Inverse-Sybil Attacks in Automated Contact Tracing

The Crypto Group at IST Austria*

June 4, 2020

Abstract

Automated contract tracing aims at helping with the current COVID-19 pandemic by alerting users of encounters with infected people. There are currently many proposals for protocols (like the "decentralized" DP-3T and PACT or the "centralized" ROBERT and DESIRE) to be run on mobile phones, where the basic idea is to regularly broadcast (using low energy Bluetooth) some values, and at the same time store (a function of) incoming messages broadcasted by users in their proximity. Should a user get diagnosed, he can upload some data to a backend server, which then is used to alert users that were in proximity with the diagnosed user.

There are many important aspects one wants those protocols to achieve, in particular simplicity/efficiency, privacy and robustness, the latter including some security against false positives, that is, users getting alerts despite not having been in proximity with a diagnosed user.

In the existing proposals one can trigger false positives on a massive scale by an "inverse-Sybil" attack, where a large number of devices (malicious users or hacked phones) pretend to be the same user, such that later, just a single person needs to be diagnosed (and allowed to upload) to trigger an alert for all users that were in proximity to any of this large group of devices.

We propose the first protocols that do not succumb to such attacks assuming the devices involved in the attack do not constantly communicate, which we observe is a necessary assumption. Our first protocol requires devices to non-interactively exchange values (like e.g. in DESIRE), while the second requires that the devices have access to some location dependent coordinate (like coarse GPS coordinates or cell tower IDs).

The high level idea of the protocols is to derive the values to be broadcasted by a hash chain, so that two (or more) devices who want to launch an inverse Sybil attack will not be able to connect their respective chains and thus only one of them will be able to upload. Apart from achieving strong privacy and good efficiency, a main challenge is to force the chains on different devices to divert, which we do by infusing unpredictable data (randomness from encounters in the first, location data in the second protocol). Our protocols also achieve security against replay, and the second even against relay attacks.

Contents

1 Introduction **2**
- 1.1 Automated Contact Tracing . . . 2
- 1.2 False Positives . . . 2
  - 1.2.1 Replay Attacks . . . 3
  - 1.2.2 Relay Attacks . . . 3
  - 1.2.3 Inverse Sybil Attacks . . . 3
  - 1.2.4 Modelling the Crown Affair . . . 4
  - 1.2.5 On the Assumption that Devices Can't Communicate . . . 5
  - 1.2.6 Using Hash Chains to Prevent the Crown Affair . . . 6

*Several contributors to this project have received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (682815 - TOCNeT), corresponding author pietrzak@ist.ac.at

2 Protocol 1: Decentralized, Non-Interactive Exchange **6**
2.1 Toy Protocol 6
2.2 Description of Protocol 1 8

3 Security of Protocol 1 **11**
3.1 Security of the toy version of Protocol 1 11
3.2 Security of Protocol 1 12

4 Protocol 2: Decentralized, Using Location for Chaining **13**
4.1 Protocol Description 13
4.2 Correctness, Privacy and Epochs 14

5 Security of Protocol 2 **16**
5.1 Security against replay and relay attacks 16
5.2 Security against a Crown Affair 16
5.2.1 A Weaker Security Model 16
5.2.2 Security Proof 16

A Protocol 1 with efficient risk-evaluation **18**

1 Introduction

1.1 Automated Contact Tracing

One central element in managing the current Covid-19 pandemic is contact tracing, which aims at identifying individuals that were in contact with diagnosed people so they can be warned and further spread can be prevented. While contact tracing is done mostly manually, there are many projects which develop automated contact tracing tools leveraging the fact that many people carry mobile phones around most of the time.

While some early tracing apps used GPS coordinates, most ongoing efforts bet on low energy Bluetooth to identify proximity of devices. Some of the larger projects include east [ePA20, CTV20] and west coast PACT [CGH+20], Covid Watch [CW20], DP-3T [TPH+20], Robert [Rob20], its successor Desire [des20] and Pepp-PT [Pep20]. Google and Apple [app20] released an API for Android and iOS phones which solves some issues earlier apps had (in particular, using Bluetooth in the background and synchronising Bluetooth MAC rotations with other key rotations). As this API is fairly specific its use is limited to basically the DP-3T protocol.

Coming up with a practical protocol is challenging. The protocol should be **simple** and **efficient** enough to be implemented in short time and using just low energy Bluetooth. As the usage of an app should be voluntarily, the app should provide strong **privacy** and **security** guarantees to not disincentivize people from using it.

1.2 False Positives

One important security aspect is preventing false positives, that is, having a user's device trigger an alert even though he was not in proximity with a diagnosed user. Triggering false positives cannot be completely prevented, a dedicated adversary will always be able to e.g. "borrow" the phone of a person who shows symptoms and bring it into proximity of users he wants to get alerted. What is more worrying are attacks which either are much easier to launch or that can easily be scaled. If such large scale attacks should happen they will likely undermine trust and thus deployment of the app. There are individuals and even some authoritarian states that actively try to undermine efforts to contain the epidemic, at this point mostly by disinformation[1], but potential low-cost large-scale attacks on tracing apps would also make a worthy target.

[1] `https://www.aies.at/download/2020/AIES-Fokus-2020-03.pdf`

1.2.1 Replay Attacks

One such attack are replay attacks, where an adversary simply records the message broadcasted by the device of a user Alice, and can later replays this broadcast (potentially after altering it) to some user Bob, such that Bob will be alerted should Alice report sick. Such an attack is clearly much easier to launch and scale than "borrowing" the device of Alice. One way to prevent replay attacks without compromising privacy but somewhat in efficiency and simplicity is by interaction [Vau20a] or at least non-interactive message exchange [des20, ABIV20]. The Google-Apple API [app20] implicitly stores and authenticates the epoch of each encounter (basically, the time rounded to 15 minutes) to achieve some security against replay attacks, thus giving up a lot in privacy to prevent replaying messages that are older than 15 minutes. This still leaves a lot of room for replays, in particular if combined with relaying messages as discussed next. A way to prevent replay attacks by authenticating the time of the exchange without ever storing this sensitive data, termed "delayed authentication", was suggested in [Pie20].

1.2.2 Relay Attacks

Even if replay attacks are not possible (e.g. because one uses message exchange [Vau20a, des20, ABIV20] or a message can only trigger an alert if replayed right away [Pie20]) existing schemes can still succumb to relay attacks, where the messages received by one device are sent to some other device far away, to be replayed there. This attack is more difficult to launch than a replay attack, but also more difficult to protect against. The only proposal we are aware of which aims at preventing them [Vau20a, Gvi20, Pie20] require some kind of location dependent value like coarse GPS coordinates or cell tower IDs.

1.2.3 Inverse Sybil Attacks

While replay and relay attacks on tracing apps have already received some attention, "inverse-Sybil" attacks seem at least as devastating but have attained little attention so far. In such attacks, many different devices pretend to be just one user, so that later it's sufficient that a single person is diagnosed and can upload its values in order to alert all users who were in proximity to any of the many devices. The devices involved in such an attack could belong to malicious covidiots, or to honest users whose phones got hacked. We'll refer to this attack as a Crown Affair after the 1999 movie *The Thomas Crown Affair* where *"Crown arrives but quickly blends into the crowd, aided by lookalikes in bowler hats"*.[2] Below we shortly discuss how this attack affects the various types of tracing protocols suggested, the discussion borrows from Vaudenay [Vau20b], where this attack is called a "terrorist attack". The attacks are illustrated in Figure 2.

Decentralized. In so called decentralized schemes like DP-3T [TPH$^+$20], devices regularly broadcast ephemeral IDs derived from some initial key K, and also store IDs broadcasted by other devices in their proximity. If diagnosed, the devices upload their keys K to a backend server. The devices daily download keys of infected users from the server and check if they have locally stored any of the IDs corresponding to those keys. If yes, the devices raise an alert.[3]

It's particularly easy to launch a Crown Affair against decenralized schemes, one just needs to initialize the attacking devices with the same initial key K.

Centralized. Centralized schemes like Robert [Rob20] are similar, but here an infected user uploads the received (not the broadcasted) IDs to the server, who then informs the senders of those broadcast about their risk. To launch a Crown Affair against such schemes the attacking devices don't need to be initialized with the same key, in fact, they don't need to broadcast anything at all. Before uploading to the server, the attacker simply collects the messages received from any devices he gets his hands on, and uploads all of them.

[2] `en.wikipedia.org/wiki/The_Thomas_Crown_Affair_(1999_film)`

[3] This oversimplifies things, in reality a risk score is computed based on the number, duration, signal strength etc., of the encounters, which then may or may not raise an alert. How the risk is computed is of course crucial, but not important for this work.

As in centralized schemes the server learns the number of encounters, he can set an upper bound on the number of encounters a single diagnosed user can upload, which makes a Crown Affair much less scalable.

Non-interactive exchange. Schemes including Desire [des20] and Pronto-C2 [ABIV20] require the devices to exchange messages (say X and Y) at an encounter, and from these then compute a shared token $S = f(X, Y)$.[4]

In Desire and Pronto-C2 the token is derived by a non-interactive exchange (NIKE), concretely, a Diffie-Hellman exchange $X = g^x, Y = g^y, S = g^{xy}$. The goal is to prevent a user who passively records the exchange to learn S.

Our first Protocol also uses an exchange, but for a different goal, and for us it's sufficient to just use hashing $S = H(X, Y)$ to derive the token.

The Crown Affair as described above also can also be launched against schemes that use a non-interactive exchange, but now the attack devices need to be active (i.e., broadcast, not just record) during the attack also for centralized schemes like Desire.

1.2.4 Modelling the Crown Affair

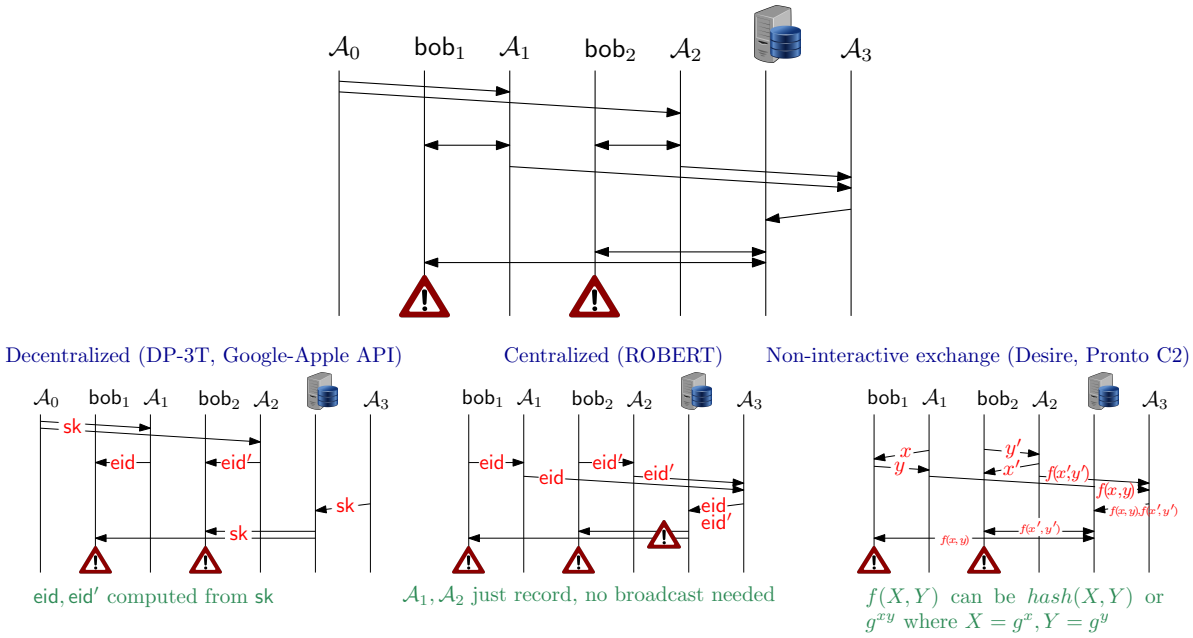

Figure 1: (top) Illustration of a successful Crown Affair: both bob's trigger an alert even though they interacted with different devices $\mathcal{A}_1, \mathcal{A}_2$. (bottom) The attacks on the various protocol types outlined in §1.2.3

Our simplest security notion for the Crown Affair (which we'll use for the toy version of our Protocol 1) considers an adversary which consists of four parts $(\mathcal{A}_0, \mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3)$ where

- $\mathcal{A}_0$ chooses initial values for $\mathcal{A}_1, \mathcal{A}_2$.

[4] In Desire it's called a "private encounter token" (PET), and is uploaded to the server for a risk assesment (so it's a more centralized scheme), while in Pronto-C2 only diagnosed users upload the tokens, which are then downloaded by all other devices to make the assessment on their phones (so a more decentralized scheme).

- $\mathcal{A}_1$ and $\mathcal{A}_2$ interact with honest devices bob_1 and bob_2.
- $\mathcal{A}_1$ and $\mathcal{A}_2$ pass their state to $\mathcal{A}_3$.
- $\mathcal{A}_3$ is then allowed one upload some combined state of the devices to the backend server (like a diagnosed user).
- The adversary wins the game if both, bob_1 and bob_2, raise an alert after interacting with the backend server.

The notions we achieve for our actual Protocols are a bit weaker, in particular, in Protocol 1 the adversary can combine a small number from two devices (basically the encounters in the first epoch) and in our Protocol 2 we nee to assume that the locations of the devices in the future are not already known when the attack starts.

1.2.5 On the Assumption that Devices Can't Communicate

Let us stress that in the security game above we do not allow $\mathcal{A}_1$ and $\mathcal{A}_2$ to communicate. The reason is that a successful Crown Affair seems unavoidable if such communication was allowed: $\mathcal{A}_1$ can simply send its entire state to $\mathcal{A}_2$ after interacting with bob_1, who then interacts with bob_2. The final state of $\mathcal{A}_2$ has the distribution of a single device $\mathcal{C}$ first interacting with bob_1 then with bob_2, and if this was the case we want both bob's to trigger an alert, this is illustrated in Figure 2. Thus, presumably the best we can hope for is a protocol which prevents a Crown Affair assuming the devices involved in the attack do not communicate. Such an assumption might be justified if one considers the case where the attack is launched by hacked

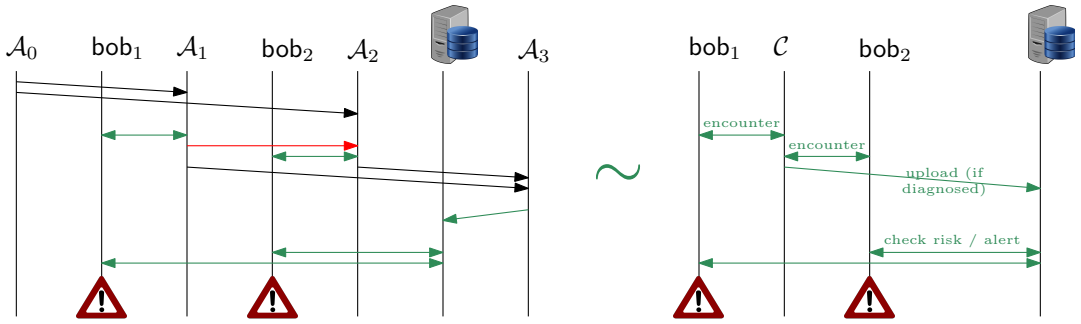

Figure 2: If the attacking devices $\mathcal{A}_1, \mathcal{A}_2$ could communicate during the attack (arrow in red), they can emulate the transcript (shown in green) of a single honest device $\mathcal{C}$ interacting with $\mathsf{bob}_1, \mathsf{bob}_2$ and the server. As such a $\mathcal{C}$ can make both bob's trigger an alert by running the honest protocol, the adversary can always win the game.

devices, as such communication might be hard or at least easily detectable.

If we can't exclude communication, we note that the security of our schemes degrades gracefully with the frequency in which such communication is possible. Basically, in our first protocol, at every point in time at most one of the devices will be able to have interactions with other devices which later will trigger an alert, and moving this "token" from one device to another requires communication between the two devices. In our second protocol the token can only be passed once per epoch, but on the downside, several devices can be active at the same time as long as they are at the same location dependent coordinate.

Only under very strong additional assumptions, in particular if the devices run trusted hardware, the Crown Affair can be prevented in various ways, even if the devices can communicate.

1.2.6 Using Hash Chains to Prevent the Crown Affair

Below we outline the two proposed protocols which do not succumb to the Crown Affair. The basic idea is to force the devices to derive their broadcasted values from a hash chain. If diagnosed, a user will upload the chain to the server, who will then verify it's indeed a proper hash chain.

The main problem with this idea is that one needs to force the chains of different attacking devices to diverge, so later, when the adversary can upload a chain, only users who interacted with the device creating that particular chain will raise an alert. To enforce diverting chains, we will make the devices infuse unpredictable values to their chains. We propose two ways of doing this, both protocol are decentralized (i.e., the risk assessment is done on the devices), but it's straight forward to change them to centralized variants.

Protocol 1 (§ 2, decentralized, non-interactive exchange) The basic idea of our first proposal is to let devices exchange some randomness at an encounter, together with the heads of their hash chains. The received randomness must then be used to progress the hash chain. Should a user be diagnosed and his hash-chain is uploaded, the other device can verify the randomness it chose was used to progress that chain from the head it received. A toy version of this protocol is illustrated in Figure 3 (the encounter) and 4 (report and alert). In this protocol the amount a diagnosed user has to upload, and more importantly, every other user needs to download, is linear in the number of encounters a diagnosed user had. In Appendix A we describe a variant of Protocol 1 where the up and downloads are independent of the number of encounters, but which has weaker privacy properties.

Protocol 2 (§ 4, decentralized, location based coordinate) Our second protocol is similar to simple protocols like the unlikeable DP-3T, but the broadcasted values are derived via a hash-chain (not sampled at random). We also need the device to measure some location dependent coordinate with every epoch which is then infused to the hash chain at every epoch. Apart from the need of a location dependent coordinate, the scheme is basically as efficient as the unlinkable variant of DP-3T. In particular, no message exchanges are necessary and the upload by a diagnosed user is linear in the number of epochs, but independent of the number of encounters.

The Privacy Cost of Hash Chains. In our protocols diagnosed users must upload the hash chain to the backend server, who then checks if the uploaded values indeed form a hash chain. This immediately raises serious privacy concerns, but we'll argue that the privacy cost of our protocols is fairly minor; apart from the fact that the server can learn an ordering of the uploaded values (which then would give some extra information should the server collude with other users), the protocols provide the same privacy guarantees as their underlying protocols without the chaining (which do not provide security against the Crown Affair).

2 Protocol 1: Decentralized, Non-Interactive Exchange

In this section we describe our first contact tracing protocol using hash chains which does not succumb to the Crown Affair. To illustrate the main idea behind the protocol, in Section 2.1 we'll consider a toy version of the protocol which assumes a (unrealistic) restricted communication model. We will then describe and motivate the changes to the protocol required to make the protocol private and correct in a general communication model.

2.1 Toy Protocol

The description of our toy protocol is given below, its broadcast/receive phase is additionally depicted in Figure 3, and its report/evaluate phase in Figure 4. To analyze it, we'll make the (unrealistic) assumptions that all parties proceed in the protocol in a synchronized manner, i.e., messages between two parties are broadcast and received at the same time, and consider a setting where users meet in pairs: a broadcasted message from user A is received by at most one other user B, and in this case also A receives the message from B.

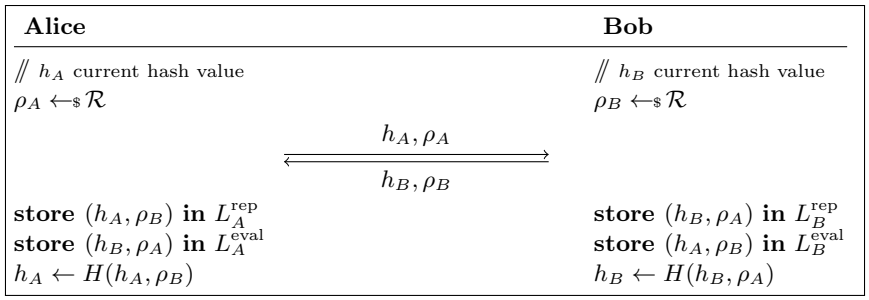

Figure 3: Broadcast/receive phase of the toy protocol.

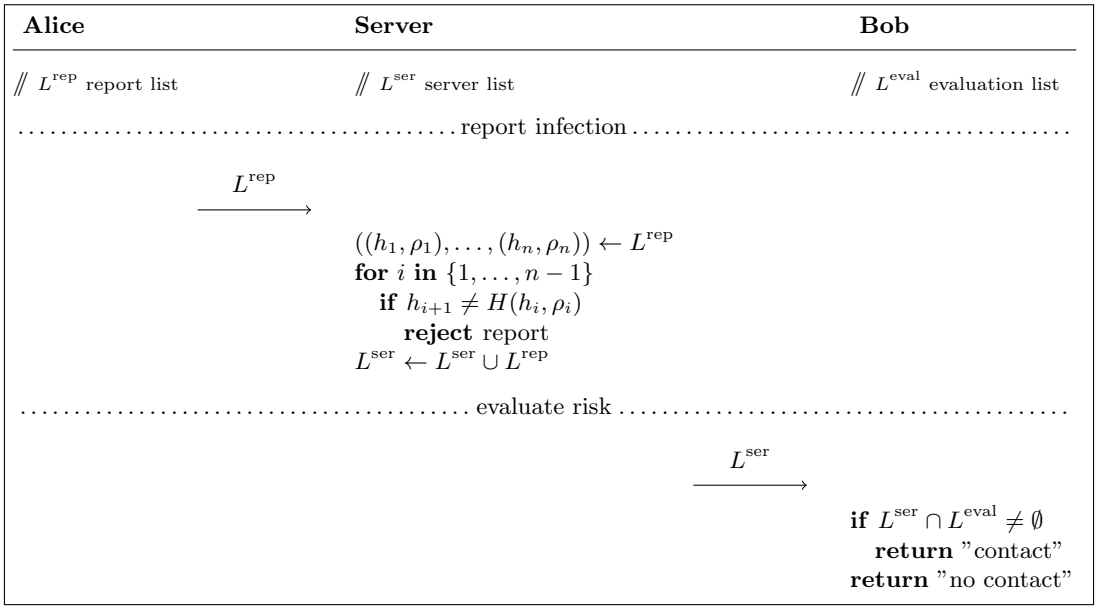

Figure 4: Report/risk-evaluation phase of the toy protocol.

- (setup) Users sample a genesis hash value h_1 and set the current head of the hash chain to $h \leftarrow h_1$. Then they initialize empty lists L^{rep} and L^{eval} which are used to store information to be reported to the backend server in case of infection or used to evaluate whether contact with an infected person occurred, respectively.
- (broadcast) In regular intervals each user samples a random string $\rho \leftarrow_\$ \mathcal{R}$ and broadcasts the message (h, ρ) where h is the current head of the hash chain.
- (receive broadcast message) When Alice with current hash value h_A receives a broadcast message (h_B, ρ_B) from Bob she proceeds as follows. She stores the pair (h_A, ρ_B) in L^{rep} and (h_B, ρ_A) in L^{eval}. Then she computes the new head of the hash chain as $h_A \leftarrow H(h_A, \rho_B)$.
- (report message to backend server) When diagnosed users can upload the list $L^{\text{rep}} = ((h_1, \rho_1), \ldots, (h_n, \rho_n))$ to the server. The server verifies that the uploaded values indeed form a hash chain, i.e., that $h_{i+1} = H(h_i, \rho_i)$ for all $i \in \{1, \ldots, n-1\}$. If the uploaded values pass this check the server includes all elements of L^{rep} to the list L^{ser}.
- (evaluate infection risk) After downloading L^{ser} from the server users check whether L^{ser} contains any of the hash-randomness pairs stored in L^{eval}. If this is the case they assume that they were in contact with an infected party.

Security. In Section 3 we provide a formal security model for the Crown Affair and give a proof of the security of the toy protocol in this model.

Correctness. Assume that Alice and Bob met and simultaneously exchanged messages (h_A, ρ_A) and (h_B, ρ_B). Then the pair (h_A, ρ_B) is stored by Alice in L^{rep} and by Bob in L^{eval}. If Alice later is diagnosed and uploads L^{rep}, Bob will learn that he was in contact with an infected person.

This toy protocol cannot handle simultaneous encounters of more than two parties. For example, assume both, Bob and Charlie, received Alice's message (h_A, ρ_A) at the same time. Even if Alice records both messages, it's not clear how to process both. We could let Alice process both sequentially, say first Bob's message as $h'_A \leftarrow H(h_A, \rho_B)$, and then Charlie's $h''_A \leftarrow H(h'_A, \rho_C)$. Then later, should Alice be diagnosed and upload $(h_A, \rho_B), (h'_A, \rho_C)$, Charlie who stored (h_A, ρ_C) will get a false negative and not recognize the encounter.

Our full protocol overcomes this issue by advancing in epochs. The randomness broadcast by other parties is collected in a pool that at the end of the current epoch is used to extend the hash chain by one link.

Privacy. The toy protocol is a minimal solution to prevent a Crown Affair but has several weaknesses regarding privacy. Below, we discuss some privacy issues of the toy protocol, and how they are addressed in Protocol 1

(i) Problem (Reconstruction of chains): After learning the list L^{ser} from the server, a user Bob is able to reconstruct the hash chains contained in this list even if the tuples in L^{ser} are randomly permuted: check for each pair $(h, \rho), (h', \rho) \in L^{\text{ser}}$ if $h' = H(h, \rho)$ to identify all chain links. If a reconstructed chain can be linked to a user, this reveals how many encounters this user had. Moreover a user can determine the position in this chain where it had encounters with this person.
Solution (Keyed hash function): We use a keyed hash function so Bob can't evaluate the hash function. Let us stress that this will not improve privacy against a malicious server because the sever is given the hashing key as it must verify the uploaded values indeed form a chain. Leakage of the ordering of encounters is the prize we pay in privacy for preventing the Crown Affair.

(ii) Problem (Correlated uploads): Parallel encounters are not just a problem for correctness as we discussed above, but also privacy. If both Bob and Charlie met Alice at the same time, both will receive the same message (h_A, ρ_A). Bob will then store (h_B, ρ_A) and Charlie (h_C, ρ_A) in their L^{rep} list. This is bad for privacy, for example if both, Bob and Charlie, later are diagnosed and upload their L^{rep} lists, (at least) the server will see that they both uploaded the same ρ_A, and thus they must have been in proximity.
Solution (Unique chaining values): The chaining value (σ_A in Figure 5 below) in Protocol 1 is not just the received randomness as in the toy protocol, but a hash of the received randomness and the heads of the hash chains of both parties. This ensures that all the L^{rep} lists (containing the chains users will upload if diagnosed) simply look like random and independent hash chains.

2.2 Description of Protocol 1

We now describe our actual hash-chain based protocol, unlike the toy protocol it proceeds in epochs. Users broadcast the same message during the full duration of an epoch and pool incoming messages in a set that is used to update the hash chain at the beginning of the next epoch. The protocol makes use of three hash functions H_1, H_2, and H_3. It is additionally parametrized by an integer γ that serves as an upper limit on the number of contacts that can be processed per epoch. Its formal description is given below. Its broadcast/receive phase is additionally depicted in Figure 5, and its report/evaluate phase in Figure 6.

- (setup) A user Alice samples a key k_A and a genesis hash value h_1. She setsthe current head of the hash chain to $h \leftarrow h_1$. Then they initializes empty lists L^{rep} and L^{eval} which are used to store information

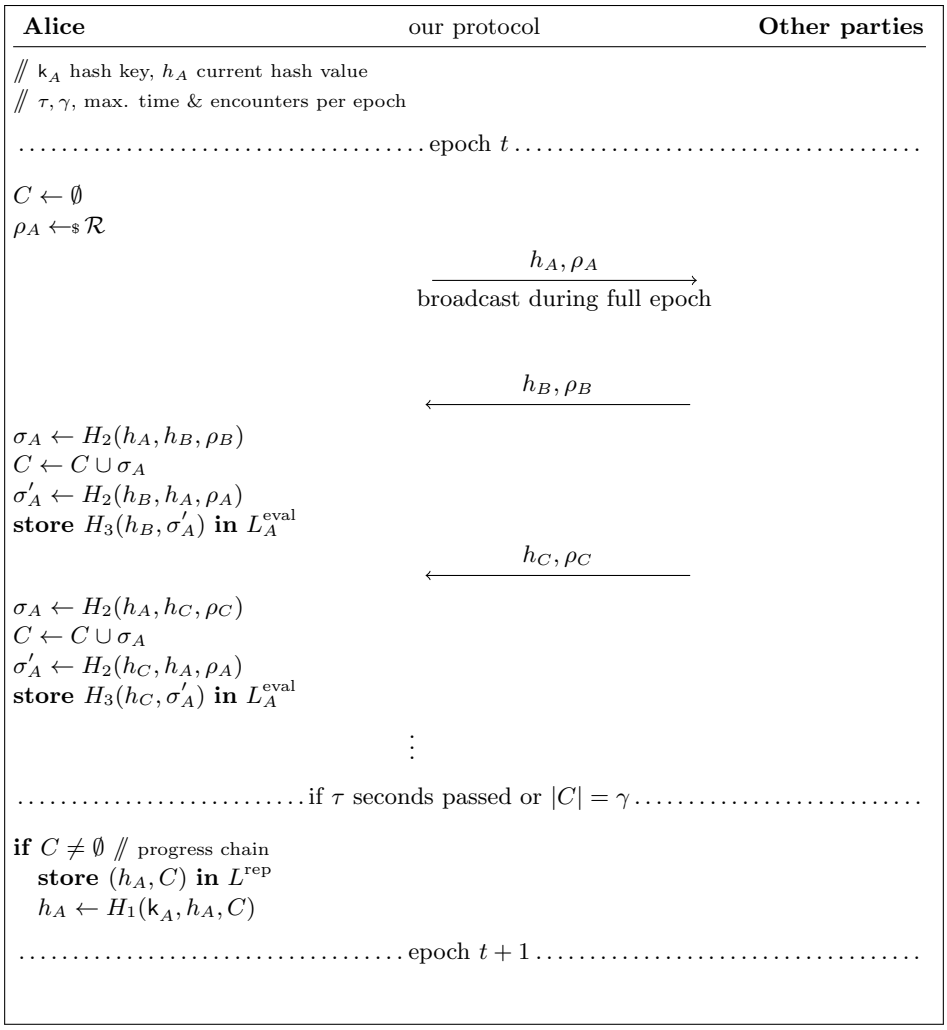

Figure 5: Broadcast/receive phase of Protocol 1

to be reported to the backend server in case of infection or used to evaluate whether contact with an infected person occurred, respectively.

- (broadcast) At the beginning of every epoch Alice samples a random string $\rho_A \leftarrow_\$ \mathcal{R}$ and sets C to the empty set. She broadcasts the message (h_A, ρ_A) consisting of the current head of the hash chain and this randomness during the full duration of the epoch.

- (receive broadcast message) Let h_A denote her current head of the hash chain. Whenever she receives a broadcast message (h_B, ρ_B) from Bob she proceeds as follows. She computes $\sigma_A \leftarrow H_2(h_A, h_B, \rho_B)$ and adds σ_A to the set C. Then she computes the value $\sigma'_A \leftarrow H_3(h_B, h_A, \rho_A)$ and stores it in L^{eval}.

- (end of epoch) When the epoch ends (we discuss when that should happen), Alice stores the tuple (h_A, C) in L^{rep} and updates the hash chain using C as $h_A \leftarrow H_1(\mathsf{k}_A, h_A, C)$, but for efficiency reasons only if they received at least one broadcast, i.e., $C \neq \emptyset$.

- (report message to backend server) In diagnosed, Alice is allowed to upload her key k_A and the list $L^{\text{rep}} = ((h_1, C_1), \ldots, (h_n, C_n))$ to the server. The server verifies that the uploaded values indeed form a hash chain, i.e., that $h_{i+1} = H(\mathsf{k}, h_i, C_i)$ for all $i \in \{1, \ldots, n-1\}$. If the uploaded values pass

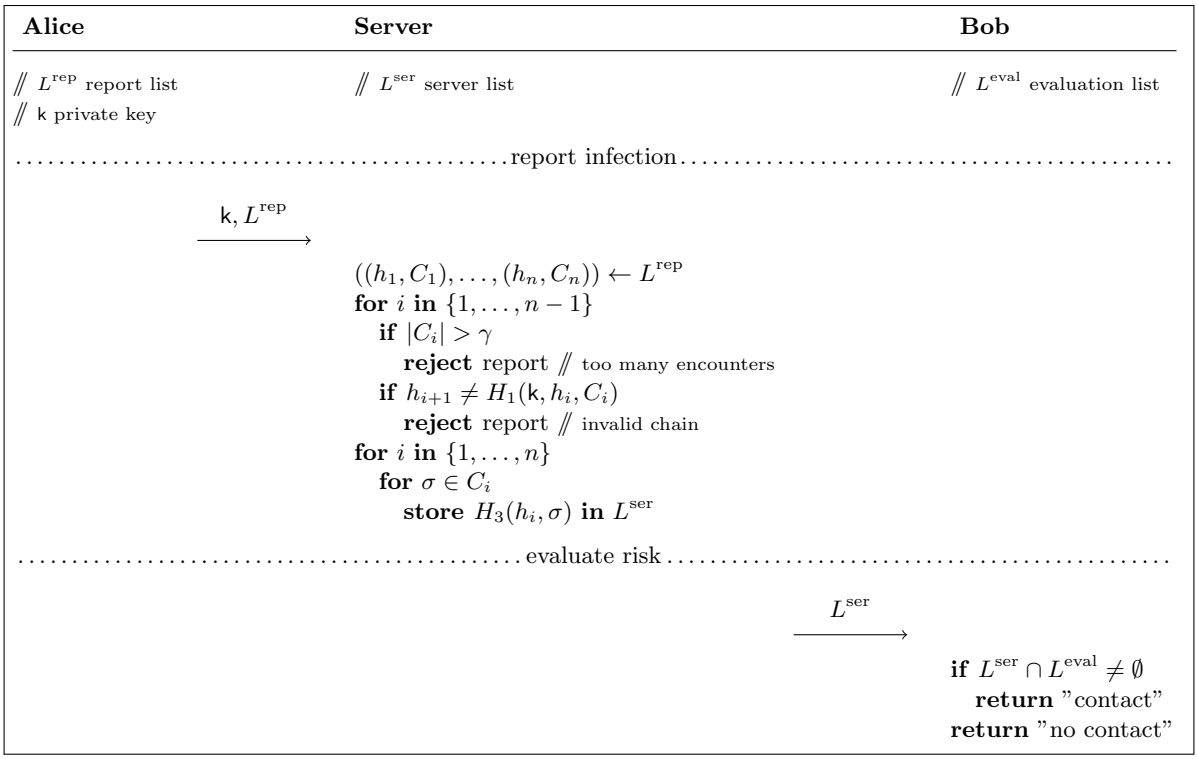

Figure 6: Report/risk-evaluation phase of Protocol 1

this check the server updates its list L^{ser} as follows. For every set C_i it adds the hash value $H_3(h_i, \sigma)$ to L^{ser} for all $\sigma \in C_i$.

- (evaluate infection risk) After downloading L^{ser} from the server a user Bob will check whether L^{ser} contains any of the pairs stored in L^{eval} (of the last two weeks say, older entries are deleted). If this is the case he assumes that he was in proximity to another infected user.

Epochs. As the hash chain only progresses if the device received at least one message during an epoch, we can choose fairly short epochs, say $\tau = 60$ seconds, without letting the chain grow by too much, but it shouldn't be too short so that we have a successful encounter (i.e., one message in each direction) of close devices within each sufficiently overlapping epochs with good probability. Choosing a small τ also gives better security against replay attacks, which are only possible within an epoch. Another advantage of a smaller τ is that it makes tracing devices using passive recoding more difficult as the broadcasts in consecutive epochs cannot be linked (except retroactively by the server after a user reports). We also bound the maximum number of contacts per epoch to some γ. We do this as otherwise a Crown Affair is possible by simply never letting the attacking devices progress the hash chain. With this bound we can guarantee that in a valid chain all but at most γ of the encounters must have been received by the same device.

Correctness. Consider two parties A and B which meet, and where B receives (h_A, ρ_A) from A, and A receives (h_B, ρ_B) from B. Then (by construction) B stores $H_3(h_A, \sigma) \in L^{\text{eval}}$ where $\sigma = H_2(h_A, h_B, \rho_B)$, while A stores $(h_A, C) \in L^{\text{rep}}$ where $\sigma \in C$. Should A be diagnosed and upload L^{rep}, B will get a L^{ser} which contains $H_3(h_A, \sigma)$, and thus B will raise a contact alert as this value is in its L^{eval} list.

Privacy. We shortly discuss the privacy of users in various cases (user diagnosed or not, server privacy breached or not).

Non-diagnosed user. As discussed in §2.1, as we use a keyed hash function a device just broadcasts (pseudo)random and unlinkeable values. Thus as long as the user isn't diagnosed and agrees to upload its L^{ser} list, the device gets hacked or is seized, there's no serious privacy risk.

Diagnosed user. We now discuss what happens to a diagnosed user who agrees to upload its L^{ser} list.

- Server view: As the chaining values are just radomized hashes, from the server's perspective the lists L^{ser} uploaded by diagnosed users just look like random and independent hash chains. In particular, the server will not see which chains belong to users that had a contact. What the chains do leak, is the number of epochs with non-zero encounters, and the number of encounters in each epoch.
- Other user view: A user who gets the L^{ser} list from the server only learns the size of this list, and combined with its locally stored data this only leaks what it should: the size of the intersection of this list with his L^{eval} list, which is the number of exchanges with a devices of later diagnosed users.
- Joint Server and other user view: If the view of the server and the data on the device X is combined, one can additionally deduce where in an uploaded chain an encounters with X happened.

The above discussion assumes an honest but curious adversary,[5] once we consider active attacks, tracking devices, etc., privacy becomes a much more complex issue. Discussions on schemes similar to ours are in [Vau20b, ABIV20, des20]. We will not go into this discussion and rather focus on the main goal of our schemes, namely robustness against false positives.

Security. Protocol 1 does not succumb to replay attacks, and most importantly, not to a Crown Affair. In the next section we prove this, but formally only for the toy protocol. The proof for the toy protocol already conveys the idea of the proof, but is cleaner than a proof for Protocol 1. We then outline how the proof can be adapted to Protocol 1.

3 Security of Protocol 1

3.1 Security of the toy version of Protocol 1

We model an adversary as a 4-tuple of algorithms $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3)$ playing the game in Figure 7. The bob_i oracles consist of two oracles: 1) The first oracle takes a message m and invokes the receive broadcast message procedure on bob_i with m as the input. 2) The second oracle takes no input and invokes the broadcast procedure on bob_i and returns the resulting message m to the adversary.

Theorem 1 (Security of the toy protocol from §2.1 against Crown Affair)**.** *If H is modeled as a random oracle, any adversary $\mathcal{A}$ making at most q queries to $H : \{0,1\}^* \to \{0,1\}^w$ and having at most t interactions with the* bob*'s can win the $CA^{\mathcal{A}}$ game against the toy protocol with probability at most $\frac{t^2+tq}{|\mathcal{R}|} + \frac{tq(t+q)}{2^w}$*

Proof. Let $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3)$ be an adversary that wins the $\text{CA}^{\mathcal{A}}$ game with non-negligible advantage. Since bob_i both evaluate to "contact", we must have that for both of them $L^{\text{ser}} \cap L^{\text{eval}} \neq \emptyset$, i.e. L^{ser} contains the pairs (h_{A_1}, ρ_{B_1}) and (h_{A_2}, ρ_{B_2}), where during the game, bob_i received h_{A_i} and chose ρ_{B_i}. Note that with overwhelming probability we have $\rho_{B_1} \neq \rho_{B_2}$ for all ρ_{B_i} sent by bob_i, so the two pairs must be distinct. Then, since the server verifies the hash chain, we must have that τ_3 consists of a list L of pairs (h_i, ρ_i) such that $h_{i+1} = H(h_i, \rho_i)$ and $(h_{A_i}, \rho_{B_i}) \in L$. W.l.o.g. assume that (h_{A_1}, ρ_{B_1}) appears before (h_{A_2}, ρ_{B_2}) in L. Note that $\mathcal{A}_2$ outputs h_{A_2} without knowing ρ_{B_1}. So $\mathcal{A}_3$ needs to find inputs to a hash chain from some

[5] And some precautions we didn't explicitly mention, like the necessity of permute the L^{ser} list and let the devices store the L^{eval} list in a history independent datastructure.

$\mathrm{CA}^{\mathcal{A}=(\mathcal{A}_0,\mathcal{A}_1,\mathcal{A}_3,\mathcal{A}_4)}$

$\tau_1, \tau_2 \leftarrow\!\!\$\ \mathcal{A}_0$
for $i \in \{1,2\}$ **do**
  $\mathsf{bob}_i \leftarrow\!\!\$\ \text{setup}$
  $\tau_i' \leftarrow\!\!\$\ \mathcal{A}_i^{\mathsf{bob}_i \text{ oracles}}(\tau_i)$
$\tau_3 \leftarrow\!\!\$\ \mathcal{A}_3(\tau_1', \tau_2')$
send τ_3 to backend server
for $i \in \{1,2\}$ **do**
  bob_i evaluates risk
if both bobs evaluate to "contact"
  return 1
else
  return 0

Figure 7: Crown Affair Security Game (used for the toy Protocol)

(h_{A_1}, ρ_1) (for some (h_{A_1}, ρ_1) being in the set of exchanges of $\mathcal{A}_1$) that collides with one of the h_{A_2} sent by $\mathcal{A}_2$. If H is a random oracle, this is infeasible with polynomially many calls to H, as we show next.

Let Q_i be the queries of $\mathcal{A}_i$ to H and $q_i = |Q_i|$. Let T_i be the values h'_{A_i} sent by $\mathcal{A}_i$ (for $i \in \{0,1\}$) and $t_i = |T_i|$. Note that in the toy protocol t_i also corresponds to the number of messages received by the respective adversaries. Finally, let $h = H(h_{A_1}, \rho_{B_1})$. Since ρ_{B_1} is chosen by bob_1, we have $(h_{A_1}, \rho_{B_1}) \notin Q_0 \cup Q_2$ except with probability $(q_0 + q_2)/|\mathcal{R}|$. So except with this probability h looks uniformly random to $\mathcal{A}_0$ and $\mathcal{A}_2$, because H is modeled as a RO. Accordingly, h is independent of any of the queries in $Q_0 \cup Q_2$ and any of the values in T_2. So constructing a hash chain between h and any of the values in T_2 requires that the value of H under some query in $Q_1 \cup Q_3$ collides with one in T_2 or with (the first entry of) any of the queries in $Q_0 \cup Q_2$. The probability of this event is less than $(q_1 + q_3) \cdot \frac{t_2+q_0+q_2}{2^w}$, where w is the output length of H. So for a specific ρ_{B_1} the probability of it causing an alert for bob_1 is at most $\frac{q_0+q_2}{|\mathcal{R}|} + \frac{(q_1+q_3)(t_2+q_0+q_2)}{2^w}$. Finally, we get another multiplicative factor of t_1 by taking the union bound over all possible ρ_{B_1}. By setting $q = \sum_i q_i$ and $t = t_1 + t_2$ we get an upper bound of $\frac{tq}{|\mathcal{R}|} + \frac{tq(t+q)}{2^w}$. □

3.2 Security of Protocol 1

Protocol 1 as illustrated in Figure 5 is not secure for the security game in Figure 7. The two devices $\mathcal{A}_1, \mathcal{A}_2$ can simply use the same (h, ρ) in the first epoch, and later $\mathcal{A}_3$ will combine their respective lists $(h, C_1), (h, C_2)$ into one $(h, C = C_1 \cup C_2)$ and upload it. To win the game it's sufficient that the C_1, C_2 lists contain a single hash each, but this attack even works as long as they jointly do not contain more than γ elements.

This restricted attack is basically the only possible Crown Affair possible against Protocol 1. Consider a Crown Affair where the adversary $\mathcal{A}_0$ initiates a large number (not just 2) of devices $\mathcal{A}_1, \ldots, \mathcal{A}_m$ (that cannot communicate during the attack), and later combines their states into a hash chain $L^{\text{rep}} = (h_1, C_1), (h_2, C_2), \ldots, (h_n, C_n)$ to upload. Assume this upload later alerts users $\mathsf{bob}_1, \mathsf{bob}_2, \ldots, \mathsf{bob}_t$ because they had an encounter with one of the $\mathcal{A}_i$'s. Then, with overwhelming probability one of two cases holds:

1. There was no Crown Affair, that is, all alerted bob's encountered the same $\mathcal{A}_i$.
2. All the encounters of the bob's are recorded in the same C_i (and thus the $C_j, j \neq i$ must contain values that cannot trigger an alert).

While the 2nd point means a Crown Affair is possible, it must be restricted to an upload which in total can only contain γ values that will actually raise an alert. We will not give a formal proof here, just the intuition

which is similar to the proof of Theorem 1:

Assume neither of the conditions 1. or 2. above holds, there there exist $(h_i, C_i), (h_{i'}, C_{i'}), \sigma \in C_i, \sigma' \in C_{i'}, i' > i$ s.t. (h_i, σ) and $(h_{i'}, \sigma')$ will raise alerts with some bob_k and $\mathsf{bob}_{k'}$ which were encountered by different $\mathcal{A}_j$ and $\mathcal{A}_{j'}$. But this means, $\mathcal{A}_{j'}$ must have decided (when sending a message to $\mathsf{bob}_{j'}$) on a value $h_{i'}$, such that later the adversary managed to create a hash chain L^{rep} where the output at some depth i' is $h_{i'}$, while the input in some earlier block $i < i'$ contains σ. Extending the argument used in the proof of Theorem 1, we can prove that such an attack is exponentially hard if hashing is done by a random oracle.

4 Protocol 2: Decentralized, Using Location for Chaining

In this section we describe our second protocol, which requires that the devices have access to some location based coordinate. This coordinate is infused into the hash-chain so chains of different devices (at different coordinates) will diverge, and thus prevent a Crown Affair. Possible coordinates are coarse grained GPS location, cell tower IDs or information from IP addresses.

The protocol progresses in epochs (say of $\tau = 60$ seconds), where at the beginning of an epoch the device samples randomness ρ and its coordinate ℓ. It then broadcasts ρ together with the head h of its hash chain. If during an epoch at least one message was received, the hash chain is extended by hashing the current head with a commitment of the location ℓ using randomness ρ.

4.1 Protocol Description

The protocol makes use of collision resistant hash functions H_1 to progress the chain, and a hash function H_2 which is basically used as a commitment scheme (and we use notation $H_2(m; \rho)$ to denote it's used as commitment for message m using randomness ρ), but we need H_2 to be hiding even if the same randomness is used for many messages. For this it's sufficient that H_2 is collision resistant (for binding), and for a random ρ, $H_2(\rho, \cdot)$ is a PRF with key ρ.

A formal description of Protocol 2 is given below. The broadcast/receive phase is additionally depicted in Figure 8, and its report/evaluate phase in Figure 9.

- (setup) Users sample a key k and a genesis hash value h_1, and set the current head of the hash chain to $h \leftarrow h_1$. Then they initialize empty lists L^{rep} and L^{eval} which are used to store information to be reported to the backend server in case of infection or used to evaluate whether contact with an infected person occurred, respectively.

- (epoch starts) An epoch starts every τ seconds, and the epoch number t is the number of epochs since some globally fixed timepoint (say Jan. 1st 2020, 12am CEST). At the beginning of every epoch the device samples a random string $\rho \leftarrow_{\$} \mathcal{R}$ and it retrieves its current coordinate $\ell \leftarrow \mathsf{get\ Coordinate}$.

- (broadcast) During the epoch the device regularly broadcasts the head h of its current chain together with ρ.

- (receive broadcast message) Whenever the device receives a message (h_B, ρ_B) it computes a commitment of the current coordinate and time (i.e., epoch number t) using the received randomness $\sigma_B \leftarrow H_2(\ell, t; \rho_B)$, and stores the tuple (h_B, σ_B) in L^{eval}.

- (epoch ends) If at the end of the epoch there was at least one message receive during this epoch ($\mathsf{contact} = 1$), the device computes a commitment $\sigma \leftarrow H_2(\ell, t; \rho)$ of its coordinate and time using randomness ρ, it stores this σ and the head h of the chain in the list L^{rep} (of values to be reported in case of being diagnosed), and progresses its hash chain as $h \leftarrow H_1(\mathsf{k}, h, \sigma)$.

- (report message to backend server) If diagnosed a users can upload their key k and the list $L^{\text{rep}} = ((h_1, \sigma_1), \ldots, (h_n, \sigma_n))$ to the server. The server verifies that the uploaded values indeed form a hash chain, i.e., that $h_{i+1} = H_1(\mathsf{k}, h_i, \sigma_i)$ for all $i \in \{1, \ldots, n-1\}$. If the uploaded values pass this check the server updates its list L^{ser} by adding L^{rep} to it.

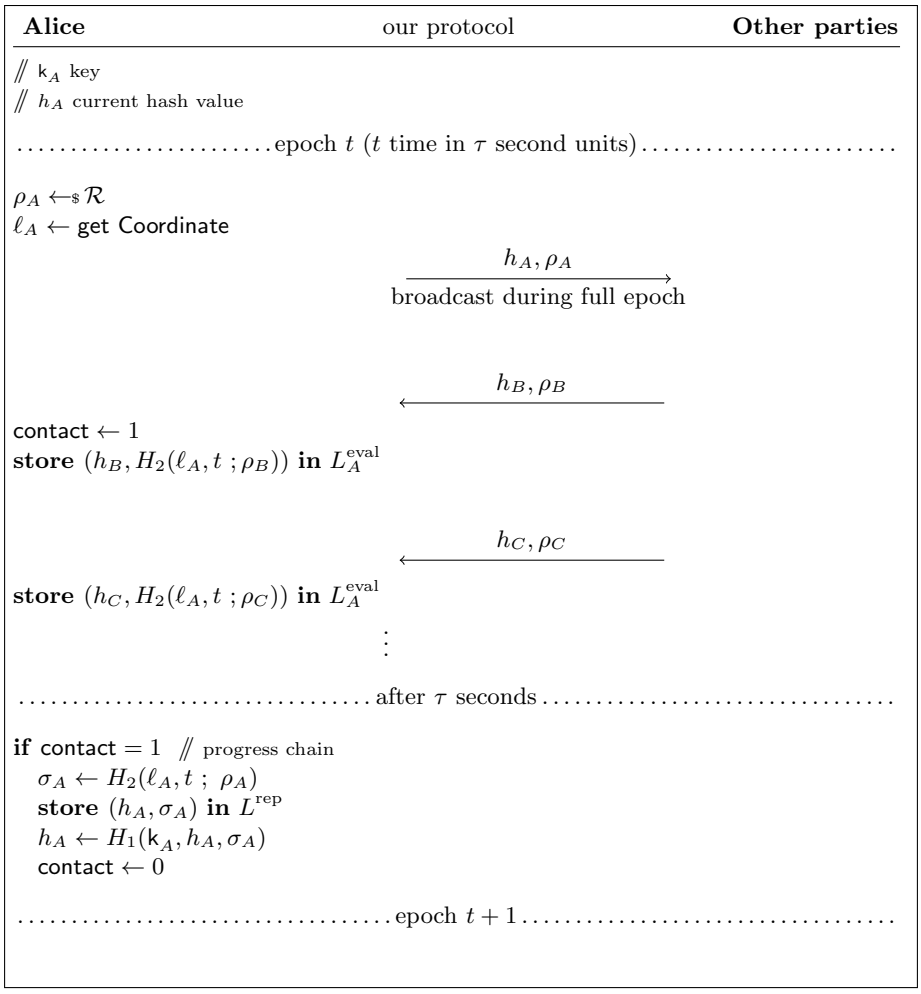

Figure 8: Broadcast/receive phase of Protocol 2

- (evaluate risk) After downloading L^{ser} from the server users check whether L^{ser} contains any of the pairs stored in L^{eval}. If this is the case they assume that they were in contact with an infected party. As the user learns the size of the intersection, a more sophisticated risk evaluation is also possible.

4.2 Correctness, Privacy and Epochs

Correctness. Consider two devices A and B which measured locations ℓ_A and ℓ_B and are in epochs t_A and t_B, and where A receives (h_B, ρ_B) from B and thus stores $\sigma = (h_B, H_2(\ell_A, t_A; \rho_B))$ in L_A^{eval}. Assume B receives at least one message during this epoch, then it will store $\sigma' = (h_B, H_2(\ell_B, t_B; \rho_B)$ in its L_B^{rep} list.

If later B is diagnosed it uploads its L_B^{rep} list to the server. At its next risk evaluation A will receive L^{ser} (which now contains L_B^{rep}) from the server. It will report a contact if $L^{\text{ser}} \cap L_A^{\text{eval}} \neq \emptyset$ which holds if $\sigma' = \sigma$ or equivalently

$$(h_B, H_2(\ell_B, t_B; \rho_B) = (h_B, H_2(\ell_A, t_A; \rho_B)) \quad \text{which is implied by} \quad (\ell_A, t_A) = (\ell_B, t_B)$$

Summing up, in the setting above A will correctly report a contact if

1. A and B are synchronised, i.e., in the same epoch $t_A = t_B$.

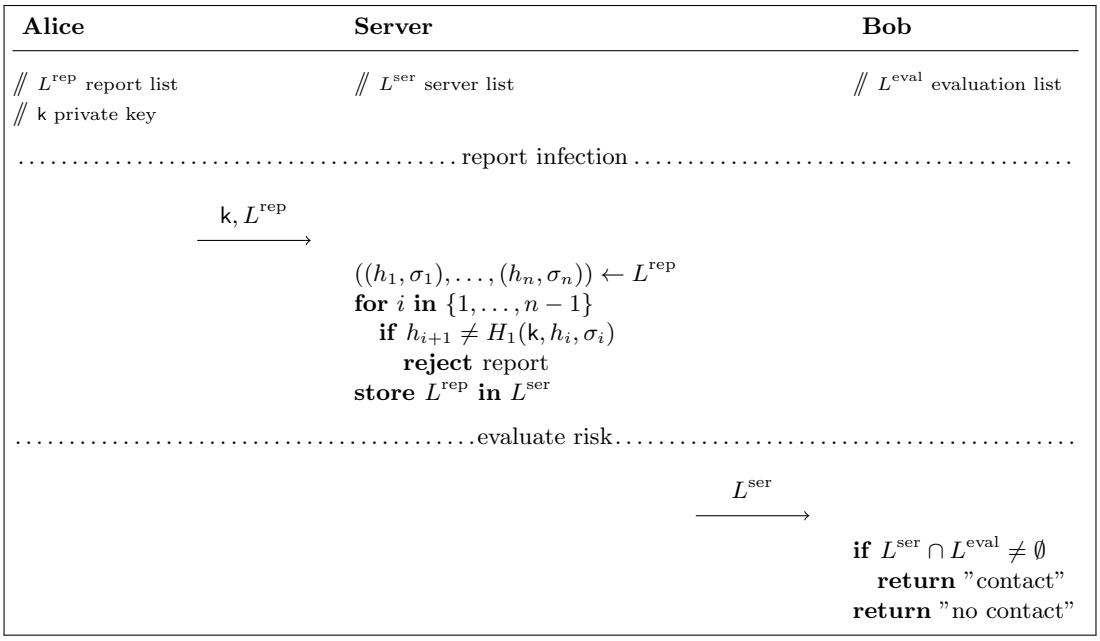

Figure 9: Report/risk-evaluation phase of Protocol 2

2. B received at least one message during epoch t_B.
3. A and B were at the same locations (i.e., $\ell_A = \ell_B$) at the beginning of the epoch.

Condition 2. should be satisfied in most cases simply because the fact that A received a message from B means B should also have received a message from A. This condition only exists because we let the devices progress their chains only in epochs where encounters happened.[6]

Epochs. As epochs are synchronized, even if the coordinates of the devices change frequently because the devices are moving, two devices will still have the same coordinate as long as they were at the same coordinate at the beginning of an epoch, think of two passengers in a moving train. But we can have a mismatch (and thus false negative) if two devices meet that were at different coordinates at the beginning of an epoch, e.g., two people meet at a train station, where at the beginning of the epoch, one person was in the moving train, while the other was waiting at the platform. To address this problem one should keep the epochs sufficiently short, in particular, much shorter than the exposure time that would raise an alert.

Privacy. Our Protocol 2 is similar to the unlinkeable variant of DP-3T, and thus has similar privacy properties. In particular, the only thing non-diagnosed users broadcast are pseudorandom and unlinkeable beacons. But there are two privacy issues that arise in our protocol which DP-3T does not have. The first is because we use chaining, the second because we use coordinates:

1. (Sever can link) Even though the beacons broadcasted by a diagnosed user are not linkeable by other users (assuming the server permutes the L^{ser} list before other users can download it), the server itself can link the beacons (it gets them in order and also the hash key to verify this). So compared to DP-3T we put more trust in the server concerning this privacy aspect.

[6]The reason for only progressing if there was an encounter is that this way the chain is shorter (thus there's less to up and download), the chain reveals less information (i.e., even the server can't tell where the empty epochs were) and tracing using passive recording devices becomes more difficult.

2. (Digital evidence) When discussing privacy, one mostly focuses on what information can be learned about a user. But there's a difference between learning something, and being able to convince others that this information is legit. While in decentralized protocols like DP-3T a malicious device can easily learn when and where an encounter with a later diagnosed user happened by simply storing the recorded beacons together with the time and location, it's not clear how the device would produce convincing evidence linking the uploaded beacon with this time and location

   In a protocol that uses time and location, like our Protocol 2, one can produce such evidence by basically time-stamping the entire transcript of an encounter (e.g. by posting a hash of it on a blockchain), and later, when a user is diagnosed and its encounter tokes become public, use this time-stamped data as evidence of the encounter. This problem already arises when one uses time to prevent replay attacks, and location to prevent relay attacks as discussed in [Pie20] for details.

5 Security of Protocol 2

5.1 Security against replay and relay attacks

The protocol is secure against replay and relay attacks in the following sense: Assume Alice is at location ℓ_A and epoch t_A, broadcasts (h_A, ρ_A) and thus stores $\sigma_A = (h_A, H_2(\ell_A, t_A \ ; \ \rho_A) \in L^{\text{rep}}$. Now assume an adversary replays the message with potentially changed randomness (h_A, ρ'_A) to user Bob who is at a different location and/or epoch $(\ell_B, t_B) \neq (\ell_A, t_A)$ than Alice was. Bob then stores $\sigma_B = (\ell_B, t_B \ ; \ \rho'_A) \in L^{\text{eval}}$. Should Alice later upload her L^{rep} list, Bob will trigger a contact warning if $\sigma_A = \sigma_B$, i.e.,

$$H_2(\ell_A, t_A \ ; \ \rho_A) = H_2(\ell_B, t_B \ ; \ \rho'_A)$$

That is, he must break the binding property of the commitment scheme.

5.2 Security against a Crown Affair

5.2.1 A Weaker Security Model

Protocol 2 achieves weaker security against the Crown Affair since there is less interaction: the randomness chosen by the users in protocol 1 is replaced by location to defend somewhat against Crown Affair. Accordingly, we need to weaken the model in order to prove security. In particular, we cannot let the adversary have control of the locations, otherwise it can trivially carry out an attack. So we assume that the locations of the encounters with the bob's follow some unpredictable distributions $\mathcal{P}_i$.

5.2.2 Security Proof

Theorem 2. *If H_1 is modeled as a random oracle, H_2 is ϵ-collision-resistant, and $\mathcal{P}_1, \mathcal{P}_2$ are independent and have min-entropy at least k, then any adversary $\mathcal{A}$ making at most q queries to $H_1 : \{0,1\}^* \to \{0,1\}^w$ and having at most t interactions with the* bob *'s can win the weak-CA$^{\mathcal{A}}_{\mathcal{P}_1,\mathcal{P}_2}$ game against protocol 2 with probability at most $2^{-k} + \frac{q}{|\mathcal{R}|} + \frac{2q^2}{2^w} + \epsilon$.*

Proof. Let $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3)$ be an adversary that wins the CA$^{\mathcal{A}}$ game with non-negligible advantage. We assume that the first samples from $\mathcal{P}_1$ and $\mathcal{P}_2$ are different, which happens with probability at least $1 - 2^{-k}$. Furthermore, since both bob's evaluate to "contact", we must have that for both of them $L^{\text{ser}} \cap L^{\text{eval}} \neq \emptyset$, i.e. L^{ser} contains pairs (h_{A_1}, σ_{A_1}) and (h_{A_2}, σ_{A_2}) such that during the game bob_j received h_{A_j} and ρ_{A_j} at coordinate ℓ_{A_j} and it holds $\sigma_{A_j} = H_2(\ell_{A_j}; \rho_{A_j})$, where $j \in \{1, 2\}$. Then, since the server verifies the hash chain, we must have that τ_3 consists of a key k and a list L of pairs (h_i, σ_i) such that $h_{i+1} = H_1(\mathsf{k}, h_i, \sigma_i)$ and $(h_{A_1}, \sigma_{A_1}), (h_{A_2}, \sigma_{A_2}) \in L$.

weak-CA$^{\mathcal{A}=(\mathcal{A}_0,\mathcal{A}_1,\mathcal{A}_2,\mathcal{A}_3)}_{\mathcal{P}_1,\mathcal{P}_2}$

$\tau_1, \tau_2 \leftarrow\!\!\$\ \mathcal{A}_0$
for $i \in \{1,2\}$ **do**
  $\mathsf{bob}_i \leftarrow\!\!\$$ setup
  $\tau_i' \leftarrow\!\!\$\ \mathcal{A}_i^{\mathsf{bob}_i \text{ oracles}}(\tau_i)$
$\tau_3 \leftarrow\!\!\$\ \mathcal{A}_3(\tau_1', \tau_2')$
send τ_3 to backend server
for $i \in \{1,2\}$ **do**
  bob_i evaluates risk
if both bobs evaluate to "contact"
  return 1
else
  return 0

bob_i receive(h_A, ρ_A)

$\ell \in\ \leftarrow\!\!\$\ \mathcal{P}_i$
run bob_i receive procedure with coordinate ℓ
return ℓ

bob_i broadcast

$\ell \in\ \leftarrow\!\!\$\ \mathcal{P}_i$
run bob_i broadcast procedure with coordinate ℓ
return the result and ℓ

Figure 10: Weak Crown Affair Security Game

We consider two cases: First, assume case 1) $(h_{A_1}, \sigma_{A_1}) = (h_{A_2}, \sigma_{A_2})$. Since $h_{A_1} = h_{A_2}$, either exactly the same sequence of coordinates were infused into the hash chain to obtain h_{A_1} and h_{A_2}, or $\mathcal{A}$ found a collision for H_1 or H_2. Furthermore, since $\sigma_{A_1} = \sigma_{A_2}$ either the coordinates where $\mathcal{A}_i$'s meet bob_i's coincide as well, or $\mathcal{A}$ found a collision for H_2. Thus, either the location histories of $\mathcal{A}_1$ and $\mathcal{A}_2$ coincide, in which case this is not a valid attack, or $\mathcal{A}$ found a collision for H_1 or H_2, which happens with probability at most $\epsilon + q^2/2^w$.

Now, let's assume case 2) $(h_{A_1}, \sigma_{A_1}) \neq (h_{A_2}, \sigma_{A_2})$. W.l.o.g. assume that (h_{A_1}, σ_{A_1}) appears before (h_{A_2}, σ_{A_2}) in L. Note that $\mathcal{A}_2$ outputs h_{A_2} without knowing (h_{A_1}, σ_{A_1}) and $\mathcal{P}_1$ has high entropy. So $\mathcal{A}_3$ needs to find inputs to a hash chain from (h_{A_1}, σ_{A_1}) that collides with h_{A_2}. Similar to the proof of protocol 1, let Q_i be the queries of $\mathcal{A}_i$ to H_1 and $q_i = |Q_i|$. Furthermore, let $h = H_1(\mathsf{k}, h_{A_1}, \sigma_{A_1})$. Since (h_{A_1}, σ_{A_1}) is not known to $\mathcal{A}_2$, we have $(h_{A_1}, \sigma_{A_1}) \notin Q_0 \cup Q_2$ except with probability $(q_0 + q_2)/|\mathcal{R}|$. So except with this probability h looks uniformly random to $\mathcal{A}_0$ and $\mathcal{A}_2$, because H_1 is modeled as a RO. Accordingly, h is independent of any of the queries in $Q_0 \cup Q_2$. So constructing a hash chain between h and any of the values in $Q_0 \cup Q_2$ requires that the value of H_1 under some query in $Q_1 \cup Q_3$ collides with (the first entry of) any of the queries in $Q_0 \cup Q_2$. The probability of this event is less than $(q_1 + q_3) \cdot \frac{q_0+q_2}{2^w}$. Thus, in case 2) the probability of τ_3 causing an alert for bob_1 and bob_2 is at most $\frac{q_0+q_2}{|\mathcal{R}|} + \frac{(q_1+q_3)(q_0+q_2)}{2^w}$. By setting $q = \sum_i q_i$ and combining the two cases, we get an upper bound of $\frac{q}{|\mathcal{R}|} + \frac{2q^2}{2^w} + \epsilon$. □

References

[ABIV20] Gennaro Avitabile, Vincenzo Botta, Vincenzo Iovino, and Ivan Visconti. Towards defeating mass surveillance and sars-cov-2: The pronto-c2 fully decentralized automatic contact tracing system. Cryptology ePrint Archive, Report 2020/493, 2020. `https://eprint.iacr.org/2020/493`.

[app20] Privacy-preserving contact tracing. `https://www.apple.com/covid19/contacttracing`, 2020.

[CGH+20] Justin Chan, Shyam Gollakota, Eric Horvitz, Joseph Jaeger, Sham M. Kakade, Tadayoshi Kohno, John Langford, Jonathan Larson, Sudheesh Singanamalla, Jacob E. Sunshine, and Stefano Tessaro. PACT: privacy sensitive protocols and mechanisms for mobile contact tracing. *CoRR*, abs/2004.03544, 2020.

[CTV20] Ran Canetti, Ari Trachtenberg, and Mayank Varia. Anonymous collocation discovery: Taming the coronavirus while preserving privacy. *CoRR*, abs/2003.13670, 2020.

[CW20] Covid watch. `https://www.covid-watch.org/`, 2020.

[des20] Desire: A third way for a european exposure notification system. https://github.com/3rd-ways-for-EU-exposure-notification/project-DESIRE, 2020.

[ePA20] Pact: Private automated contact tracing. `https://pact.mit.edu/`, 2020.

[Gvi20] Yaron Gvili. Security analysis of the covid-19 contact tracing specifications by apple inc. and google inc. Cryptology ePrint Archive, Report 2020/428, 2020. `https://eprint.iacr.org/2020/428`.

[Pep20] Pepp-pt: Pan-european privacy-preserving proximity tracing. `https://www.pepp-pt.org/`, 2020.

[Pie20] Krzysztof Pietrzak. Delayed authentication: Preventing replay and relay attacks in private contact tracing. Cryptology ePrint Archive, Report 2020/418, 2020. `https://eprint.iacr.org/2020/418`.

[Rob20] Robert: Robust and privacypreserving proximity tracing. `https://github.com/ROBERT-proximity-tracing`, 2020.

[TPH+20] Carmela Troncoso, Mathias Payer, Jean-Pierre Hubaux, Marcel Salath, James Larus, Edouard Bugnion, Wouter Lueks, Theresa Stadler, Apostolos Pyrgelis, Daniele Antonioli, Ludovic Barman, Sylvain Chatel, Kenneth Paterson, Srdjan Capkun, David Basin, Dennis Jackson, Bart Preneel, Nigel Smart, Dave Singelee, Aysajan Abidin, Seda Guerses, Michael Veale, Cas Cremers, Reuben Binns, and Thomas Wiegand. Dp3t: Decentralized privacy-preserving proximity tracing, 2020. `https://github.com/DP-3T`.

[Vau20a] Serge Vaudenay. Analysis of dp3t. Cryptology ePrint Archive, Report 2020/399, 2020. `https://eprint.iacr.org/2020/399`.

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

A Protocol 1 with efficient risk-evaluation

More efficient contact evaluation. Users evaluating their risk status in Protocol 1 have to download a list L^{ser} of size linear in the number of contacts of all infected users. We now describe a variant of the protocol with download size depending only on the number of epochs with non-zero encounters but otherwise independent of the number of encounters. The variant makes use of the fact that, unless a Crown Affair occurred, already storing one element of the hash chain of an infected user is enough to indicate that contact with this user occurred. The variant's broadcast/receive phase is as in Figure 5, except that we don't apply H_3 (think of H_3 as the identity function), so the incoming tuples (h, σ) are stored in the clear. The report/risk-evaluation of this variant is shown in Figure 11.

Regarding reports of infections the server now prepares two lists L_1^{ser} and L_2^{ser}. The former contains all elements h_i of valid hash chains, the latter all tuples of the form (h_i, σ) with sigma ranging over all elements of the randomness pool C_i corresponding to h_i. Users evaluate their risk status in two steps. After downloading L_1^{ser} from the server they first check whether any of the elements of L_1^{ser} is contained in a tuple stored in L^{eval}. In this case they either were in contact with a diagnosed person or potentially some attack like a Crown Affair is going on. In an optional second step users can exclude the latter possibility. To this end they send the tuple (h, σ) that triggered the alert to the server which in turn verifies whether it is contained in L_2^{ser}. If this is not the case the user has fallen victim to a Crown Affair.

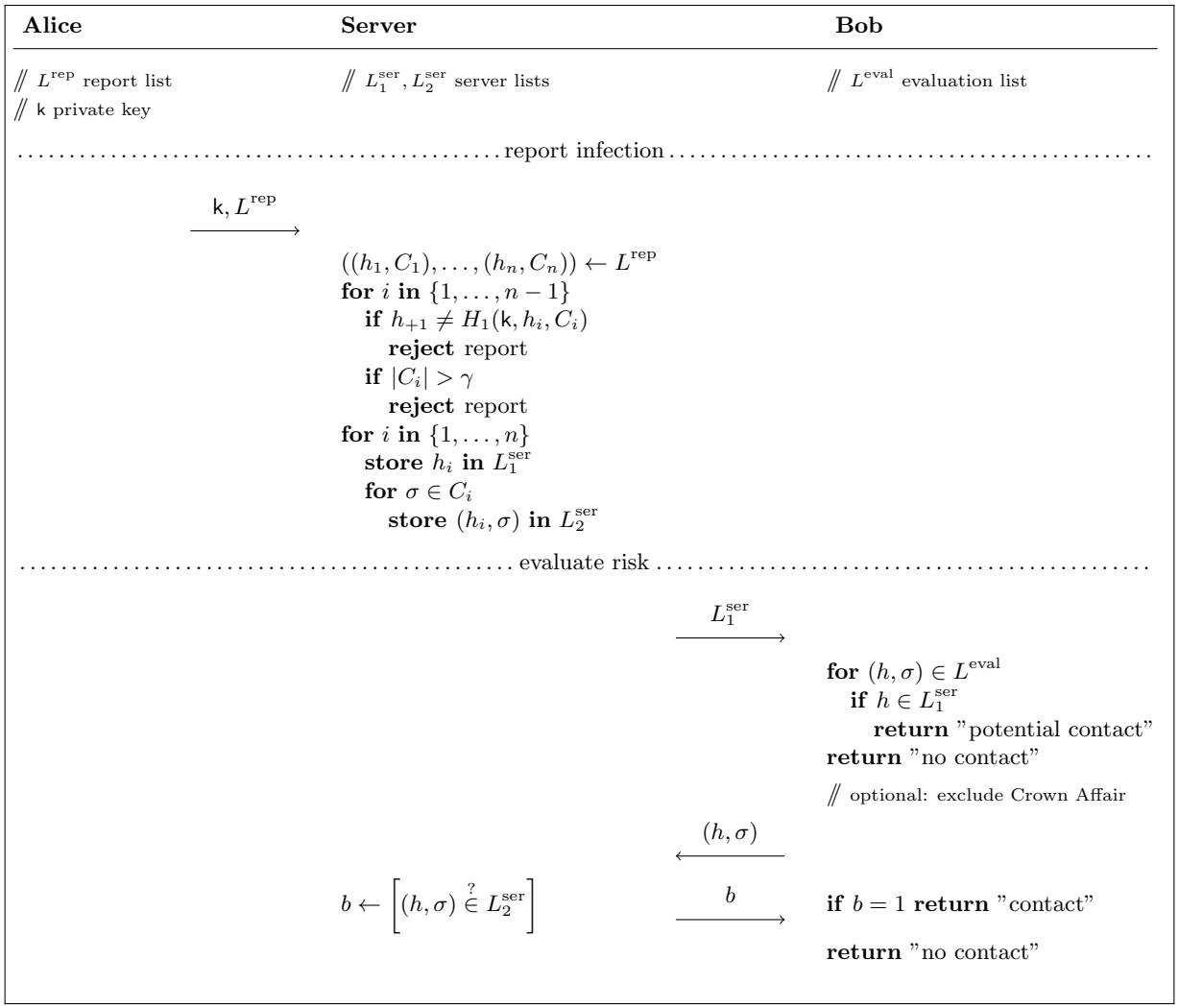

Figure 11: Alternative report/risk-evaluation phase for Protocol 1 which is more efficient but has weaker privacy

Regarding this variant's efficiency, note that the head of a user's hash chain is updated only once per epoch. Thus, L_1^{ser}'s size corresponds to the number of infected users times the number of epochs.

We stress that the modified protocol introduces additional privacy issues. Note that users which did not have contact with infected individuals will and assuming no attack is going on, will never execute the second phase of the risk evaluation protocol. Thus to prevent the server from learning this information it would be necessary to make the download of L_1^{ser} and the second phase of the risk evaluation unlinkable and protect the user's identity in the second phase of the protocol. A second concern is that the server, as it is given access to (h, σ), even learns with whom and at which position in the chain the contact occurred. To prevent this one could implement the second part of the risk evaluation with a private-set-intersection protocol.