scheme is binding if for all security parameters, $\lambda \in \mathbb{Z}_{>0}$, and any polynomial adversary, $\mathcal{A}$, if $(C, S, O, S', O') \leftarrow \mathsf{Adv}(pp)$, $\mathsf{Open}(pp, C, S, O) = 1$, $\mathsf{Open}(pp, C, S', O') = 1$, then the probability that $S \neq S'$ is $\leq negl(\lambda)$.

Definition 25 (Subset soundness)**.**
A FHS set-commitment scheme has subset soundness if for all security parameters, $\lambda \in \mathbb{Z}_{>0}$ and any polynomial adversary, $\mathcal{A}$, if $(C, S, O, T, W) \leftarrow \mathsf{Adv}(pp)$, $\mathsf{Open}(pp, C, S, O) = 1$, and $\mathsf{VerifySubset}(pp, C, T, W) = 1$, then the probability that $T \not\subseteq S$ is $\leq negl(\lambda)$.

Figure 10: Properties of FHS commitments [FHS19]

- $\mathsf{Setup}(1^\lambda, 1^s) \to pp$: Sample a bilinear pairing group, $(e, \mathbb{G}_1, \mathbb{G}_2, P, \hat{P})$ of prime order, p. Pick a trapdoor, $\alpha \leftarrow_\$ \mathbb{Z}_p$ and output: $pp = (P, \hat{P}, \{P^{\alpha^i}, \hat{P}^{\alpha^i}\}_{i \in [s]})$. The message space for this commitments scheme is sets of size s with elements in $\mathbb{Z}_p$.
- $\mathsf{Commit}(pp, S; O) \to (C, O)$: If O is omitted, sample an opening, $O \leftarrow_\$ \mathbb{Z}_p$. Compute $C = (\prod_{i \in [s]} P^{\alpha^i} / P^{m_i})^O$ where $S = \{m_i\}_{i \in [s]}$. Output (C, O).
- $\mathsf{Open}(pp, C, S, O) \to \{0, 1\}$: Recompute: $C' = \mathsf{Commit}(pp, C, S; O)$ and check that $C' = C$.
- $\mathsf{OpenSubset}(pp, C, T, O) \to W$: Compute $W = (\prod_{m \in S \setminus T} P^{\alpha^i} / P^m)^O$.
- $\mathsf{VerifySubset}(pp, C, T, W) \to \{0, 1\}$: Compute $C' = \prod_{m \in T} \hat{P}^{\alpha^i} / \hat{P}^m$. Ensure that $e(W, C') = e(C, \hat{P})$.

Figure 11: FHS set commitment scheme [FHS19]

pairing. Thus, the discrete log between the two elements of the public key is $\log_{\hat{X}_1}(\hat{X}_2)$. A public key from [CL19] is randomized by computing $pk' = \{\hat{X}_1^\rho, \hat{X}_2^\rho\}$ for $\rho \in \mathbb{Z}_p$. We can see that no matter what the choice of ρ is, this discrete log remains constant. Using this discrete log we can ensure that the same VRF key is used across all representations of a mercurial public key. This can be done using the eqrep protocol from Appendix A.1. A proof similar to [CHK+06] can be used to prove the correct output of a VRF. Like in [CHK+06], the Dodis-Yampolskiy VRF [DY05] is efficient for bilinear pairings and can be used for our constructions in Section 4.1. Similarly, we can use the protocol in Section A.1 to open CoECs and compute VRFs on their attributes in zero knowledge.

A.3 FHS signatures

The FHS set-commitment scheme has the properties, hiding, binding, and subset soundness described in Fig. 10 along with correctness.

We review the construction of the FHS set commitment scheme from [FHS19] in Figure 11. We use the insight from [MSBM23] to simplify the construction, which allows us to ignore the case when the trapdoor is committed to. In practice, no user will have knowledge of this trapdoor and since it is drawn uniformly at random from an exponentially sized set, no user can compute any sets that contain this.

- $\mathsf{ProveSubset}_{\mathsf{FHS}}(pp, C, C', S, S', O, O') \rightarrow \pi$: This proof computes $\pi_W = (\prod_{m \in S' \setminus S} \hat{P}^{\alpha^i} / \hat{P}^m)^{\mu O'/O}$ such that $e(C, \pi_W) = e(C', \hat{P})^{\mu}$. It then produces a NIZK such that a μ is known that satisfies this pairing equation and that the opening of C and C' are known.
- $\mathsf{VerifySubset}_{\mathsf{FHS}}(pp, C, C', \pi) \rightarrow \{0, 1\}$: Verify that $A = e(C, \pi_W)$ and $B = e(C', \hat{P})$ and verify the NIZK proving that $\mathsf{dlog}_A(B)$ is known as well of the construction of π_W.

Figure 12: Additional functions for FHS set commitments

To create the FHS subset proofs in our proof system in Section 4.3.2, we need an additional function, $\mathsf{ProveSubset}_{\mathsf{FHS}}$ constructed in Figure 12. This proof verifies that the set S committed to by a commitment C is a subset of the set S' committed to by another commitment, C'. In Camenish-Stadler notation, this proves the relation: $NIZK[S, S', O, O' : \mathsf{Open}(C, S, O) = 1 \wedge \mathsf{Open}(C', S', O') = 1 \wedge S \subset S']$. In this case, we can see that if the π output by $\mathsf{ProveSubset}_{\mathsf{FHS}}$ verifies and C and C' are opened, then we can extract a W which proves that C is a subset of C'.

Proof of simulation-extractability and zero-knowledge of the proof system in Fig. 12 To prove simulation-extractability, we can see that we can extract μ, O from the NIZK and compute $W = \pi_W^{O/\mu}$. Because we can also extract $\{m \in S' \setminus S\}$ such that $\pi_W = (\prod_{m \in S' \setminus S} \hat{P}^{\alpha^i} / \hat{P}^m)^{\mu O'/O}$, we know that $W = (\prod_{m \in S' \setminus S} \hat{P}^{\alpha^i} / \hat{P}^m)^{O'}$. We can see that this satisfies $e(\prod_{m \in S} P^{\alpha^i - m}, W) = e(C', \hat{P})$. This means that if an adversary can open C' to some set, S', such that $S \not\subset S'$, we can use this adversary to violate subset soundness of the FHS commitment scheme.

To prove zero-knowledge, we can create a simulator that randomly samples π_W and uses the eqrep simulator to produce proofs for μ and $m \in S' \setminus S$.

A.3.1 Concatenating attributes to fit into the FHS message space

While constructing our proof in Section 4.3.2, we need to be able to construct FHS commitments to sets of vectors of the attributes committed to by our CoEC commitments. To do this, we use a concatenation function similar to [CHK+06] which divides the message space of the FHS commitment scheme into bits and assigns sets of those bits to the different attribute spaces of the CoEC commitment. For example, in our construction we only need to concatenate the time and location attributes. If our FHS commitment scheme has 128 bits of message space (i.e. the order p that defines $\mathbb{Z}_p$ is at least 128 bits) then we can define our time attribute space as $[0, ..., 2^{64} - 1]$ and our location attribute space as $[2^{64}, 2 * 2^{64}, 3 * 2^{64}, ..., 2^{128} - 1]$. Our concatenation function is then simply adding the two attributes, which can be efficiently proven in zero knowledge. This ensures that the concatenation function is bijective, meaning that vectors of attributes will collide when concatenated into the FHS message space. We also need range proofs to ensure that our commitments are in the correct ranges. This can be done using bulletproofs [BBB+18, CBC+24].

B Concrete efficiency

To sumarize, our uploads size is $15|\mathbb{G}| + 48n|\mathbb{G}| + 4|\mathbb{Z}_p| + 30n|\mathbb{Z}_p| + 4n|\text{range proof}|$ where n is the number of interactions in the upload. If we consider 256 bits for group elements [BD19], 128 bits for elements of $\mathbb{Z}_p$ and 700 bytes for range proofs as described in [CBC+24, BBB+18], our uploads total: $592 + 4880n$ bytes. These values may change

depending on the implementation, but should be close enough for an estimate. With these parameters, our chirps total ten group elements and one element of $\mathbb{Z}_p$, totaling 336 bytes. After a batch has been verified, which only needs to be done a single time by any trusted party as the database is public, then checking the database only requires recognizing a public key in the database, which only requires one multiplication for each entry in the database. Verifying chirps requires two mercurial signature verifications, of which the bulk of the computation is the pairing operation of which there are three instances per signature (one for each element in key/message vectors and one to verify Y and $\hat{Y}$). Thus, this requires six pairing operations in total. Verifying chirps can be delayed until an upload is required. Computing a chirp requires only elliptic curve multiplication and exponentiation which is much more efficient than pairing operations. We describe the computation of these totals below:

Chirps (described in Figure 5) consist of a public key, $pk_{\mathcal{U}}$, which takes up two group elements. The chirp also contains two mercurial signatures, $cert'$ and σ_{c}, requiring three group elements each, totaling six group elements. The chirp also contains a CoEC commitment, C, requiring two more group elements, and a random roken, r, requring one element of $\mathbb{Z}_p$. The opening can be defined canonically (e.g. as $O = 1$) to omit it from the chirp. Thus, chirps contain 10 group elements and one element of $\mathbb{Z}_p$. While storing chirps, the listener will store two more elements of $\mathbb{Z}_p$ (to remember the time and location). The epoch can be rederived from the stored time. For a batch in an upload, described in Figure 6, the uploader will create $pk'_{\hat{\mathcal{U}}}$ which consists of two group elements. They will also compute $cert'$ which is three group elements, Y which is a single group element, and S^* which will cost $7n$ group elements. These $7n$ group elements are: two group elements for each C_i^*, two group elements for each ${pk_{\mathcal{U}}}_i^*$, and three group elements for each $\sigma_{\mathsf{c},i}^*$. The π_Y value includes a proof over which pertains to three group elements, Y and $pk_{\hat{\mathcal{U}}}$ ($pk'_{\hat{\mathcal{U}}}$ takes up two group elements). There are two secrets in this proof, (the elements of the secret key, $sk'_{\hat{\mathcal{U}}}$). This proof is also computed on the current epoch, d, but this is public so it is not included in the proof and is not a witness. Thus, a discrete log representation proof as described in [CS97] (as well as in Appendix A.1) will require three group elements and two elements of $\mathbb{Z}_p$. The batch also contains the proof, π_{clone} (described in Figure 7), which contains two Schnorr-style keys, K, and K', each consisting of one group element, as well as $2n$ commitments, $\{C'_i, B'_i\}_{i\in[n]}$, which can be represented with $4n$ group elements. The values $\{Y_{r,i}, Y_{t,i}\}_{i\in[n]}$ can be represented as $2n$ group elements. The proof, π_d, contains $n+1$ witnesses, as well as $2n$ elements (for each of the commitments). The proofs, $\pi_{r,i}$, each pertain to five group elements (which make up $Y_{r,i}$, $pk'_{\hat{\mathcal{U}}}$, and C_i) as well as four elements of $\mathbb{Z}_p$ for witnesses, which include the token r, the secret key and randomization of $pk'_{\hat{\mathcal{U}}}$, and the opening of C_i. Each $\pi_{t,i}$ has the same number of elements, five group elements and four elements of $\mathbb{Z}_p$ for the same reasons. All of these proofs can share the same challenge, which can be computed as a hash of the statement. The proof $\pi_{t,l}$ requires two group elements for each CoEC commitment, C_i and two group elements for each time/location commitment, C'_i. For $\pi_{t,l}$ we also need six elements of $\mathbb{Z}_p$ for the witnesses, one for each CoEC attribute one for each of the randomization of both C_i and C'_i. For the proof, π_{clone}, descibed in Figure 8, we require one group element for each of C, D, A'_i, B'_i, totaling $2 + 2n$ group elements. Further, we require $n + 1$ group elements and $2n + 1$ $\mathbb{Z}_p$ elements for the proof, $\pi_{\{B_i\}\approx D}$, totaling $3 + 3n$ group elements and $2n + 1$ $\mathbb{Z}_p$ elements. For π', we require $12n$ elements of $\mathbb{Z}_p$ for the witnesses, and $4n$ group elements for the commitments. We then need $4n$ range proofs, for each of the concatenations of attributes.

C Proofs

C.1 ProvenParrot proofs

C.1.1 Proof of Clone integrity (Theorem 1)

Let $\mathcal{E}$ be the following set of extractors. We define extra extractors that are not required for the definition, $\mathcal{E}_B, \mathcal{E}_{pk_{rp}}$, and $\mathcal{E}_c$, but we'll see that these are useful to simplify the proof.

Definition 26 (Extractors for security proofs)**.**

- $\mathcal{E}_{pk}(pp, pk)$: Use the trapdoor to compute sk such that $P^{sk} = pk$ (for $\ell = 2$, this is $pk = (P^{sk_0}, P^{sk_1})$). Normalize this so that $sk_0 = 1$ (multiply sk_1 by $(sk_0)^{-1}$ to derive sk_1). Output sk.
- $\mathcal{E}_c(pp, c)$: Use $\mathcal{E}_{pk}$ for the given $pk_{\mathcal{U}}$ in c.
- $\mathcal{E}_{pk_{rp}}(pp, c)$: Use $\mathcal{E}_{pk}$ for the given pk_{rp}.
- $\mathcal{E}_B(pp, B)$: Take $pk'_{\hat{\mathcal{U}}}$ from the batch and use $\mathcal{E}_{pk}$ to extract a secret key, $sk_{\hat{\mathcal{U}}}$. Also extract the attributes, t, l, from the NIZKs for each chirp. Extract sk from the public key in each chirp. Output a set of each interaction with $sk_{\hat{\mathcal{U}}}$ as the receiver.
- $\mathcal{E}_{DB}(pp, DB)$: Iterate through each $B \in DB$ and compute $\mathcal{E}_B(B)$ and union the result i.e. output: $\bigcup_{B \in DB} \mathcal{E}_B(B)$.

The adversary in Game 1 can win if after the game, any of the 3 conditions (Equations 1, 2, and 3) in the integrity definition are false. We will go through each equation and prove that the probability of any PPT adversary violating them is negligible. In each of these three proofs, we create a reduction $\mathcal{B}$ that plays either the commitment binding game, the NIZK simulation-extractability game, or the mercurial signature unforgeability game. $\mathcal{B}$ acts as the challenger, setting up the clone integrity game (Game 1) and providing honest or simulated responses to the oracle queries of $\mathcal{A}$ until they exit. The reduction, $\mathcal{B}$, will transform the adversary's inputs to the oracles and the adversary's state in a number of ways dependent on how the integrity game fails. We'll start by showing that Equation 3 holds as the proof of Equation 2 relies on this.

Proving Equation 3 - Clone protection To prove that this equation holds, we'll break the equation into two equations which each ensure a separate guarantee. The first equation (Eq. 5) ensures that clone protection holds within a batch.

$$\forall B \in DB, \nexists ((*, *, t, l), (*, *, t', l')) \in (\mathcal{E}_B(pp, B))^2 \text{ s.t. } t = t' \wedge l \neq l' \tag{5}$$

The second equation (Eq. 6) ensures that no two batches share a listener across their extracted exposure tuples.

$$\forall B \in DB, \nexists B' \in DB \setminus \{B\} \text{ s.t. } \exists(*, sk_{\mathcal{U}}, *, *) \in B \wedge \exists(*, sk_{\mathcal{U}}, *, *) \in B' \tag{6}$$

We can see that, because $\mathsf{EI} = \mathcal{E}_{DB}(DB)$, if the Equations 5 and 6 hold, then Equation 3 holds. This is because for pair of tuples to exist in EI that violate Equation 3, they need to have the same listener, thus, by Equation 6, they are in the same batch and by Equation 5, then they must have distinct time values or the same locations.

First we'll prove Equation 6. Informally, we use the fact that the uploader must compute $\mathsf{PRF}_{sk'_{\hat{\mathcal{U}}}}(d_{\mathsf{now}})$ (where $d_{\mathsf{now}} = \mathsf{Epoch}(t_{\mathsf{now}})$), and prove that this is related to $pk'_{\hat{\mathcal{U}}}$, resulting in Y, π_Y. Y is checked for uniqueness among batches in DB, during $\mathsf{VerifyBatch}$. Because DB is emptied when $\mathsf{Epoch}(d_{\mathsf{now}})$ changes, we know that each of these Y across batches are computed on the same d. Because $\mathsf{PRF}_{(\cdot)}(\cdot)$ normalizes the secret key before evaluation of the PRF, then, even if the user randomizes their keys, this function will still be evaluated on the same values, $\frac{sk_1}{sk_0}$ and d_{now}. Thus, if two batches are uploaded with equivalent public keys, then the Y^{pk} values in the batches will be the same. This will be detected and rejected during $\mathsf{VerifyBatch}$, thus ensuring Equation 6 can't be violated.

More formally, because we can extract a canonical secret key from Y, π_Y (in the random oracle model using the NIZK associated with Y) and because VRFs are deterministic, if

all the Y values are distinct, then we know that we can we must be able to extract distinct $sk_{\hat{\mathcal{U}}}, d$ from each Y,π_Y pair in the database. Because d is public and each is fixed for each proof in the database, π_Y, then each sk must be distinct. This formal proof is inspired by the soundness proof in [CHK+06].

Our extractor, $\mathcal{E}_B$, uses this canonical $sk_{\hat{\mathcal{U}}}$ extracted from π_Y to output the listener value for each extracted tuple from the batch (in Def. 26). Thus, if none of these $sk_{\hat{\mathcal{U}}}$ values are shared across any batch, then none of the listener keys in the tuples of this batch are shared with any other batch. Thus, Equation 6 cannot be violated.

We see that verifying π_{clone} directly proves that Eq. 5 holds. For a more detailed analysis, see the proof of simulation extractability for the no-cloning relation in Appendix C.3. Because the π_{clone} relation also opens the commitments, if the adversary computes a $Y_{t,i}$ for a $\mathbf{M}_i[\mathsf{idx_t}]$ which the honest chirper did not sign, a reduction can use this extracted opening from $\pi_{t,i}$ along with the honest adversary's opening to double open the commitment, thus breaking Definition 11 or a reduction can break the forgeability of mercurial signatures.

Proving Equation 1 - Correct exposure count This equation ensures that the count of exposures by each user matches the extracted interactions from the database, fulfilling the equality in Equation 1.

We first need to prove that there's no duplicated tuples extracted in the batch that indicate that a chirp from the same honest user at the same time and location was sent. Because CountExposures would double count this, while extracting a duplicate into a set would only count it once, we need to ensure this duplication doesn't happen (a set contains no duplicate tuples). Thus, we need to prove that a batch does not contain duplicates of any interaction where the sender is honest. Let's reduce an adversary that violates the integrity in this way to an adversary that can open a commitment to two openings or a mercurial signature forgery. An honest user will never chirp twice on one t and so they will never sign two commitments that share a t and have difference nonces (r). An uploader must compute a VRF on the nonce committed to by the chirp, $\mathsf{PRF}_{sk_{\hat{\mathcal{U}}}}(r)$. We can see that simply rerandomizing another chirp honestly will not yield a new PRF output as it is still committed to the same nonce and private key class. Thus, if the adversary is able to produce a distinct $Y_{r,i}$ in π_{clone} for this rerandomized proof, they must have been able to open the commitment up to another r_i', forged a signature on a new commitment, or violated the soundness of the NIZK.

More formally, let's say this adversary produced two commitments C, C' such that we extract the same tuple from them using the extractor $\mathcal{E}_B$. If $[C]_{\mathsf{R}} = [C]'_{\mathsf{R}}$, because the adversary creates a proof of knowledge of each attribute that the commitment is committed to as well as the opening information, a reduction can extract this information from both openings and output this as a double opening in the game for Definition 11 as $r_i \neq r_i'$. If these two commitments are not in the same equivalence class, and we have $t = t', r \neq r', pk_{\mathcal{U}} = pk_{\mathcal{U}}{}'$, then we have another two cases: (1) this is a forgery in the game in Definition 17, since honest users never sign two distinct nonces for a single t in the PACIFIC game; or (2) one of these is a double opening from another commitment the user made (to change the time value that the commitment is committed to). Even if the adversary did not include this second commitment in the upload, we can look back on chirps that honest users made to find the opening information and attributes.

Now that we've proven that duplicates in the extraction are prevented by our assumptions on underlying schemes, we can see that Equation 1 holds from inspection of the extractor, $\mathcal{E}_B$, the CountExposures function, and Recognize. CountExposures uses Recognize to count exactly the number of chirper public keys in the batch where the discrete log is equivalent to the normalized user's secret key, $\mathsf{HU}_{sk}(i)$. This is exactly what we're counting in the extracted tuples, the normalized secret key from the chirper.

Proving Equation 2 - Batch reflects possible interactions This equation ensures that the extracted interactions are a subset of a set of possible interactions. During extraction, if we have B such that $\mathcal{E}_B(pp, B) \not\subset \mathsf{PI}$, there must exist a tuple $(sk, sk', t, l) \in \mathcal{E}_B(pp, B)$ such that (sk, sk', t, l) is not in PI (remember the format of tuples: (chirper, listener, time, location)).

We will now prove that if Eq. 2 does not hold, a reduction can transformation the adversary's input to constitute a violation of either the commitment binding game or the mercurial signature unforgeability game.

We'll break down this violation of Eq. 2 into 4 different restrictions on the tuple (sk, sk', t, l) to make the proof easier to follow. These restrictions only concern qualities of the chirper and listener and together constitute all possible combinations, thus exhausting possible pairs of uploader and listener. The first three restrictions are straightforward, but the last restriction categorizes a false exposure and thus is the most important and difficult to prove. Let U be the set of all registered users: $U = \mathsf{CU}_{sk} \cup \mathsf{HU}_{sk}$ where CU_{sk} are the secret keys of all corrupted users (extracted with $\mathcal{E}_{pk}$) and HU_{sk} is the set of all honest user secret keys.

1. sk or $sk' \notin U$
2. $sk, sk' \in (\mathsf{HU}_{sk})^2$
3. $sk, sk' \in (\mathsf{CU}_{sk})^2$.
4. $sk \in \mathsf{HU}_{sk}$ and $sk' \in \mathsf{CU}_{sk}$.

Restriction 1 If $sk' \notin U$, then the certificate, $cert'$, for this batch holds a forgery, since the secret keys are extracted from the public key $pk'_{\hat{\mathcal{U}}}$ for this batch and $cert'$ is a signature on this key. The secret key for the uploader, sk', is extracted from $pk'_{\hat{\mathcal{U}}}$. Also, because the batch also includes randomized versions of each chirp's certificate, if $sk \notin U$, then there exists a certificate with a forgery in the batch.

Restriction 2 Let us look at the case where $sk, sk' \in (\mathsf{HU}_{sk})^2$. We know from equation 6 that no two batches share an uploader. Because honest users upload their batches before the adversary learns them, the adversary cannot upload a second batch by replaying that user's public key and signature. This is because the Y would be the same for the second upload, and so the database would reject it in the $\mathsf{VerifyBatch}$ function. Thus this interaction must have come from an honest user's batch and is in PI due to the correctness of the scheme.

Restriction 3 This instance is the adversary notifying themselves, which intuitively is not something we care about for integriy. In this case, we know this is in PI because we include in PI all possible interactions between malicious users.

Restriction 4 Our last, but most important case: $sk \in \mathsf{HU}_{sk}$ and $sk' \in \mathsf{CU}_{sk}$, indicates a fake exposure. Because, for every chirp that an honest user made, we add a tuple to PI for each $sk' \in \mathsf{CU}_{sk}$, this means that no honest user signed a commitment with t, l. Thus, the adversary must have either double opened a commitment to a distinct t', l' or forged a signature. Our reduction, is assured that the adversary knows the opening of this commitment because the uploader proves knowledge of these attribute witnesses in the batch. Also, during chirping, honest users provide openings for the commitments in their chirps. Similarly to our proof Equation 1, we'll break this into the case where this violation shares an equivalence class with another commitment and the case where it doesn't. If this violating commitment shares an equivalence class with a commitment in another chirp, the adversary violated the commitment scheme to open the commitment in a second way. Thus a reduction, can extract the opening information from the adversary and sumbit this as a double opening along with the honest user's original opening information in their

chirp. If this commitment doesn't share an equivalence class with any other commitment signed by the user, then it is a forgery in the mercurial signature scheme under the user's public key. This proves that Equation 2 cannot be violated given the security definitions of our commitment scheme along with the unforgeability of mercurial signatures.

Note that to find these forgeries in the game, we can choose a commitment in the game randomly to be signed by the mercurial signature unforgeability challenger. Because the adversary can only call a polynomial number of functions, and because our construction samples honest users' keys in the same way that the mercurial signature unforgeability challenger does, we have a non-negligible chance of having this forgery reduce to the mercurial signature unforgeability game. A reduction can similarly use a probablistic method for double opening if equivalence classes are undecidable.

We have now proven that Equations 1, 2, and 3 cannot be violated by a PPT adversary given our construction and extractors, thus proving Thm. 1.

C.1.2 Proof of chirp privacy

Proof intuition We're going to reduce this to the mercurial public key class-hiding game from Fig. 9b. In the ideal function, $\mathsf{RecvChirp}^{sim}$, we've used a new random user for each simulated chirp. The mercurial public key class-hiding game This means we need to construct a hybrid argument where we iteratively replace calls of $\mathsf{RecvChirp}$ with simulated values using the pk_2^b given to the reduction from the mercurial signature public key class-hiding game in Fig. 9b. We do not need to hide the attributes since this is given as input to ideal function.

Proof of Theorem 2 Assume an adversary can win in Game 2 with non negligible probability. Using this adversary, we'll create a reduction $\mathcal{B}^{\mathsf{MS}}$ that wins the mercurial public key class-hiding game.

Let q be the maximum number of times that the adversary queries any oracle. Thus, this q bounds the number of times the adversary queries $\mathsf{RecvChirp}$ for any user and the number of honest users the adversary creates. Note that if the adversary is PPT, then q is less than $p(\lambda)$ for some polynomial, p. Let $Hybrid_{i,j}$ be similar to Game 2 but for the first j chirps computed by any of the first i honest users, the keys and certificate $(sk_{\mathcal{U}}, pk_{\mathcal{U}}, cert)$ for the chirp are replaced by new, randomly generated keys $(sk_{\mathcal{U}}, pk_{\mathcal{U}})$ and a freshly issued certificate on those keys. These hybrids use the original location given to $\mathsf{RecvChirp}$, l, and the time stored in the global state, t_{now}. Note that $Hybrid_{i,q+1}$ and $Hybrid_{i+1,0}$ are identical.

Let $\mathcal{B}^{\mathsf{MS}}$ be a reduction that chooses a random $Hybrid_{i,j}$ to emulate, but acts differently for the j-th chirp from user i. During the registration for user i, the reduction uses sk_1, pk_1 given by the class-hiding game. Then, for the j-th chirp for that user, the reduction uses pk_2^b for that chirp. We can see that if $q \geq j \geq 0$, this reduction looks like $Hybrid_{i,j+b}$. Thus, which hybrid this reduction emulates depends on the bit in the class-hiding scheme and thus distinguishing the hybrids is equivalent to distinguishing the public keys in the class-hiding game.

Because $Hybrid_{0,0}$ is our real game and $Hybrid_{q,q}$ is our simulated game, and we've shown that distinguishing each step is negligible, we can see that these two games are indistinguishable.

In order to complete this reduction, we need to show that sk_{rp} is accessible by a PPT reduction. This is why we have the adversary include a NIZK that they know the secret key of the registration party. In the random oracle model, this NIZK proves that sk_{rp} is somewhere in the adversary's state. If this extraction fails, the proof of the registration party's key must be incorrect and thus the adversary cannot create any honest users in the game as they will all output $\perp$, aborting the honest user creation.

C.1.3 Proof of upload privacy

Proof intuition Similarly to the proof of chirp privacy, we will reduce this to the hiding definitions of mercurial signatures. In contrast to the proof of chirp privacy, we also need to simulate the time and locations of interactions. We will see that distinguishing the simulator (which uses random times and locations instead of the real ones) will reduce to the class-hiding or origin-hiding properties of our CoEC commitment scheme defined in Section 3.2.

Proof of Theorem 3 Assume an adversary can win in Game 3 with non negligible probability. We'll create a reduction $\mathcal{B}^{\mathsf{MS}}$ that plays the mercurial public key class-hiding game and a reduction $\mathcal{B}^{\mathsf{Com}}$ that plays the commitment hiding game for CoECs (described in Section 2.2), both using this assumed adversary that can win Game 3.

Let q be defined as in the proof for Theorem 2. Let $Hybrid^{\mathsf{c}}_{i,j}$ use fresh, random users and certificates for the first j chirps from the first i users just like the hybrids in the proof of Theorem 2. We can use the reductions from the proof of Theorem 2 to ensure that $Hybrid^{\mathsf{c}}_{0,0}$ is indistinguishable from $Hybrid^{\mathsf{c}}_{q,q}$. Let $Hybrid^{B}_{i,j}$ act like $Hybrid^{\mathsf{c}}_{q,q}$ but use the simulator for the first j batches uploaded by user i [1]. Observe that, similarly to the proof of Theorem 2, $Hybrid^{\mathsf{c}}_{q+1,q+1} = Hybrid^{B}_{0,0}$ and that $Hybrid^{B}_{q+1,q+1}$ is the ideal game.

Claim 1. *$Hybrid^{B}_{i,j}$ is indistinguishable from $Hybrid^{B}_{i,j+1}$*

Proof of Claim 1 To prove Claim 1, we construct additional hybrids and reductions. We construct one hybrid for each chirp given to Notify and the simulator. We first create hybrids that replace the honest chirps given to Notify, which we'll label $Hybrid^{B,\mathsf{HU}}_{i,j,k}$. This hybrid acts like $Hybrid^{B}_{i,j}$ but, for the first k hoenst chirps passed to Notify (for the j call to Notify from user i), we generate each of these chirps using a freshly registered user: $sk_{\mathcal{U}}, pk_{\mathcal{U}} \leftarrow \mathsf{UserKeyGen}(1^k)$, $cert \leftarrow \mathsf{RegisterUser}(pp, sk_{rp}, pk_{\mathcal{U}})$ and random attributes (t, l). Next, we'll create hybrids for chirps from corrupted users, $Hybrid^{B,\mathsf{CU}}_{i,j,k}$. These hybrids simulate all chirps for honest users like $Hybrid^{B,\mathsf{HU}}_{i,j,q+1}$ but instead replace the first k corrupted chirps. The corrupted chirps are simulated in the same ways as the honest chirps, but the secret keys are not regenerated. Instead the simulated chirps are created using the secret keys of the adversaries that sent chirps to this user during this time period.

Claim 2. *$Hybrid^{B,\mathsf{CU}}_{i,j,k}$ is indistinguishable from $Hybrid^{B,\mathsf{CU}}_{i,j,k+1}$*

Proof of Claim 2 In this proof, we are proving that (in the j-th batch from the i-th honest user) replacing the k-th chirp from a corrupted user is indistinguishable from when a fresh key and signature is generated and random attributes are used. If we replace the public key and its certificate of this chirp by randomizing them, the origin-hiding of ChangeRep proves that this new public key and signature looks entirely independent from any freshly generated public key and signature in the same equivalence class. The simulator generates a fresh key and certificate in the same equivalence class. Thus, replacing the public key and certificate with freshly generated ones from the batch is indistinguishable. Next, we can create a reduction that replaces the commitment and reduce this to our hiding game for CoEC in Section 2.2 by creating a reduction $\mathcal{B}^{\mathsf{Com}}$ that computed C, C' to output in the game as commitments to M as the original attributes (t, l, d, r) for this chirp and M' as random attributes respectively. Depending on what CoEC returned to our reduction, this reduction would either be replacing the commitment with random

[1] A reader that just read the integrity definition might be confused as to why there are multiple batches for a single user. This is because the adversary's decision in the privacy game can be informed by older batches from previous time periods. These batches are not considered in the integrity game.

attributes or not. We see that we can use the simulator for the no-cloning scheme as well as the VRF scheme to ensure these proofs are indistinguishable as well.

Claim 3. *$Hybrid^{B,\mathsf{HU}}_{i,j,k}$ is indistinguishable from $Hybrid^{B,\mathsf{HU}}_{i,j,k+1}$*

Proof of Claim 3 We'll first consider a reduction to the public key class-hiding game. For the k-th chirp from an honest user in this batch (the j-th batch from the i-th honest user), the reduction replaces this honest user's sk, pk with a new, randomly generated one. We'll call this user the "hybrid user" in this proof. Our reduction then simulates any chirps from the hybrid user passed to the Notify function up to $k-1$ using freshly sampled keys, commitments and signatures. For chirp k from the hybrid user our reduction uses the public key class-hiding oracle $\mathsf{Sign}(sk_2^b, \cdot)$. For chirps $k+1$ and beyond from the hybrid user, we use the public key class-hiding oracle $\mathsf{Sign}(sk_1, \cdot)$. We can see that this makes our reduction look like one of these two hybrids $Hybrid^{B,\mathsf{HU}}_{i,j,k+b}$ depending on the bit for the public key class-hiding challenger. We can also replace the attribute sets in the commitment as we did for the proof of Claim 2, reducing to the hiding of the commitment scheme.

Claim 4. *$Hybrid^{B,\mathsf{CU}}_{i,j,q+1}$ is indistinguishable from $Hybrid^{B}_{i,j+1}$*

Proof of Claim 4 In this proof, we aren't simulating any new chirps in the batch, but rather the other elements such as $pk'_{\hat{\mathcal{U}}}$, $cert'$, Y^{pk}, π^{pk}, ... We can immediately use the indistinguishability property of VRFs/PRFs to replace all Y values with random values. This is because $\mathsf{PRF}_{(\cdot)}(\cdot)$ is a PRF and each invocation by an honest user depends on secrets not known to the adversaries (the user's secret key). Since the user is honest, we know that none of these Y values collide (they never listen for the same t at multiple locations and they never upload twice for a given $\mathsf{Epoch}(t)$[2]). We can then simulate all proofs π using the simulator for the NIZK. This includes running the simulator for the proof of subset. All C_i^* can then be randomly generated as shown by the simulator and any PRFs computed on them can be simulated.

Claim 5. *$Hybrid^{B}_{i,q+1}$ is indistinguishable from $Hybrid^{B}_{i+1,0}$*

Proof of Claim 5 From Claims 4, 3, and 2, we can see that Claim 1 is true. Now from Claim 1, the fact that $Hybrid^{\mathsf{c}}_{0,0} = Hybrid^{\mathsf{c}}_{q+1,q+1}$ (shown in the proof of chirp privacy), $Hybrid^{\mathsf{c}}_{q+1,q+1} = Hybrid^{B}_{0,0}$ as well as the fact that $Hybrid^{B}_{i,q+1} = Hybrid^{B}_{i+1,0}$, we find that $Hybrid^{\mathsf{c}}_{0,0}$ is indistinguishable from $Hybrid^{B}_{q+1,q+1}$.

Noticing that $Hybrid^{\mathsf{c}}_{0,0}$ is the real game and $Hybrid^{B}_{q+1,q+1}$ is the ideal game as well as using Claim 5 proves Theorem 2. □

C.2 Proofs for CoECs

Proof of Thm. 4 To prove Thm. 4, we split it into 4 theorems for correctness (Thm. 11), binding (Thm. 12), hiding (Thm. 13), and class-hiding (Thm. 14).

Theorem 11. *The commitment scheme in Definition 13 is correct as defined in Section 2.2.*

Proof of Thm. 11 The commitment scheme in Definition 13 is correct by inspection.

Theorem 12. *The commitment scheme in Definition 13 is binding as defined by Definition 11 under the discrete logarithm problem.*

[2]To ensure that no $t = \mathsf{Epoch}(t)$, we simply need to make the output of Epoch distinct from its input. Imagining the t values as a Unix timestamp, we can possibly make all outputs of Epoch negative or multiply the output by 2^{64} or similar.

Proof of Theorem 12 We assume an adversary can output (C, C', M, O, M', O') where $M \neq M'$, $[C]_{\mathsf{R}} = [C']_{\mathsf{R}}$ and both C, M, O and C', M', O' are valid openings. We assume the adversary outputs this double opening with non-negligible probability. To prove that this adversary cannot exist by contradiction, we will reduce to thediscrete logarithm problem. Let's first look at the scenario where there's only one pair of bases, A, B_0 $(s = 1)$. Because these were valid openings, we know that: $C = (A^{\alpha}, {B_0}^{\alpha m})$ $C' = (A^{\beta}, {B_0}^{\beta m'})$ where $O = \alpha$ and $O' = \beta$. If $[C]_{\mathsf{R}} = [C']_{\mathsf{R}}$, then $\log_{C_0}(C_1) = \log_{C_0'}(C_1') = \frac{b_0}{a}m = \frac{b_0}{a}m'$ where b_0 is the discrete log $\mathsf{dlog}_P(B_0)$ and a is the discrete log $\mathsf{dlog}_P(A)$. Thus, $m = m'$ so we have a contradiction. This implies that for one attribute, this scheme is perfectly binding. This changes to a computational hardness when we increase s.

Now we will prove this holds for $s > 1$. Because $M \neq M'$, we know $\exists j$ such that $m_j \neq m'_j$ where $M = (m_1, ..., m_{|M|})$ and $M' = (m'_1, ..., m'_{|M'|})$. Because $[C]_{\mathsf{R}} = [C']_{\mathsf{R}}$, we know that:

$$\log_{C_0}(C_1) = \log_{C_0'}(C_1')$$

$$\alpha(\sum m_i b_i)/\alpha a = \beta(\sum m'_i b_i)/\beta a$$

$$\alpha(\sum m_i b_i) * \beta a = \beta(\sum m'_i b_i) * \alpha a$$

$$\alpha\beta(\sum m_i b_i) * a = \alpha\beta(\sum m'_i b_i) * a$$

$$(\sum m_i b_i) = (\sum m'_i b_i)$$

$$0 = (\sum m'_i b_i) - (\sum m_i b_i)$$

$$m_j b_j - m'_j b_j = (\sum_{i \in [s]\setminus\{j\}} m'_i b_i) - (\sum_{i \in [s]\setminus\{j\}} m_i b_i)$$

$$(m_j - m'_j) b_j = (\sum_{i \in [s]\setminus\{j\}} m'_i b_i) - (\sum_{i \in [s]\setminus\{j\}} m_i b_i) \quad (7)$$

Where b_i is the discrete log $\mathsf{dlog}_P(B_i)$ and a is the discrete log $\mathsf{dlog}_P(A)$. Because we know $m_j - m'_j$ is non-zero, the left side of Equation 7 must also be non-zero. Thus, the right side must be non-zero.

Knowing that an adversary that can break the binding of our commitment scheme produces a set with the property described in Equation 7, we'll create a reduction that solves the discrete log problem with this. Our reduction to discrete log receives $P, P' = P^d$ and is tasked with recovering d. The reduction then creates a commitment scheme with a number (s) of random bases where the reduction knows all the discrete logs over P, i.e., we know all b_i such that: $(b_i = \mathsf{dlog}_P(B_i))$. The reduction then replaces one base, B_j, at random where the reductions instead replaces put in our challenge from the discrete logarithm game, $B_j = P'$, for a random $j \in [s]$. The reduction now gives $pp = \{A, B_i\}_{i \in [s]}$ to the adversary (sampling A randomly). Observing Equation 7, we can see that we retrieve some C, C', M, M', O, O' such that the adversary can compute:

$$(m_k - m'_k) B_0 = \sum_{i \in [s]\setminus\{k\}} (m'_i - m_i) B_i$$

Where k is the index of the attribute that the adversary has changed (which must exist due to this being a double opening). The reduction reruns this experiment until it happens to pick $j = k$. This will happen with non-negligible probability since the j base is a randomly sampled element just like the other bases: $\{P^d : d \xleftarrow{\$} \mathbb{Z}_p^*\} \approx \{Q : Q \xleftarrow{\$} \mathbb{G}\}$. So for the rest of the proof, we'll assume $j = k$. Because the reduction knows the discrete logs of all the other bases, the reduction can find d' such that:

$${B_j}^{m_j - m'_j} = P^{\sum_{i \in [s]\setminus\{j\}} (m'_i - m_i) b_i}$$

$$B_j = P^{\sum_{i\in[s]\setminus\{j\}} b_i(m'_i - m_i)/(m_j - m'_j)}$$

$$B_j = P^{d'}$$

We can divide by $(m_j - m'_j)$ if these values are distinct, which is true because we know this is where the message sets, M and M' differ. Thus, because we set $B_j = P'$, we can recompute $d' = \sum_{i\in[s]\setminus\{j\}} b_i(m'_i - m_i)/(m_0 - m'_0)$ since we know all the b_i values aside from b_j. Thus, we find the discrete log: $d' = \mathsf{dlog}_P(P')$.

Theorem 13. *The commitment scheme in Definition 13 is hiding as defined in Section 2.2 as long as the decisional Diffie-Hellman assumption holds.*

Proof of 13 Our reduction takes in $(X, Y, Z) = (P^a, P^b, P^c)$ from the DDH challenger. The reduction then computes parameters pp_1 for the commitment scheme by first sampling random values, d, r_i for $i \in [s]$ and computing $A = X^d$ and $B_i = Y^{r_i}$ and setting $pp_1 = (A, \{B_i\}_{i\in[s]})$. Notice that $\log_A(B_i) = ar_i/d$. The reduction gives pp_1 to the adversary in the commitment hiding game and this adversary returns two vectors of messages that they want the reduction to commit to: M_0, M_1. The reduction then flips a coin, $b' \in \{0, 1\}$ and commits to $M_{b'}$ using a second set of parameters: $pp_2 = (A', \{B'_i\}_{i\in[s]})$ where $A' = Y^d$ and $B'_i = Z^{r_i}$. We label this commitment $C = (C_0, C_1)$. Notice that if $c = ab$, then $\log_{A'}(B'_i) = ar_i/d$, which is the same discrete log as pp_1. Thus, because the equivalence class of commitments is defined by the discrete log between elements, any commitments based on these parameters will look identical to commitments using pp_1. If $c \neq ab$, then $\log_{A'}(B'_i) = cr_i/(bd)$. Thus, the discrete log will be dependent on c. We can see that the discrete log of C_1 over C_0 will be: $\log_{C_0}(C_1) = (c/b)(\sum_{i\in[s]} M_{b',i}r_i)/(\sum_{i\in[s]} d)$ where $M_{b',i}$ is the i-th message in the vector, $M_{b'}$. If $c \neq ab$, then this discrete log will be a random value that the adversary has never seen before. Because we pick a random representation uniformly, and the equivalence class is defined by the discrete log, then only the discrete log matters when the adversary is distinguishing this commitment. Because the discrete log is a random value when $c \neq ab$, then the adversary has no advantage. Thus, if the adversary can guess the reduction's bit, b', then the reduction guesses that $c = ab$ and otherwise, guesses that $c \neq ab$. We can see that if this assumed adversary has non-negligible success at distinguishing in the commitment hiding game, then our reduction has a non-negligible chance of winning the decisional Diffie-Hellman game.

Theorem 14. *The commitment scheme in Definition 13 is class-hiding as defined by Definition 12 as long as*

Proof of Theorem 14 Our reduction takes in a Diffie-Hellman tuple, $(X, Y, Z) = (P^a, P^b, P^c)$, and decides if $c = ab$. Similar to the proof of Theorem 13, the reduction constructs two sets of parameters: (1) $pp_1 = \{A, B_i\}_{i\in[s]}$ where $A = Y^d$ and $B_i = Z^{r_i}$; (2) $pp_2 = \{A', B'_i\}_{i\in[s]}$ where $A' = Y^d$ and $B'_i = Z^{r_i}$. The reduction gives pp_1 to the adversary and retrieves C, O, M, C', O', M'. The reduction then generates a new commitment C_0 and C_1 to M and M' (respectively) using pp_2 and gives one of these randomly to the adversary. If $M = M'$, then, because randomizing a commitment perfectly chooses a random commitment in the same equivalence class, C and C' are indistinguishable. If $M \neq M'$, then we can instead reduce using our previous reduction from the proof of Theorem 13. This is done passing the adversary's M, M' to that reduction to mercurial message class-hiding and returning the given $C^{b'}$ to the class-hiding adversary and having the reduction return $b^\dagger$ dependent on whether the adversary guesses correctly just like the proof of Theorem 13. Again, because randomizing a commitment perfectly chooses another representative in the equivalence class, being able to specify the opening (O, O')

does not help the adversary distinguish this reduction since the commitments returns to the adversary will always be entirely independent of O, O'.

C.3 Proofs for our no cloning construction in Section 4.3.1

Proof of Theorem 5 Our scheme is correct by inspection.

Proof of Theorem 7 Intuitively, since $\mathsf{VerifSubset}(pp, \{C'_i\}_{i\in[m]}, \{B'_i\}_{i\in[m]}, \pi_{\subset}) = 1$, we know that for each $\mathbf{M}_i$ committed to with C'_i, there exists a $\mathbf{L}_i$ committed to by a B'_i. If this is not the case, we can create a reduction that breaks the soundness of the underlying zk-SPoC scheme. Because we verify each π_i and check that each Y_i is distinct, this ensures that no two $\mathbf{L}_i[\mathsf{idx_t}]$ are the same. If $\exists i, j : i \neq j \wedge \mathbf{L}_i[\mathsf{idx_t}] = \mathbf{L}_j[\mathsf{idx_t}]$, then the Y_i would be the same since we've computed and proven each Y_i using the same public VRF key, pk_{VRF}. Otherwise, the adversary has broken the soundness of the NIZK for the VRF scheme or the binding of the commitment. Because the zero knowledge proofs in our no cloning construction are simulation extractable, our no cloning proof is also simulation extractable.

More formally, we see that the adversary proves that the pairs of times and locations committed to by the commitments, $\{C'_i\}_{i\in[n]}$, are a subset of the pairs committed to by $\{B'_i\}_{i\in[n]}$ in π_{clone}. Thus, if any of the pairs in $\{C'_i\}_{i\in[n]}$ collide such that $\mathbf{M}_i[\mathsf{idx_t}] = \mathbf{M}_j[\mathsf{idx_t}], \mathbf{M}_i[\mathsf{idx_l}] \neq \mathbf{M}_j[\mathsf{idx_l}]$, then for $\{B'_i\}_{i\in[n]}$ to be a super set of the messages, it must include two pairs where $\mathbf{L}_i[\mathsf{idx_t}] = \mathbf{L}_j[\mathsf{idx_t}]$ (as it must include both pairs, $(\mathbf{M}_i[\mathsf{idx_t}], \mathbf{M}_i[\mathsf{idx_l}])$ and $(\mathbf{M}_j[\mathsf{idx_t}], \mathbf{M}_j[\mathsf{idx_l}])$). We also see that $Y_{t,i} = \mathsf{PRF}_{sk_{\mathcal{U}}}(\mathbf{L}_i[\mathsf{idx_t}])$ in π_{clone} is computed for each $\mathbf{L}_i[\mathsf{idx_t}]$ committed to by $\{B'_i\}_{i\in[n]}$. Thus, because each $Y_{i,t}$ is computed solely on the time value, if these values are computed honestly, we'll see that $Y_{t,i} = Y_{t,j}$ (for $i \neq j$) and the VerifyNoCloning function will reject this proof. If an adversary were to include a false $Y_{t,i}$ or $Y_{t,j}$ then a reduction would be able to extract a fake proof for at least one of the VRFs, $\pi_{t,i}$ or $\pi_{t,j}$. Because the verifier also verifies that the time and location values comitted to by $\{C'_i\}_{i\in[n]}$ are equivalent to the values time and location values in $\{C_i\}_{i\in[n]}$, we know that this relation of no-cloning holds for the inputted commitments, $\{C_i\}_{i\in[n]}$.

Proof of Theorem 6 The commitments in the proof output in Figure 7 can be randomly sampled elements. The NIZK proofs can be simulated so that a simulator can open these commitments to any value. Thus, we can commit to random attribute vectors and simulate all the proofs. If the adversary can distinguish, we can break either the zero knowledge of the NIZK scheme or the hiding of the commitments. Because each VRF is distinct and independent when replaced with a truely random function, our simulator can sample the VRF keys and generate the VRF outputs on the randomly sampled attribute vectors.

C.4 Proofs for zk-SPoCs

Proof of Theorem 9. Let $\mathcal{S}^{\mathsf{ProveSubset}}$ be a simulator as defined in Def. 27:

Definition 27 (zk-SPoC simulator - $\mathcal{S}^{\mathsf{ProveSubset}}(\{A_i\}_{i\in[m]}, \{B_i\}_{i\in[n]})$**.** Create random commitments, messages, and openings: $C, D, \{A'_i, L'_i, O'_i\}_{i\in[n]}, \{B'_i, M'_i, P'_i\}_{i\in[n]}$ such that the messages and opening satisfy the relationship. Next, simulate the other NIZK proofs output by $\mathsf{ProveSubset}_{\mathsf{FHS}}$, $\pi_{\{B_i\}\approx D}, \pi', \pi_{A_i\subset C}, \pi_{C\subset D}$.

Because FHS commitments are perfectly hiding, randomly sampling commitments implies that there exists a valid witness that satisfy the relations. Thus, we can use the eqrep simulator to create simulated proofs while keeping our simulator indistinguishable to any PPT adversary.

Specifically, we can first create hybrids to prove that the real proof function is indistinguishable from when the proofs are simulated. This is done by replacing each proof

one-by-one and reducing to the respective zero knowledge challenger. Thus, we can replace the proofs generated by $\mathsf{ProveSubset}_{\mathsf{FHS}}$ as well as the NIZK, $\pi_{\{B_i\}\approx D}$, and the proofs of correct concatenation, π'. After the proofs have been replaced, we can replace all the commitments with random group elements. These are indistinguishable from correct commitments as FHS set commitments are perfectly hiding. This last case is exactly our simulator, which is independent of witnesses, and thus, we have zero knowledge.

Proof of Theorem 10. If an adversary can produce a π that satisfies $R_{\subset}$ defined in Section 4.3.2, we can extract a witness that proves the statement (i.e. that $\{A_i\}_{i\in[n]}$ is a subset of $\{B_i\}_{i\in[n]}$. We first note that because the concatenation function (described in Section A.3.1) is bijective and maps input the space from which the A_i, B_i vectors are drawn, and maps to the space from which the A'_i, B'_i values are drawn, proving the relation for A'_i, B'_i will ensure the relation is true for A_i, B_i. π' ensures the concatenation relationship between A_i, B_i and A'_i, B'_i holds. We can see that using each $\pi_{A_i\subset C}$ (generated by $\mathsf{ProveSubset}_{\mathsf{FHS}}$), we can extract witnesses to prove that C is committed to all the messages in $\{A_i\}_{i\in[n]}$. We can then see that using $\pi_{C\subset D}$, we prove that the set that C is comitted to is a subset of the set that D is committed to. We then use $\pi_{\{B_i\}\approx D}$ to extract the witnesses that prove that D and $\{B'_i\}_{i\in[n]}$ are committed to the same set, $\{M'_i\}_{i\in[n]}$. If an adversary were to attempt to use a fake witness in this proof, we can create a reduction that violates the subset soundness of FHS set commitments (in Def. 25). Thus, we've proven that $\{L'_i\}_{i\in[n]} \subset \{M'_i\}_{i\in[n]}$ and because of the bijectivity of the concatenation function described earlier, we see that $\{L_i\}_{i\in[n]} \subset \{M_i\}_{i\in[n]}$ (where these are the messages that $\{A_i, B_i\}_{i\in[n]}$ are committed to).