SPiRiT: Verifiable Contact Tracing for Balanced Security, Privacy, and Safety

Reyhaneh Rabaninejad[1] ✉, Azra Abtahi[2,3], Antonis Michalas[1], and Amir Aminifar[3]

[1] Tampere University, Tampere, Finland
rabaninejad@gmail.com, antonios.michalas@tuni.fi
[2] Malmö University, Malmö, Sweden
[3] Lund University, Lund, Sweden

Abstract. Digital contact tracing systems play a pivotal role in mitigating the spread of infectious diseases during pandemics by identifying individuals exposed to confirmed cases. In decentralized privacy-preserving contact tracing protocols, users broadcast pseudo-random IDs and, upon infection, upload the relevant IDs to a cloud server. Other users can then download these IDs to determine if they have been in proximity to infected individuals. However, existing protocols often lack verifiability, implicitly assuming honest user behavior when uploading data to the server. This gap creates vulnerabilities, including false positives and false negatives, which hinder the widespread adoption crucial for the effectiveness of contact tracing apps. This paper introduces SPiRiT, a secure and privacy-preserving verifiable contact tracing protocol, designed to address these shortcomings by introducing verifiability, ensuring the legitimacy of uploaded IDs, while still maintaining user privacy. Verifiability ensures whether an upload contains a valid set of pseudo-random IDs with respect to the contagious period of the infected individual, and that the individual claiming those IDs is the legitimate owner, in a privacy-preserving manner. The goal of our design is to maintain a balance between the competing objectives of privacy, security, and decentralization while fostering greater public trust and adoption by addressing the aforementioned vulnerabilities. Through rigorous evaluation using a face-to-face individual interaction dataset, we demonstrate that SPiRiT not only reduces false alarms but also can be on par with or even outperform existing protocols in detecting high-risk contacts.

Keywords: Digital Contact Tracing, False Alarm, Pandemic Risk Mitigation, Privacy, Security, Verifiability.

1 Introduction

Digital contact tracing protocols serve the fundamental purpose of alerting individuals who have spent a significant time in close physical proximity to a confirmed case of a fast-spreading, highly-contagious pathogen. Notifications of this type alert us for potential exposure to the virus, prompting precautionary

actions, such as testing or quarantine, as recommended by public health authorities. Confidentiality regarding the diagnosed individual's identity and the specifics of when and where the contact occurred is crucial in this process. In our smartphone-dominated society, individuals carry these devices throughout the day, presenting an opportunity for a mobile application to collect data on close physical interactions. This application could play a crucial role in tracing contacts potentially infected through the spread period of the pathogen. The app keeps track of contacts when personal smartphones are in close physical proximity for a predefined duration and distance. Utilizing proximity tracing, the app identifies individuals who should be notified of a potential high-risk exposure without disclosing specific details about the diagnosed person or the location of the contact [10].

A substantial body of research has explored proximity-based contact tracing, producing a diverse range of protocol designs and analysis (e.g. [14, 9, 6, 35, 7, 25, 32]). A cutting-edge protocol for contact tracing is the Decentralized Privacy-Preserving Proximity Tracing (DP-3T) [36, 37], which also heavily influenced the Apple/Google-API (GAEN) developed for COVID-19 contact tracing [5]. In the DP-3T protocol, a user's digital device continuously transmits ephemeral pseudo-random IDs using Bluetooth Low Energy (BLE) while concurrently recording the pseudo-random IDs received from other users' devices in close proximity. When a user gets infected, they can disclose the pseudo-random IDs previously broadcasted by their device during the contagious period by uploading them to a central server. All user applications periodically download these pseudo-random IDs from the cloud. Subsequently, they cross-check these IDs against locally recorded IDs to identify high-risk contacts. Users with high-risk contacts receive an alert from the application, advising them to enter quarantine and undergo testing, facilitating prompt preventive measures.

Recent work [3] first introduces A/P DP-3T, an extended version of the DP-3T protocol. This protocol empowers users to control their privacy by enabling participation as either active or passive users, enhancing privacy and utility compared to previous work [36]. Specifically, passive users, embodying a more conservative approach, choose not to broadcast BLE beacons. Meanwhile, active participants continue to broadcast BLE beacons with pseudo-random IDs (employing the same mechanism as DP-3T). Passive participants do not broadcast BLE beacons but actively listen for broadcasted BLE beacons. Consequently, passive participants become aware of their potential exposure and can take appropriate action. Then, [3] introduces DP-ACT, where passive participants who become infected disclose their list of high-risk contacts based on the *received* pseudo-random IDs. Using this information, active participants can undergo home quarantine if their pseudo-random IDs are reported by passive participants. The authors emphasize that it is necessary to have at least one active participant in each contact group (defined as groups of users in close proximity for a considerable duration) in order to identify all high-risk contact groups with at least one infected participant.

However, there are some security risks associated with DP-ACT as also discussed in their paper [3]: *(I)* Malicious passive users possess the ability to actively manipulate the protocol by uploading IDs broadcasted by other active users, potentially leading to *false positive* outcomes. This vulnerability arises because passive users are granted the right to claim another's pseudo-random ID upon infection. This feature may be maliciously exploited to generate false positives. *(II)* An attacker, such as a malicious active user, has the capability to execute *flooding* and *replay* attacks by generating a substantial number of contact events, potentially overwhelming a user's phone memory, creating a vulnerability to trigger *false alerts*. Notably, this attack is also viable in DP-3T protocol [36, 18]. It is crucial to recognize that these identified security shortcomings can also impact the system's utility. Users may find it challenging to trust the protocol's alerts, ultimately impeding widespread adoption by the public [4, 13, 30, 27].

We argue that the root cause of false alert attacks lies in the inability to *verify* both *(I)* the legitimacy of an individual claiming a set of pseudo-random IDs, and *(II)* the validity of those IDs with respect to the contagious period. While some existing works partially address aspect *(I)*, as discussed below, there is– to the best of our knowledge– a surprising lack of solutions that fully address verifiability, encompassing both *(I)* and *(II)*.

Authorization in Prior Systems. Existing systems such as DP-3T [36] and its derivatives implement basic user authorization schemes to prevent arbitrary uploads. Typically, users are issued authorization codes alongside positive COVID-19 test results, which must be presented to upload their broadcasted pseudo-random IDs. While this ensures that only diagnosed users can upload data, it has two major limitations: *(a)* There is no cryptographic binding between the uploaded pseudo-random IDs and the persistent identities of the users. As a result, while authorized DP-3T [36] provides *user authorization*, it does not *verify the trace information* itself, leaving the system vulnerable to false alerts. *(b)* Moreover, such authorization mechanisms unfortunately introduce privacy risks by creating identifiable links between the meant-to-be-anonymous database of pseudo-random IDs and unique identities in public health registries. The approach proposed by Silde and Strand [34] addresses the latter issue and takes a step toward more private contact tracing by employing a direct extension of Privacy Pass tokens. In their design, the verification and backend services hold a master secret key k (in a designated verifier setting) and derive daily keys using a key derivation function: $\mathsf{KDF}(k, \mathsf{date})$. The corresponding public keys are published by the verification service. However, a fundamental limitation remains: there is no secret key associated with the users that can cryptographically bind the broadcasted pseudo-random IDs to the users. Consequently, issue *(a)* remains unresolved. Furthermore, the interactive nature of the design (user are reqiuired to obtain fresh keys and tokens from the verification service on a daily basis), the designated verifier setting, and the lack of support for threshold issuance, collectively lead to a more centralized architecture. The protocol also does not support passive users.

Given these observations in prior work, we ask the following question: "*Is it possible to develop an efficient contact tracing protocol that enables the verification of uploads, all while preserving user privacy?*"

Before presenting our contributions, it is crucial to clarify that in this paper, we solely focus on advocating for a cryptographic solution to the verifiable digital contact tracing problem, while electromagnetic side channels or BLE tracking mechanisms [23] are orthogonal to this work. Notably, there exist parallel advancements in the practical obfuscation of BLE tracking mechanisms [24], a topic beyond the scope of this paper.

1.1 Contributions

In this work, we aim to incorporate verifiability into digital contact tracing and propose a practical, secure, and privacy-preserving protocol grounded in well-established cryptographic assumptions. Our ultimate goal is to foster widespread acceptance by effectively addressing the notable concerns outlined above. Following we present our contributions:

Definitions. We formalize SPiRiT (**S**ecure and **P**r**i**vacy-Preserving Ve**Ri**fiable Contact **T**racing) (Section 3.1), where users perform a one-time registration to obtain a unique token under a privacy-preserving distributed trust model. The token enables users to broadcast unlinkable yet verifiable pseudo-random IDs. When diagnosed, a user submits the pseudo-random IDs generated during the contagious period to the Public Health Authority (PHA) together with an efficient zero-knowledge proof showing *(I)* that the IDs correspond to the contagious period and *(II)* legitimate ownership of those IDs. User devices periodically query the PHA for published IDs and check for matches with stored observations. From a privacy perspective, all stages–registration, ID broadcasting, and trace disclosure–must remain unlinkable to the user's identity and across broadcasts (unlinkability). From a security perspective, the system must prevent adversaries from producing valid proofs for IDs not derived from a registered key (upload integrity) and from rebroadcasting previously observed IDs (replay security).

Efficient Construction. We put forth an efficient construction of SPiRiT that realizes our formal model under standard cryptographic assumptions (Section 4). For token issuance during registration, we leverage Anonymous Counting Tokens (ACTs) [8] in a threshold setting (tACT) [1] (Section 2), enabling anonymous user registration and deduplication. Each user obtains a unique token that is stored together with a commitment to the user's key after verification against redeemed tokens. During verification, the PHA checks a zero-knowledge proof on the uploaded trace information using the ACT token and the committed key, ensuring legitimate ownership while mitigating false alerts. Our design is compatible with A/P DP-3T, allowing users to operate in active or passive modes depending on their privacy and security preferences. The system preserves decentralization while enabling zero-knowledge verification, balancing privacy, security, and deployability, mitigating the aforementioned vulnerabilities.

Evaluation. We evaluate SPiRiT using the face-to-face interaction InVS15 dataset [22] (Section 5) under two scenarios. First, we consider attackers who maliciously

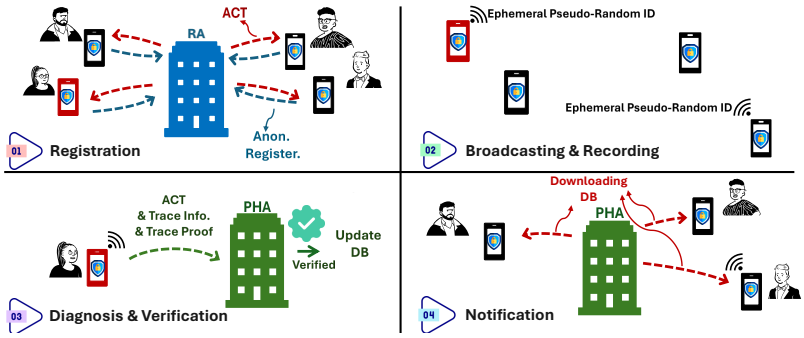


Fig. 1: SPiRiT system flow. ① Users partake in a one-time registration process to obtain a unique Anonymous Counting Token (ACT) from a committee of Registration Authorities (RAs) within a privacy-preserving and distributed trust framework. ② During the broadcast phase, active users generate ephemeral pseudo-random IDs tied to the user's registered key and the current epoch number, in an unlinkable and yet verifiable manner. ③ Upon a positive diagnosis, the infected active user generates trace information (pseudo-random IDs), along with a zero-knowledge proof which is used by Public Health Authority (PHA) to verify the validity of claimed IDs. The PHA runs the SPiRiT verification algorithm to confirm both the authenticity of the trace information with respect to the user's ACT and its validity with respect to the contagious period. These IDs are then stored in PHA database for efficient querying. If registered as passive, the user only listens for broadcasted IDs (Step 2) and does not upload trace data if infected (Step 3). ④ During the trace phase, user devices regularly check for updates in the server DB and alerts the user of any matches found within the locally stored database of pseudo-random IDs.

reveal IDs of non-infected users as those of infected individuals. SPiRiT prevents this attack and maintains a zero false-alarm rate, while existing protocols accumulate high false alarms over time. In addition, SPiRiT outperforms DP-3T and achieves performance comparable to A/P DP-3T in detecting high-risk contacts. Second, we study the case where diagnosed users withhold a fraction of their broadcasted IDs during the contagious period. While this behavior significantly degrades the performance of existing protocols, SPiRiT remains robust. Benchmark results from an unoptimized implementation show that user registration completes in approximately 2.6 seconds, and proof generation for trace verification takes about 173 ms on the slowest tested device. The implementation is publicly available at SPiRiT GitHub.

Figure 1 outlines a high-level architecture of the SPiRiT system, while Table 1 compares contact tracing protocols across key aspects.

1.2 Applications Beyond Contact Tracing

Our primary motivation stems from decentralized privacy-preserving contact tracing protocols such as DP-3T [36], and we believe that preparedness for future pandemics remains a critical global priority. However, our design approach extend to a broader set of applications where privacy, integrity, and decentralization are essential. These include:

– **Crowd density monitoring without tracking individuals' movements:** Users can anonymously broadcast ephemeral identifiers when present in a physical location (e.g., a café, public transport hub, or event venue), to contribute to aggregate crowd size estimations, enabling services like *Crowd Alerts*

Table 1: Comparison of contact tracing protocols across key aspects. $\mathbb{CP}$ and id_U denote contagious period and unique user identifier, respectively.

Aspect	**DP-3T (2020) [36]**	**Silde & Strand (2022) [34]**	**A/P DP-3T & DP-ACT (2024) [3]**	**This Work**
Verify User Legitimacy (I)	Full (authorization codes)	Full (tokens)	Not Addressed	Full (tokens)
Verify Uploads (II) w.r.t. $\mathbb{CP}$	Not Addressed	Not Addressed	Not Addressed	Full (cryptographic binding)
Verify Uploads (II) w.r.t. id_U	Not Addressed	Not Addressed	Not Addressed	Full (cryptographic binding)
Preserve User Privacy	Partial (linkability issue)	Partial (improved)	Full (no formal proof provided)	Full (strong unlinkability, formal definition and rigorous proof)
Support Passive Users	Not Addressed	Not Addressed	Full	Full
Prevent False Alarms	Not Addressed	Not Addressed	Not Addressed (vulnerable)	Effectively Reduced (verified uploads)

that inform users about how busy a place is *before* they head out—without revealing individual presence or movement patterns.

– **Secure check-in systems for events or shared spaces:** Powered by SPiRiT's verifiability feature, users can anonymously *prove* their presence in locations such as co-working spaces or libraries, enabling capacity management or access tracking without storing identifiable data.

2 Foundation of SPiRiT: Anonymous Counting Tokens

In the next lines, we revisit anonymous counting tokens (ACT) and review their security properties. ACT primitive recently introduced in [8] allows users to acquire tokens for private messages while enabling issuers or verifiers to enforce rate limits on the number of tokens obtained or redeemed per message. One of the basic challenges in ACT system design is balancing anonymity. This prevents mapping tokens to user identities, with the need to limit individual user contributions. Benhamouda et al. [8] propose two ACT constructions. The first uses PRF evaluation for token issuance with keyed verification. It is suitable when the issuer and verifier are the same entity. The second leverages equivalence class signatures (EQS) [20, 21] to build an ACT scheme. These two approaches operate in a centralized setting, where a single issuer issues tokens for users. A recent proposal in this field [2] further enhances the action by operating in a distributed-trust environment and proposes a publicly verifiable threshold issuance ACT scheme. The proposed construction is also accompanied by full implementations demonstrating its practicality. To ensure trust among issuers and avoid single points of failure, we leverage the threshold ACT scheme referred to as tACT [1] in our final SPiRiT implementation. Nevertheless, it is important

to mention that, any ACT scheme may serve as a building block in SPiRiT design allowing anonymous user deduplication during the registration phase.

Consider a private unique user attribute denoted as $\mathsf{id_U}$, such as social security number (SSN). ACT mainly aims at allowing users to obtain a deterministic high-entropy token corresponding to $\mathsf{id_U}$ and ensuring issuers remain oblivious to $\mathsf{id_U}$ (oblivious issuance). Furthermore, the token must be *anonymously* verifiable. This means that a public verifier can authenticate the token without learning anything about $\mathsf{id_U}$. In the following, we describe a high-level abstract interface of tACT algorithms. For technical details on concrete instantiations, we refer to [1].

Definition 1 (tACT). *A threshold anonymous counting token scheme consists of the following algorithms (public parameters* pp *are implicit inputs):*

- $(\mathsf{pp}, \{\mathsf{prv}_j\}_{j\in[1,N]}) \leftarrow \mathsf{Setup}(1^\lambda, t, N)$*: Generates system public parameters* pp *and issuer secret keys* prv_j *for* N *issuers with threshold* t*.*
- $(\mathsf{st_{Rg}}, \mathsf{cm}) \leftarrow \mathsf{Register}(\mathsf{id_U})$*: The user commits to identifier* $\mathsf{id_U}$ *by computing* $\mathsf{cm} = \mathsf{Commit}(\mathsf{id_U}; r)$ *and stores registration state* $\mathsf{st_{Rg}} = (\mathsf{id_U}, r)$*.*
- $(\mathsf{blindRequest}, \mathsf{rand}_{\mathsf{id_U}}) \leftarrow \mathsf{TokenRequest}(\mathsf{st_{Rg}}, \mathsf{cm})$*: Generates a blinded token request and corresponding state information.*
- $\mathsf{blindToken}_j \leftarrow \mathsf{tIssue}(\mathsf{blindRequest}, \mathsf{prv}_j)$*: Issuer* I_j *produces a partial blind token.*
- $\mathsf{blindToken} \leftarrow \mathsf{Aggregate}(\{\mathsf{blindToken}_j\}_{j\in[t]})$*: The user combines* t *partial tokens into a blinded token.*
- $\mathsf{token} \leftarrow \mathsf{Unblind}(\mathsf{blindToken}, \mathsf{rand}_{\mathsf{id_U}})$*: The user derives the final token embedding attribute* $\mathsf{id_U}$*.*
- $\pi \leftarrow \mathsf{Prove}(\mathsf{token}, \mathsf{aux}, \mathsf{rand}_{\mathsf{id_U}})$*: Generates a proof of token validity.*
- $\{0,1\} \leftarrow \mathsf{Verify}(\mathsf{token}, \pi, \mathsf{blindRequest})$*: Verifies the token with respect to the blinded request, outputting* 1 *if valid and* 0 *otherwise.*

This scheme provides: *(I)* Unforgeability (Theorem 2, [1]), *(II)* Unlinkability (Theorem 3, [1]), and *(III)* Threshold issuance. Loosely speaking, unforgeability ensures that an adversary controlling less than a threshold of issuers cannot create tokens for more attributes than requested, nor can it generate more than one token for a specific attribute per user. Unlinkability ensures that even issuers cannot link a user's token requests to the corresponding unblinded tokens.

3 Building a Verifiable Contact Tracing System

3.1 Definitions and Properties

Subsequently, we formalize a **S**ecure and **Pri**vacy-Preserving Ve**Ri**fiable Contact **T**racing (SPiRiT) system and provide a precise definition of its essential properties. SPiRiT consists of the following actors.

Registration Authorities. Registration authorities denoted as $\mathbb{I} = \{I_1, \ldots, I_N\}$, form a committee of N entities that run a distributed key generation protocol [31] to generate a secret key sk where party $j \in [1, N]$ receives the share sk_j of sk such

that t shares of the secret reconstruct sk. These authorities attest to the binding between users U and their identifiers $\mathsf{id_U}$ and issue unique tokens to users who aim to join the system during registration process.

User. A user denoted as U, owns a private unique human-readable identifier, such as SSN, denoted as $\mathsf{id_U}$. U interacts with registration authorities in a registration process to obtain a unique token within a privacy-preserving and distributed trust framework. This token empowers U, who wishes to be discoverable to other close-proximity SPiRiT users, to emit verifiable ephemeral pseudo-random IDs all while protecting U's privacy.

Public Health Authority. This authority denoted as PHA acts during the protocol's diagnostic and tracing phases. When a user is diagnosed with the virus, they are required to submit the pseudo-random IDs emitted from their device to the PHA, while contagious. Additionally, the infected individual is tasked with computing an efficient zero-knowledge proof that the PHA can use to verify (I) the authenticity of the pseudo-random IDs corresponding to the contagious period and, (II) verify the individual's legitimate ownership of those IDs. During tracing, each user's device periodically queries the PHA server for newly published pseudo-random IDs and alerts the user of any matches found within the locally stored database of pseudo-random IDs.

Definition 2 (SPiRiT)**.** *A verifiable contact tracing scheme consists of the following algorithms (public parameters* pp *are implicit inputs):*

- $(\mathsf{pp}, \{\mathsf{prv}_j\}_{j\in[1,N]}) \leftarrow \mathsf{Setup}(1^\lambda, t, N)$*: Generates system public parameters* pp *and issuer secret keys* prv_j *for* N *issuers with threshold* t*.*
- $(\mathsf{pp_U}, \mathsf{prv_U}, \mathsf{T_{Rgstr}}) \leftarrow \mathsf{Register}(\mathsf{id_U}, \{\mathsf{prv}_j\}, \mathsf{T_{Rgstr}})$*: Interactive protocol between user (with identifier* $\mathsf{id_U}$*) and issuers. Outputs user parameters* $(\mathsf{pp_U}, \mathsf{prv_U})$*, where* $\mathsf{pp_U}$ *includes a tACT* token*, and updates the registration table* $\mathsf{T_{Rgstr}}$*.*
- $\mathsf{T_{eID}} \leftarrow \mathsf{Broadcast}(i, \mathsf{prv_U}, \mathsf{T_{eID}})$*: At epoch* i*, the user derives an epoch-specific pseudo-random ID from* $\mathsf{prv_U}$ *and broadcasts it. Received beacons and exposure scores are stored locally in* $\mathsf{T_{eID}}$*, which is periodically pruned.*
- $(\mathsf{pp_U}, \mathsf{tr}, \pi_{\mathsf{tr}}) \leftarrow \mathsf{Diagnosis}(\mathsf{pp_U}, \mathsf{prv_U}, \mathbb{CP})$*: A diagnosed user generates trace information* tr *for contagious period* $\mathbb{CP}$ *and a zero-knowledge proof* π_{tr} *attesting to its authenticity.*
- $(\mathsf{db}, \mathsf{bit}) \leftarrow \mathsf{Verify}(\mathsf{tr}, \pi_{\mathsf{tr}}, \mathsf{pp_U}, \mathsf{T_{Rgstr}}, \mathbb{CP})$*: The* PHA *verifies token validity and proof* π_{tr}*, ensuring* tr *corresponds to* $\mathbb{CP}$ *and belongs to the user. If valid (*$\mathsf{bit} = 1$*),* tr *is added to backend database* db*; otherwise* $\perp$ *is returned.*
- $(\mathsf{intCnt}, \mathsf{alrmBit}) \leftarrow \mathsf{Trace}(\mathsf{db}, \mathsf{T_{eID}})$*: The user device compares local records* $\mathsf{T_{eID}}$ *with backend database* db*. If matches exist (*$\mathsf{intCnt} > 0$*), it outputs* $\mathsf{alrmBit} = 1$ *to signal potential exposure.*

Security and Privacy Properties. SPiRiT presents the following core features. In terms of security, we require that an adversary should not be able to upload inauthentic trace information that does not embed a key registered by the presenting user (upload integrity). Furthermore, it must be infeasible for an adversarial user to rebroadcast pseudo-random IDs that were previously sent or received (replay security). The properties of correctness, upload integrity, and

replay security collectively ensure that the adversary is unable to trigger a false alarm for an honest SPiRiT user. More precisely, an adversary cannot cause SPiRiT users not in contact with a diagnosed person to receive an alarm (false positive) or cause users who were in contact not to receive an alarm (false negative). From a privacy perspective, the protocol ensures that the entire process— from registration and ID broadcasting to the disclosure of pseudo-random IDs during the trace phase— preserves the user's information such as the real-world identity of the user, locations, relationships between users, or any information beyond what is available without the use of the exposure notification system, and prevents the linkage of different broadcasts from the same user (unlinkability). Next, we formally define upload integrity, replay security, and unlinkability. To establish upload integrity, we consider an adversary capable of corrupting up to $t-1$ out of N issuers, while also potentially compromising any number of users. For the strongest unlinkability guarantee, we assume an adversary with the power to corrupt any number of issuers and verifiers. The adversary may also corrupt multiple users. However, even with access to multiple compromised devices, the adversary should not be able to gain any additional information beyond what could be obtained by compromising each device individually. Our model excludes adversaries who tamper with the contact tracing app on devices belonging to general users, such as by generating false notifications. We also consider out-of-scope side-channel attacks, such as those exploiting signal strength or power levels to link different transmissions from the same user, as well as physical attacks, like surveillance or the implantation of hidden listening devices. The backend server maintains a publicly available database and is operated by an honest but curious administrator.

Definition 3 (Correctness). *A SPiRiT scheme is correct if for any parameters* $(\mathsf{pp}, \{\mathsf{prv}_j\}_{j\in[1,N]}) \leftarrow \mathsf{Setup}(1^\lambda)$ *and any user* id_U*, after registration* $(\mathsf{pp}_\mathsf{U}, \mathsf{prv}_\mathsf{U}, \mathsf{T}_\mathsf{Rgstr}) \leftarrow \mathsf{Register}(\mathsf{id}_\mathsf{U}, \{\mathsf{prv}_j\}_{j\in[1,N]}, \mathsf{T}_\mathsf{Rgstr})$ *and broadcast* $\mathsf{T}_\mathsf{eID} \leftarrow \mathsf{Broadcast}(\mathsf{prv}_\mathsf{U}, \mathsf{T}_\mathsf{eID})$*, the trace information* $(\mathsf{tr}, \pi_\mathsf{tr}) \leftarrow \mathsf{Diagnosis}(\mathsf{pp}_\mathsf{U}, \mathsf{prv}_\mathsf{U}, \mathbb{CP})$ *produced by an honest user is accepted by* $(\mathsf{db}, \mathsf{bit}) \leftarrow \mathsf{Verify}(\mathsf{tr}, \pi_\mathsf{tr}, \mathsf{pp}_\mathsf{U}, \mathsf{T}_\mathsf{Rgstr}, \mathbb{CP})$ *with overwhelming probability (*$\mathsf{bit} = 1$*). Furthermore, any user whose device previously received pseudo-random IDs from the diagnosed user obtains* $(\mathsf{intCnt}, \mathsf{alrmBit}) \leftarrow \mathsf{Trace}(\mathsf{db}, \mathsf{T}_\mathsf{eID})$ *with* $\mathsf{alrmBit} = 1$*.*

Upload Integrity. Informally, this property describes the infeasibility of a malicious uploader submitting inauthentic trace information. Specifically, an adversary controlling issuing entities in corrupted set CS has a negligible chance of deceiving an honest verifier into believing that the user possesses a list of pseudo-random IDs, if, in reality, presented IDs do not embed a key registered by the presenting user. Figure 2 outlines the detailed game setup, where the adversary begins by registering users of its choice, each possessing a unique identifier. The adversary succeeds by outputting authentic trace information that does not correspond to any of the users it controls. In interactive $\mathsf{Register}$ algorithm, subscripts U and I denote interactions on the user side and the issuer side, respectively.

Definition 4 (Upload Integrity). *A SPiRiT scheme achieves upload integrity if for any PPT adversary* $\mathcal{A}$ *and any* $R \geq 0$

$$\mathbf{Adv}^{UI}_{SPiRiT,\mathcal{A},R}(\lambda) := \Pr[UI_{SPiRiT,\mathcal{A},R}(\lambda) \rightarrow 1] \leq \mathsf{negl}(\lambda),$$

where $UI_{SPiRiT,\mathcal{A},R}(\lambda)$ is defined in Figure 2.

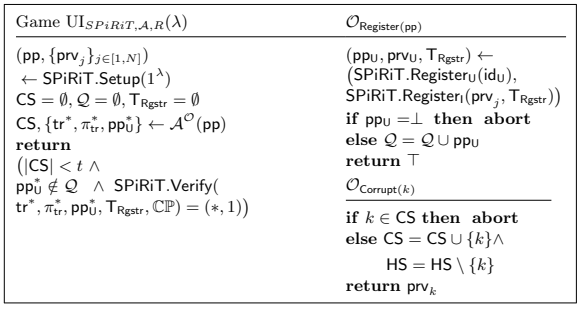


Fig. 2: SPiRiT upload integrity experiment.

Replay Security. This property concerns the integrity of contacts. It captures the infeasibility of an adversary to rebroadcast pseudo-random IDs that were either *sent* or *received* at an earlier time. The definition is based on the replay security game $\mathrm{Replay}_{SPiRiT,\mathcal{A},\Delta}(\lambda)$ as depicted in Figure 3 and follows the same intuition as described in [11, 26]. The game is parameterized by a delay $\Delta \in \mathbb{N}$. Adversary $\mathcal{A}$ selects a time t and honest users $\mathsf{U}_1, \mathsf{U}_2$ to perform a delayed replay attack. Then, $\mathcal{A}$ interacts with U_1 during the period $[0, t - \Delta - 1]$. It then waits for Δ time steps before interacting with U_2. $\mathcal{A}$ wins the game if it causes U_2, which was not in proximity to U_1, to receive false alarms. The notation $\mathsf{SPiRiT}.\Pi_{\mathsf{U}}$, refers to the execution of algorithm $\mathsf{SPiRiT}.\Pi$ on the user U's side.

Definition 5 (Replay Security). *A SPiRiT scheme is replay secure if for any PPT adversary $\mathcal{A}$ and $\Delta \in \mathbb{N}$*

$$\mathbf{Adv}^{Replay}_{SPiRiT,\mathcal{A},\Delta}(\lambda) := \Pr[Replay_{SPiRiT,\mathcal{A},\Delta}(\lambda) \to 1] \leq \mathsf{negl}(\lambda),$$

where $Replay_{SPiRiT,\mathcal{A},\Delta}(\lambda)$ is defined in Figure 3.

Unlinkability. This property captures two distinct privacy features: pseudo-random ID unlinkability and positive user unlinkability. It ensures that an adversarial verifier, potentially colluding with multiple issuers and verifiers, cannot link the execution of Broadcast with either another execution of Broadcast (pseudo-random ID unlinkability for all users, including healthy and positive ones), or with the executions of Register algorithm for a specific user identifier id_{U} (positive-user unlinkability). This is defined by $\mathrm{UNLINK}_{SPiRiT,\mathcal{A}}(\lambda)$ as illustrated in Figure 4. The adversary $\mathcal{A}$, can corrupt a subset CU of users registered through the $\mathcal{O}_{\mathsf{Register}}$ oracle and get access to their private parameters $\mathsf{prv}_{\mathsf{U}}$. Then, $\mathcal{A}$ selects and presents two identifiers $(\mathsf{id}_{\mathsf{U}_0}, \mathsf{id}_{\mathsf{U}_1})$ provided they do not belong to corrupted users. Next, one of them is randomly picked to generate trace information via the $\mathsf{Chl}_{\mathsf{Diagnosis}}$ oracle. The adversary's task is to distinguish eID for the same user across two different Broadcast sessions (pseudo-random ID unlinkability); or to link trace information with the private user identifier (positive-user unlinkability). It is important to note that we do not consider indirect methods of data extraction (side-channel attacks) in our model, such as those exploiting Received Signal Strength Indicator (RSSI) or Bluetooth Low Energy (BLE) tracking mechanisms to link different eIDs from the same user. There exist recent advancements in addressing such vulnerabilities including [24].

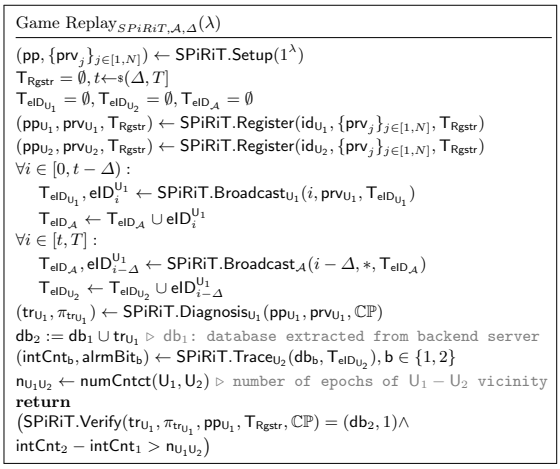


Fig. 3: SPiRiT replay security experiment.

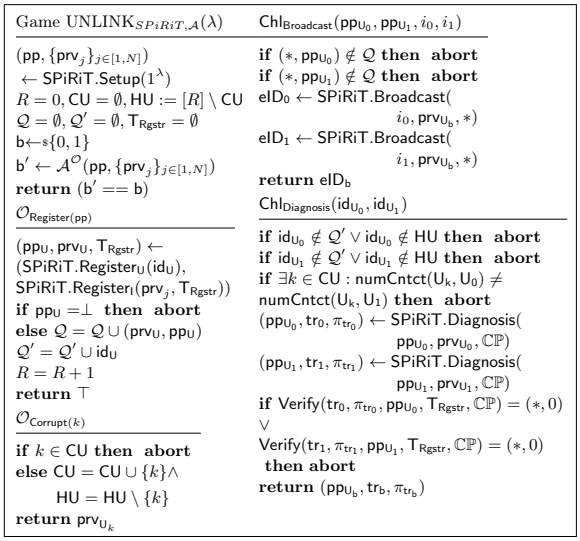


Fig. 4: SPiRiT unlinkability experiment.

Definition 6 (Unlinkability). *A SPiRiT system achieves unlinkability if for PPT adversary $\mathcal{A}$ the advantage defined as*

$$\mathbf{Adv}^{UNLINK}_{SPiRiT,\mathcal{A}}(\lambda) := |2\Pr[UNLINK_{SPiRiT,\mathcal{A}}(\lambda) \rightarrow 1] - 1| \leq \mathsf{negl}(\lambda).$$

where $UNLINK_{SPiRiT,\mathcal{A}}(\lambda)$ is defined in Figure 4.

4 SPiRiT Design

We now introduce SPiRiT, our verifiable contact tracing scheme outlined in Algorithm 1. This protocol is grounded on the tACT system, which facilitates anonymous user registration, thereby ensuring privacy and verifiability in the contact tracing process. In SPiRiT.Register algorithm, token generated in the underlying tACT algorithm serves as a distinctive high-entropy function derived from the private unique identifier id_U–such as SSN–, allowing for anonymous user deduplication and registration. Upon successful verification of token the issuers proceed by comparing the token against their database of redeemed tokens and reject the token, if the value already appears. Otherwise, they update the database by adding the registered token to $\mathsf{T}_{\mathsf{Rgstr}}$. In $\mathsf{T}_{\mathsf{Rgstr}}$, alongside the redeemed token token, issuers also keep a commitment cm_k to the unique key k registered by a user.

In the SPiRiT.Broadcast algorithm, a user's digital device periodically transmits ephemeral pseudo-random IDs while concurrently logging the received pseudo-random IDs from nearby user devices. Specifically, during each epoch with unique epoch number i, the user's device generates an ephemeral pseudo-random ID, denoted as eID, by evaluating a pseudo-random function (PRF) under the user's registered key k and the epoch number i. Hence, $\mathsf{eID}_i = \mathsf{PRF}(\mathsf{k}, i)$ is unique per user and per epoch. The selected PRF must possess a structural design enabling efficient integration with Sigma protocols [16, 15]. Two potential instantiations include the Naor-Pinkas-Reingold PRF [33] $\mathsf{PRF}(\mathsf{k}, i) = \mathsf{H}(i)^\mathsf{k}$ or the Dodis-Yampolskiy PRF [17] $\mathsf{PRF}(\mathsf{k}, i) = g^{(\mathsf{k}+i)^{-1}}$, where g is a generator of a group $\mathbb{G}$. Later in the SPiRiT.Diagnosis algorithm, the user efficiently computes a zero-knowledge proof π_{tr} to prove the correct computation of trace information tr with respect to epoch numbers i during the contagious period, and the user's key k committed to during the registration phase. This reinforces the non-transferability of pseudo-random IDs among users as each user must demonstrate the consistency of claimed IDs with their registered key k. Finally, the user broadcasts the ephemeral pseudo-random ID eID_i via BLE. Concurrently, the user's device listens for BLE beacons broadcasted by other users' devices in close proximity and records them in a table denoted by $\mathsf{T}_{\mathsf{eID}}$. Additionally, the device computes an exposure score es_i and records it in $\mathsf{T}_{\mathsf{eID}}$, which is a function of *proximity* and *contact duration* for each received eID_i, determined based on the latest policies provided by the public health authority (PHA). The algorithm's output comprises the updated database $\mathsf{T}_{\mathsf{eID}}$, which is stored locally on each user's device.

In the SPiRiT.Diagnosis algorithm, following a positive diagnosis, the infected user with public parameter pp_U, computes the tracing information consisting of pseudo-random IDs broadcasted from their device during the contagious period $\mathbb{CP}$[4], denoted as $\mathsf{tr} = \{\mathsf{eID}_i = \mathsf{PRF}(\mathsf{k}, i)\}_{i \in \mathbb{CP}}$. Furthermore, the user computes a zero-knowledge proof π_{tr} to demonstrate the accurate computation of tr across epoch numbers $i \in \mathbb{CP}$, relative to the key k committed to during the registration phase. This process ensures that malicious users cannot claim somebody else's eID as their own, thereby maintaining the integrity of uploads. Subsequently, the user transmits $(\mathsf{pp}_\mathsf{U}, \mathsf{tr}, \pi_{\mathsf{tr}})$ to the PHA for verification.

In the SPiRiT.Verify algorithm, the PHA, upon receiving input $(\mathsf{pp}_\mathsf{U}, \mathsf{tr}, \pi_{\mathsf{tr}})$, first checks whether the user is already registered by checking if $\mathsf{pp}_\mathsf{U} = \{\mathsf{token}, \mathsf{cm}_\mathsf{k}\} \in \mathsf{T}_{\mathsf{Rgstr}}$. If $\mathsf{token} \notin \mathsf{T}_{\mathsf{Rgstr}}$, the user is directed to undergo the registration phase. Else, the PHA

[4] The contagious period denoted as $\mathbb{CP}$, encompasses epoch indices defining the duration considered infectious for a particular virus in accordance with health policies.

Algorithm 1: SPiRiT Construction. Σ denotes tACT scheme from Section 2.

$\underline{\mathsf{SPiRiT.Setup}(1^\lambda, t, N)}$

Run $(\mathsf{pp}', \{\mathsf{prv}_j\}_{j=1}^N) \leftarrow \Sigma.\mathsf{Setup}(1^\lambda, t, N)$ and let $\mathsf{H}'' : \{0,1\}^* \to \mathbb{F}_p$ and $\mathsf{h} : \mathbb{G}_1 \to \mathbb{F}_p$ be secure hash functions.

Let $\mathsf{I} = \{I_1, \ldots, I_N\}$ denote the set of authorized issuers. Issuers initialize a shared table $\mathsf{T}_{\mathsf{Rgstr}} := \emptyset$. Each user device initializes a bit $\mathsf{alrmBit} \leftarrow 0$ and a local table $\mathsf{T}_{\mathsf{eID}} := \emptyset$.

Output: System public parameters $\mathsf{pp} = (\mathsf{pp}', \mathsf{H}'', \mathsf{h}, \mathsf{T}_{\mathsf{Rgstr}})$, issuers' private parameters $\{\mathsf{prv}_j\}_{j\in[1,N]}$

$\underline{\mathsf{SPiRiT.Register}(\mathsf{id}_\mathsf{U}, \{\mathsf{prv}_j\}_{j\in[1,N]}, \mathsf{T}_{\mathsf{Rgstr}})}$ [a user U and issuers I]:

U on input private unique identifier id_U runs $(\mathsf{blindRequest}, \mathsf{rand}_\mathsf{U}) \leftarrow \Sigma.(\mathsf{Register}, \mathsf{TokenRequest})(\mathsf{id}_\mathsf{U})$. I_j runs $\mathsf{blindToken}_j \leftarrow \Sigma.\mathsf{tIssue}(\mathsf{blindRequest}, \mathsf{prv}_j)$

U runs $\mathsf{token} \leftarrow \Sigma.(\mathsf{Aggregate}, \mathsf{Unblind})(\{\mathsf{blindToken}_{jk}\}_{j\in[t]}, \mathsf{rand}_\mathsf{U})$, and $\pi \leftarrow \Sigma.\mathsf{Prove}(\mathsf{token}, \mathsf{rand}_\mathsf{U})$

U samples a key $\mathsf{k} \leftarrow\!\!\$\ \mathbb{F}_p$ and outputs $\mathsf{cm}_\mathsf{k} \leftarrow \mathsf{Ped.Commit}(\mathsf{k}, r_\mathsf{k})$, where $r_\mathsf{k} \leftarrow\!\!\$\ \mathbb{F}_p$

I check $\mathsf{bit} \leftarrow \Sigma.\mathsf{Verify}(\mathsf{token}, \pi)$: **if** $(\mathsf{bit} = 0 \vee (\mathsf{bit} = 1 \wedge \mathsf{token} \in \mathsf{T}_{\mathsf{Rgstr}}))$, abort. **Else**, $\mathsf{T}_{\mathsf{Rgstr}} \leftarrow \mathsf{T}_{\mathsf{Rgstr}} \cup \{\mathsf{token}, \mathsf{cm}_\mathsf{k}\}$.

Output: $\mathsf{pp}_\mathsf{U} \leftarrow \{\mathsf{token}, \mathsf{cm}_\mathsf{k}\}$, $\mathsf{prv}_\mathsf{U} \leftarrow \{\mathsf{id}_\mathsf{U}, \mathsf{k}\}$, $\mathsf{T}_{\mathsf{Rgstr}}$

$\underline{\mathsf{SPiRiT.Broadcast}(i, \mathsf{prv}_\mathsf{U}, \mathsf{T}_{\mathsf{eID}})}$ [a user U]:

At each epoch with unique epoch number i, U's device does the following:

a: Broadcast pseudo-random ID– Generate $\mathsf{eID}_i = \mathsf{PRF}(\mathsf{k}, i)$ and broadcast eID_i via BLE.

b: Log received BLE beacons– Compute an exposure score es_i for each received eID_i, record $(\mathsf{eID}_i, \mathsf{es}_i)$ in $\mathsf{T}_{\mathsf{eID}}$.

Output: $\mathsf{T}_{\mathsf{eID}}$

$\underline{\mathsf{SPiRiT.Diagnosis}(\mathsf{pp}_\mathsf{U}, \mathsf{prv}_\mathsf{U}, \mathbb{CP})}$ [a user U]:

if U receives a positive diagnosis test result **then**

a: Parse $\mathsf{pp}_\mathsf{U} = \{\mathsf{token}, \mathsf{cm}_\mathsf{k}\}$, $\mathsf{prv}_\mathsf{U} = \{\mathsf{id}_\mathsf{U}, \mathsf{k}\}$. ▷ token : unique anonymous token acquired during registration phase.

b: Re-computes the following vector as the trace information $\mathsf{tr} \leftarrow \{\mathsf{eID}_i = \mathsf{PRF}(\mathsf{k}, i)\}_{i\in\mathbb{CP}}$ ▷ pseudo-random IDs broadcasted from U's device during $\mathbb{CP}$.

c: Generates $\pi_{\mathsf{tr}} = \mathsf{ZKPoK}\{(\mathsf{k}, r_\mathsf{k}) : \mathsf{cm}_\mathsf{k} = \mathsf{Ped.Commit}(\mathsf{k}, r_\mathsf{k}), \{\mathsf{eID}_i = \mathsf{PRF}(\mathsf{k}, i)\}_{i\in\mathbb{CP}}\}$

Output: $(\mathsf{pp}_\mathsf{U}, \mathsf{tr}, \pi_{\mathsf{tr}})$ to PHA.

$\underline{\mathsf{SPiRiT.Verify}(\mathsf{tr}, \pi_{\mathsf{tr}}, \mathsf{pp}_\mathsf{U}, \mathsf{T}_{\mathsf{Rgstr}}, \mathbb{CP})}$ [public health authority PHA]:

The PHA, upon receiving input from a positively diagnosed U, does the following:

a: Parse $\mathsf{pp}_\mathsf{U} = \{\mathsf{token}, \mathsf{cm}_\mathsf{k}\}$

b: **if** $\{\mathsf{token}, \mathsf{cm}_\mathsf{k}\} \in \mathsf{T}_{\mathsf{Rgstr}} \wedge \mathsf{ZKVerify}(\mathsf{cm}_\mathsf{k}, \pi_{\mathsf{tr}}) = 1$ **then** $\mathsf{bit} \leftarrow 1 \wedge \forall i \in \mathbb{CP} : \mathsf{CF} \leftarrow \mathsf{CF.Insert}(\mathsf{tr}[i])$. **Else**, $\mathsf{bit} \leftarrow 0$. ▷ insert items $\{\mathsf{tr}[i] = \mathsf{eID}_i\}_{i\in\mathbb{CP}}$ into CF.

Output: $(\mathsf{CF}, \mathsf{bit})$ to the backend server database.

$\underline{\mathsf{SPiRiT.Trace}(\mathsf{CF}, \mathsf{T}_{\mathsf{eID}})}$ [all system users]:

Each user device periodically queries the backend server for updated CF and does the following:

for $i \in [1, |\mathsf{T}_{\mathsf{eID}}|]$ **do**

a: $\mathsf{bit} \leftarrow \mathsf{CF.Check}(\mathsf{T}_{\mathsf{eID}}[i])$ ▷ Compute the set intersection of CF and user's local database $\mathsf{T}_{\mathsf{eID}}$ through the lookup operation.

b: **if** $\mathsf{bit} = 1 \wedge\ \mathsf{es}_i \geq \mathtt{exposurelimit}$ **then** $\mathsf{intCnt} \leftarrow \mathsf{intCnt} + 1$

Output: $\mathsf{intCnt}, \mathsf{alrmBit} \leftarrow (\mathsf{intCnt} \geq 1)?1 : 0$

uses the commitment cm_k to the user-specific registered key k and verifies the zero-knowledge proof π_{tr}, ensuring the authenticity of the returned eIDs with regard to: *(I)* the key k *(II)* the epoch numbers in contagious period $i \in \mathbb{CP}$, all while maintaining anonymity. More precisely, first, the verification ensures that only the true owner can release the eIDs (non-transferability). This is achieved by tethering the eIDs to the committed unique user key k. Second, this verification prevents malicious users from revealing a different set of eIDs instead of the ones corresponding to the contagious period (replay attack). This is made possible due to the design of eIDs which are bound to epoch numbers i. Upon successful verification, the PHA inserts the list $\{\mathsf{eID}_i\}_{i\in\mathbb{CP}}$ into a Cuckoo filter CF for compact representation, which is subsequently stored on the backend server. A Cuckoo filter (CF) is a probabilistic data structure with a compact set representation facilitating fast membership testing [19]. We rely on standard definitions of Cuckoo filter [19].

In the $\mathsf{SPiRiT.Trace}$ algorithm, each user device periodically queries the backend server for updates in CF. The user device then cross-references its local database $\mathsf{T}_{\mathsf{elD}}$ with CF to determine the intersection. More precisely, the algorithm looks up all pseudo-random ID in $\mathsf{T}_{\mathsf{elD}}$, checking if each one of them belongs to the filter CF through the lookup operation $\mathsf{CF.Check}$. It then assesses an exposure score es for all matching pseudo-random IDs. If the exposure score exceeds a threshold denoted as `exposurelimit` determined by the PHA, a bit $\mathsf{alrmBit}$ is set to 1, triggering the device to display a notification that alerts the user of potential exposure to the virus due to prolonged close proximity to the infected individual(s). The notification offers guidance on necessary actions including staying in quarantine and getting tested for the viral infection and directs the user to additional information resources. Furthermore, to save storage space and improve trace lookup efficiency, applications automatically delete records that are no longer relevant for epidemiological purposes, e.g., those that exceed a 10-day timeframe. Similarly, the backend server can remove old items from the Cuckoo filter using the $\mathsf{CF.Delete}$ operation. This deletion capability allows the filter to accommodate new items without necessitating reconfiguration or expansion.

Remark. Here we describe minor modifications to Algorithm 1 necessary to accommodate passive users. First, the PHA must establish an upper limit on the percentage of users who can opt for passive participation. This is essential to maintain a balance between accommodating user preferences and ensuring the SPiRiT system's effectiveness in detecting high-risk cases. Increasing the number of passive users will inevitably reduce the overall detection probability, as these users refrain from broadcasting pseudorandom IDs. During registration, users are required to specify their participation mode—either active or passive. Consequently, $\mathsf{T}_{\mathsf{Rgstr}}$ will also reflect the mode chosen by the user. The Broadcast, Diagnosis, and Trace algorithms for passive users, which follow same intuition as the A/P DP-3T protocol [3], are as follows:

- $\mathsf{SPiRiT.Broadcast}$: Passive users do not broadcast pseudo-random IDs; they only listen for broadcasts from others.
- $\mathsf{SPiRiT.Diagnosis}$: If a passive user becomes infected, and the PHA confirms their passive status based on registration data, they are not required to upload tracing information. Consequently, $\mathsf{SPiRiT.Verify}$ algorithm is not executed for passive users.
- $\mathsf{SPiRiT.Trace}$: This algorithm for passive users operates in the same way as it does in Algorithm 1 for active users.

4.1 Security Analysis of the Proposed Scheme

Theorem 1. *[Upload Integrity] If* $(\mathsf{ZKSetup}, \mathsf{ZKPoK}, \mathsf{ZKVerify})$ *is a* NIZK *with knowledge soundness property, and the* tACT *scheme is an unforgeable threshold anonymous counting token, then, the* SPiRiT *scheme described in Algorithm 1 satisfies upload integrity as in Definition Definition 4.*

Proof. Assume an adversary $\mathcal{A}$ can successfully cheat the upload integrity game in Figure 2 with non-negligible probability. We construct two efficient adversaries $\mathcal{B}_1$ and $\mathcal{B}_2$. $\mathcal{B}_1$ can break the unforgeability of the tACT system, while $\mathcal{B}_2$ can compromise the soundness of the NIZK argument. We will show that:

$$\mathbf{Adv}^{\mathsf{UI}}_{\mathsf{SPiRiT},\mathcal{A}} \leq \mathbf{Adv}^{\mathsf{UF}}_{\mathsf{tACT},\mathcal{B}_1}(\lambda) + \mathbf{Adv}^{\mathsf{zksound}}_{\mathsf{NIZK},\mathcal{B}_2}(\lambda). \tag{1}$$

Construction of $\mathcal{B}_1$. $\mathcal{B}_1$ interacts with a challenger C in the unforgeability game for tACT and $\mathcal{B}_1$ simulates the upload integrity game for $\mathcal{A}$.

- C runs $(\mathsf{pp}, \mathsf{prv}) \leftarrow \mathsf{tACT.Setup}(1^\lambda)$ and gives public parameters pp to $\mathcal{B}_1$. Then $\mathcal{B}_1$ provides pp to $\mathcal{A}$.
- $\mathcal{B}_1$ simulates the environment and oracles for $\mathcal{A}$. When oracles $\mathcal{O}_{\mathsf{Corrupt}}$ and $\mathcal{O}_{\mathsf{Register}}$ are called, $\mathcal{B}_1$ queries $\mathcal{O}_{\mathsf{Corrupt}}$ and ($\mathcal{O}_{\mathsf{Register}}$,$\mathcal{O}_{\mathsf{tIssue}}$) of the tACT scheme, respectively. $\mathcal{B}_1$ answers register queries $\mathsf{blindRequest}_i$ from $\mathcal{A}$ as follows. $\mathcal{B}_1$ queries C and provides the answer tok_i. In addition, $\mathcal{B}_1$ runs Ext, extractor algorithm of the underlying ZKPoK to extract the values r_i and $\mathsf{id}_{\mathsf{U}i}$. Subsequently, $\mathcal{B}_1$ sets $\mathcal{Q} = \mathcal{Q} \cup \{\mathsf{id}_{\mathsf{U}i}, r_i, \mathsf{bR}_i, \mathsf{tok}_i, \pi_{\mathsf{tok}_i}\}$.
- Upon game completion, $\mathcal{A}$ generates the output $(\mathsf{tr}^*, \pi^*_{\mathsf{tr}}, {\mathsf{pp}_{\mathsf{U}}}^* = (\mathsf{tok}^*, \pi^*_{\mathsf{tok}}, {\mathsf{cm}_{\mathsf{k}}}^*)) \notin \mathcal{Q}$. $\mathcal{B}_1$ submits $((\mathsf{id}^*_{\mathsf{U}i}, r^*_i, \mathsf{bR}^*_i, \mathsf{tok}^*, \pi^*_{\mathsf{tok}}) \cup \mathcal{Q})$ to the tACT challenger C as its forgery.

Construction of $\mathcal{B}_2$. $\mathcal{B}_2$ interacts with a challenger C' in the soundness game for NIZK. At the end of the game, $\mathcal{A}$ produces the outputs $(\mathsf{tr}^*, \pi^*_{\mathsf{tr}}, {\mathsf{pp}_{\mathsf{U}}}^* = (\mathsf{tok}^*, \pi^*_{\mathsf{tok}}, {\mathsf{cm}_{\mathsf{k}}}^*))$ for which $\mathsf{SPiRiT.Verify}$ outputs 1. $\mathcal{B}_2$ returns $(\pi^*_{\mathsf{tr}}, {\mathsf{pp}_{\mathsf{U}}}^* = (\mathsf{tok}^*, \pi^*_{\mathsf{tok}}, {\mathsf{cm}_{\mathsf{k}}}^*))$ to the NIZK challenger C' as proof of an incorrect statement. Notably, neither $\mathcal{B}_1$ nor $\mathcal{B}_2$ can determine in polynomial time if their submissions to the respective challengers tACT/NIZK will be successful, i.e., whether they will win the $\mathsf{tACT}_{\mathcal{B}_1}(\lambda)$ and $\mathsf{zkSND}_{\mathcal{B}_2}(\lambda)$ games. Nonetheless, this does not hinder the proof that at least one of adversaries $\mathcal{B}_1$ and $\mathcal{B}_2$ must have a non-negligible advantage if the adversary $\mathcal{A}$ can breach upload integrity experiment with non-negligible probability. Our proof strategy involves defining two events, namely $\mathsf{E}_{\mathsf{tACT}}$, and $\mathsf{E}_{\mathsf{zkSND}}$ with the following conditions:

- Only one of these events must occur if the adversary $\mathcal{A}$ wins.
- If either event happens, either adversary $\mathcal{B}_1$, or $\mathcal{B}_2$ will successfully break unforgeability or soundness, respectively.

Definition of Events. Consider the experiment $\mathsf{UI}_{\mathsf{SPiRiT},\mathcal{A}}(\lambda)$ for which we define the following two events:

- Event $\mathsf{E}_{\mathsf{tACT}}$: The adversary $\mathcal{A}$ terminates and produces a $\mathsf{pp}_{\mathsf{U}} = (\mathsf{token}, \ldots)$ and π_{tok} that are not in the Register query list (while computing a valid trace tr and its associated NIZK proof π_{tr}).

– Event $\mathsf{E}_{\mathsf{zkSND}}$: The adversary $\mathcal{A}$ terminates and produces an invalid trace tr accompanied by a valid proof π_{tr} (while presenting a valid token). Specifically, π_{tr} is valid but proves a wrong tr as NIZK statement, and, and the conditions for event $\mathsf{E}_{\mathsf{tACT}}$ must not be met.

Conclusion of the Proof. Given that exactly one of the events $\mathsf{E}_{\mathsf{tACT}}$ and $\mathsf{E}_{\mathsf{zkSND}}$ must occur and these events are disjoint, by applying the total probability rule we can write:

$$\mathbf{Adv}^{\mathsf{UI}}_{\mathsf{SPiRiT},\mathcal{A}} = \mathsf{Pr}[\mathsf{UI}_{\mathcal{A}}(\lambda) = 1] = \mathsf{Pr}[\mathsf{UI}_{\mathcal{A}}(\lambda) = 1 \wedge \mathsf{E}_{\mathsf{tACT}}] + \mathsf{Pr}[\mathsf{UI}_{\mathcal{A}}(\lambda) = 1 \wedge \mathsf{E}_{\mathsf{zkSND}}]$$

Furthermore, considering the construction of $\mathcal{B}_1$ and the fact that this reduction perfectly simulates the environment for $\mathcal{A}$, we have:

$$\mathsf{Pr}[\mathsf{UI}_{\mathcal{A}}(\lambda) = 1 \wedge \mathsf{E}_{\mathsf{tACT}}] = \mathsf{Pr}[\mathsf{UF}_{\mathsf{tACT},\mathcal{B}_1}(\lambda) = 1 \wedge \mathsf{Event}_{\mathsf{tACT}}] \leq \mathsf{Pr}[\mathsf{UF}_{\mathsf{tACT},\mathcal{B}_1}(\lambda) = 1]$$

Likewise, we have, $\mathsf{Pr}[\mathsf{UI}_{\mathcal{A}}(\lambda) = 1 \wedge \mathsf{E}_{\mathsf{zkSND}}] \leq \mathsf{Pr}[\mathsf{zkSND}_{\mathsf{NIZK},\mathcal{B}_2}(\lambda) = 1]$. This proves Equation (1) and concludes the proof of Theorem 1.

Theorem 2. *[Replay Security] If* $(\mathsf{ZKSetup}, \mathsf{ZKPoK}, \mathsf{ZKVerify})$ *is a* NIZK *with knowledge soundness property, then, the* SPiRiT *scheme described in Algorithm 1 satisfies replay security as in Definition Definition 5.*

Proof. Replay security aims to prevent false positives meaning that an adversary $\mathcal{M}$ cannot cause an honest SPiRiT user B not in contact with an honest diagnosed user A or corrupted diagnosed user C to receive an alarm. Consider the experiment $\mathsf{Replay}_{\mathsf{SPiRiT},\mathcal{M},\Delta}(\lambda)$ for which we define the following two events, where only one of these events must occur if the adversary $\mathcal{M}$ wins and it causes B, which was not in proximity to an infected user to receive false alarms.

– Event E_1: The adversary $\mathcal{M}$ has been in close proximity to honest user B. Also, $\mathcal{M}$ controls an infected corrupted user C. This event requires proximity of adversary $\mathcal{M}$ to B, while B and C have not been in contact with each other. The adversary $\mathcal{M}$ terminates at time t and rebroadcasts IDs that were sent by C (and known to $\mathcal{M}$) at time $t - \Delta$.
– Event E_2: The adversary $\mathcal{M}$ has been in close proximity to both honest users A and B, with A being infected, but B and A have not been in contact with each other. The adversary $\mathcal{M}$ terminates at time t and rebroadcasts IDs that it previously received from A at time $t - \Delta$.

For an honest user B to receive a false alarm, there must be an intersection between the set of announced IDs and the IDs stored on B's device. This can only happen if B's local database contains an ID associated with C (event E_1: warning triggered by a corrupted user C) or an ID associated with A (event E_2: warning triggered by an honest user A). Given that one of the events E_1 and E_2 must occur for the adversary to succeed in the experiment in Figure 3, we can write:

$$\mathbf{Adv}^{\mathsf{Replay}}_{\mathsf{SPiRiT},\mathcal{M}} = \mathsf{Pr}[\mathsf{Replay}_{\mathcal{M}}(\lambda) = 1] = \mathsf{Pr}[\mathsf{Replay}_{\mathcal{M}}(\lambda) = 1 \wedge \mathsf{E}_1] + \mathsf{Pr}[\mathsf{Replay}_{\mathcal{M}}(\lambda) = 1 \wedge \mathsf{E}_2]$$

If the conditions for event E_1 are met and the honest user B receives a false alarm, it implies that B's local database contains an ID associated with the corrupted user C, and that the epoch numbers corresponding to the matching IDs in both

B's local database and the backend server's database must match. This matching is necessary because the epoch number is integrated into the ID generation process. For this to happen, the corrupted user C must produce an invalid trace tr accompanied by a valid proof π_{tr} (while presenting a valid token). Essentially, π_{tr} is valid but proves an incorrect trace tr as a NIZK statement, unless the backend server is compromised and updates the database with incorrect data and manipulated epoch numbers. Therefore, we conclude:

$$\mathsf{Pr}[\mathsf{Replay}_{\mathcal{M}}(\lambda) = 1 \wedge \mathsf{E}_1] \leq \mathsf{Pr}[\mathsf{zkSND}_{\mathsf{NIZK},\mathcal{B}_2}(\lambda) = 1].$$

Where $\mathcal{B}_2$ is constructed similarly to Section 4.1. Likewise, we have,

$$\mathsf{Pr}[\mathsf{Replay}_{\mathcal{M}}(\lambda) = 1 \wedge \mathsf{E}_2] \leq \mathsf{Pr}[\mathsf{zkSND}_{\mathsf{NIZK},\mathcal{B}_2}(\lambda) = 1].$$

Hence, assuming the knowledge soundness property of NIZK, this concludes the proof of Theorem 2.

Theorem 3. *[Unlinkability] If the* tACT *scheme is an unlinkable threshold anonymous counting token, the* $\mathsf{Ped.Commit}$ *scheme provides hiding property, and* NIZK : $(\mathsf{ZKSetup}, \mathsf{ZKPoK}, \mathsf{ZKVerify})$ *provides zero-knowledge property, then, the* SPiRiT *scheme described in Algorithm 1 satisfies unlinkability as in Definition Definition 6 in the random oracle model.*

Proof. We prove the theorem using a sequence of hybrid games.

G_0. The original game defined in Figure 4.

G_1. Replace honest $\mathsf{Diagnosis}$ proofs with simulated proofs for the challenge token. By the zero-knowledge property of NIZK, G_0 and G_1 are computationally indistinguishable.

G_2. Replace PRF evaluations $\mathsf{eID}_i = \mathsf{PRF}(\mathsf{k}, i)$ in $\mathsf{Broadcast}$ queries with random values. Indistinguishability follows from PRF security.

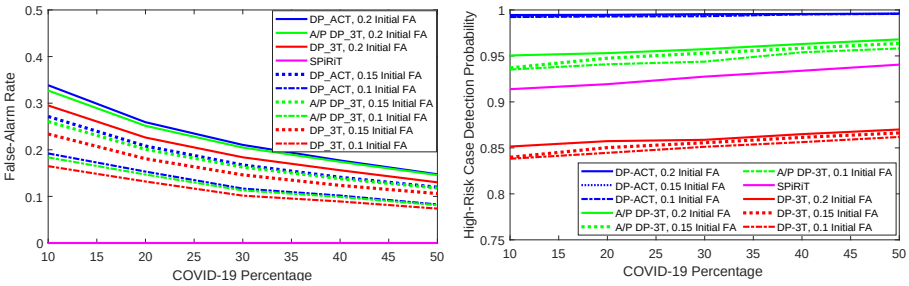
G_3. This game is similar to its predecessor except that in any $\mathsf{Register}$ query, we change $\mathsf{Ped.Commit}$ evaluations for the key k as $\mathsf{cm}_{\mathsf{k}} = \mathsf{Ped.Commit}(\mathsf{k}, r_{\mathsf{k}})$ to randomly sampled cm_{k}. This change is indistinguishable due to the hiding property of $\mathsf{Ped.Commit}$ and the zero-knowledge property of NIZK.

G_4. Replace the user identifier id_{U} with a random string during registration. By the hiding property of $\mathsf{Ped.Commit}$ and the unlinkability of tACT, G_3 and G_4 are indistinguishable.

In G_4, $\mathsf{Register}$ queries are independent of id_{U}, and $\mathsf{Chl}_{\mathsf{Diagnosis}}$ returns fresh parameters and simulated proofs. Hence the adversary's advantage in guessing b is negligible under the security of PRF, the hiding property of $\mathsf{Ped.Commit}$, the zero-knowledge property of NIZK, and the unlinkability of tACT.

5 Evaluation

In this section, we evaluate the proposed protocol using a face-to-face individual interaction dataset, specifically the InVS15 dataset [22]. The InVS15 dataset comprises face-to-face interactions measured over 12 days in an office building in France in 2015. We assume all participants use COVID-19 contact tracing mobile applications. The median incubation period (time from exposure to symptom onset) for COVID-19 is reported to be 5 days, but it can range from 2 to 14 days [28]. Studies indicate that people can start transmitting the virus even before symptoms appear, typically around 1 to 2 days before they feel sick [12]. We assume that all users who had high-risk contact with infected individuals will become infected and can transmit the virus to



(a) (b)

Fig. 5: (a) High-risk case detection probability versus the COVID-19 percentage; (b) False-alarm rate versus the COVID-19 percentage; for initial false-alarm rates of 0.1, 0.15, and 0.2, with 10% of users being passive.

others after 4 days from the high-risk contact. In addition, we assume that none of these newly infected users can be detected without receiving high-risk alarms due to the lack of symptoms. We consider several scenarios with varying percentages of initially infected users. The objective is to identify all users with high-risk contacts with both the initial infected users and the secondary high-risk users after 4 days. We update the set of infected users likely to transfer the virus to other users (after 4 days from having a high-risk contact) day by day. Here, a high-risk contact is defined as a face-to-face interaction lasting 5 minutes or more. In our simulations, we conduct 1,000 runs with randomly selected COVID-19-infected users and passive users. The *high-risk case detection probability* is the ratio of detected high-risk cases to the total number of high-risk cases, averaged over these 1,000 runs. A high-risk case is an individual, who had a high-risk contact with an infected person (either those initially revealed or those infected by the initially revealed and capable of transmitting the virus after 4 days). The *COVID-19 percentage* is defined as the number of initially revealed COVID-19-infected users relative to the total number of users.

5.1 False Positive

In the following, we explore the impact of attackers attempting to falsely reveal the IDs of the non-infected users as those of the infected ones. We define the *initial false-alarm rate* as the proportion of users falsely revealed as infected to the total number of users. We also define the *false-alarm rate* as the proportion of users falsely identified as infected (including those falsely revealed as infected) to the total number of users. In each run of our experiment, the users falsely revealed as infected are selected randomly. In Figure 5a, we illustrate the false-alarm rate versus the COVID-19 percentage for initial false-alarm rates of 0.1, 0.15, and 0.2, with 10% of users being passive. Figure 5b also shows the high-risk case detection probability versus the COVID-19 percentage for initial false-alarm rates of 0.1, 0.15, and 0.2, with 10% of users being passive. As can be seen, having an initial false-alarm rate results in a slightly higher high-risk case detection probability. This is because some real high-risk cases have high-risk contact with the falsely identified users. However, as this initial false alarm propagates over our study period, it results to a significant false-alarm rate after 12 days according to Fig 5a. The proposed SPiRiT protocol, however, is not affected by the initial false alarm

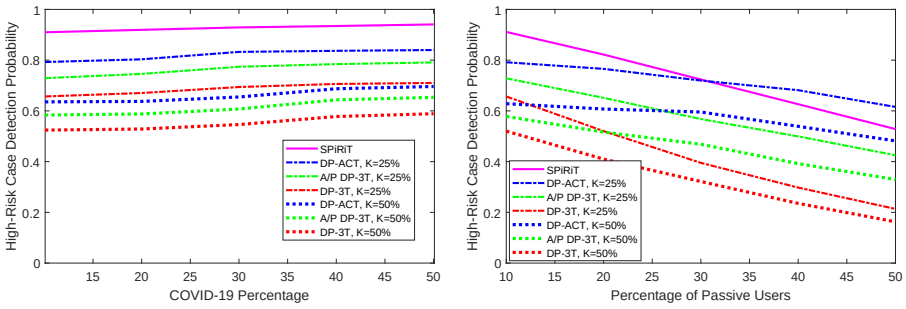


Fig. 6: (a) High-risk case detection probability versus the COVID-19 percentage with 10% of users being passive; (b) High-risk case detection probability versus the percentage of passive users with 10% of users being diagnosed with COVID-19; assuming that 20% and 50% percentages of the broadcasted IDs during the contagious period are not revealed by the diagnosed users.

due to integrating the verifiability mechanism. The false-alarm rate is 0 when using the proposed SPiRiT protocol, while our performance is better than DP-3T and on par with A/P DP-3T according to Fig 5b.

5.2 False Negative

We assume that diagnosed users tend not to reveal a certain percentage of their broadcasted IDs during their contagious period, denoted by K. Fig. 6a illustrates the high-risk case detection probability versus the COVID-19 percentage with 10% of users being passive, assuming that 20% and 50% of the broadcasted IDs during the contagious period are not revealed by the diagnosed users. Furthermore, Fig. 6b shows the high-risk case detection probability versus the percentage of passive users with 10% of users being diagnosed with COVID-19 assuming that 20% and 50% percentages of the broadcasted IDs during the contagious period are not revealed by the diagnosed users. As can be seen from Fig. 6, if some percentage of the broadcasted IDs during the contagious period are not revealed by the diagnosed users, the performance of the COVID-19 contact tracing can decrease significantly. However, this issue is resolved in the proposed SPiRiT protocol.

5.3 Efficiency Analysis

We now provide a cost analysis of SPiRiT. We leverage the tACT proposed in [1] along with Naor-Pinkas-Reingold PRF [33] in our instantiation. Our implementation is developed in Rust using the `ark_bls12_381` library, compiled for cross-platform execution on ARM-based mobile devices (`aarch64-apple-ios` for iOS, `aarch64-linux-android` for Android) via the Rust toolchain, ensuring portability and near-native performance. The code is not optimized, leaving room for potential performance improvements in future iterations. The issuer runtime costs are measured on an Intel Core i7-1265U with 16 GB of RAM spanning high-performance, mid-range, and older hardware still in use. The overview of the runtime costs, broken down by individual algorithms, is presented in Table 2. With $t = \frac{N}{2} + 1$ as parameter of tACT, where N denotes the number of issuers, running Register on the user side requires approximately 359 ms for

Table 2: Estimated runtime of SPiRiT with $N = 4$, $n = 30$, $t = \frac{N}{2} + 1$, $b = 3$, $v = 32$, $\epsilon_{max} \approx 2^{-29}$. All runtimes are given in milliseconds. Register is a one-time process, and Diagnosis is only necessary to be run by a user once the user gets infected. Trace is executed on a daily basis (e.g., daily scanning). $\mathsf{Register}_{\mathsf{I}}$ and Verify were obtained on an Intel Core i7-1265U, while user-side operations ($\mathsf{Register}_{\mathsf{U}}$, Broadcast, Diagnosis, Trace) were obtained on the specified devices.

Algorithm	iPhone 13 Pro	Galaxy S10	Huawei P10
$\mathsf{Register}_{\mathsf{I}}$	103.6	103.6	103.6
$\mathsf{Register}_{\mathsf{U}}$	359.0	1333.4	2592.7
Broadcast	0.1	0.3	0.6
Diagnosis	23.9	88.9	172.8
Verify	50.0	50.0	50.0
Trace	346.2	1285.7	2500.0

$N = 4$ on the iPhone 13 Pro, increasing to 2592.7 ms on the Huawei P10. We note that Register is a *one-time* operation executed as a single-round protocol, that can be run in parallel with all issuers. This mitigates the impact of Register on overall system performance. Also, Diagnosis is *only necessary to be run by a user once the user gets infected*. On the other hand, algorithms Broadcast and Trace are expected to be executed frequently by system users during the system's operation. As such, their runtime performance is critical for ensuring the scheme's practical efficiency and scalability in real-world deployments.

The Trace algorithm follows the same operational steps as in prior works; thus, we expect its runtime performance to be comparable, since the verifiability features of our scheme do not impact the trace phase. For our implementation, the Cuckoo filter parameters follow the configuration in [29], with a bucket size of $b = 3$, tag length $v = 32$, and a maximum false positive rate of $\epsilon_{\max} \approx 2^{-29}$. In Trace algorithm, the user device looks up the downloaded CF to determine the intersection. Consequently, the computational cost of the Trace is linear with respect to the size of the table $\mathsf{T}_{\mathsf{elD}}$ stored on the user device. Assuming a relevant epidemiological timeframe of 10 days and an epoch duration i of 15 minutes, we have $|\mathsf{T}_{\mathsf{elD}}| = 960$. This results in an average total lookup time of 346.2 ms on an iPhone 13 Pro and up to 2500 ms on the slowest tested device, demonstrating the practicality of the protocol for real-time usage.

The primary communication overhead arises from transmitting the CF from the server to the user. For instance, based on the database size considered in [29], a compressed CF representing a server database of size $|DB| = 2^{28}$ with a FPR of $\epsilon_{max} \approx 2^{-29}$, has a size of $\approx 1\ GiB$. Such a large transmission is impractical for mobile networks due to bandwidth limitations and data costs. To address this bottleneck, we refer to the techniques outlined in [29] to reduce the user-server communication complexity to logarithmic in the size of the server database DB. Additionally, any changes in the server database–such as insertions or deletions–trigger an update phase. Instead of retransmitting the entire CF, the server sends a small delta update to the user device, which reflects only the modified entries. This incremental update mechanism further minimizes the communication overhead.

6 Discussion and Conclusion

The effectiveness of contact tracing systems ultimately depends on user participation, as detection coverage increases approximately quadratically with the fraction of app users in the population [14]. By enhancing reliability through verifiable reporting and minimizing false alarms, SPiRiT strengthens user trust, an essential factor for

widespread uptake. While our simulation results may not fully capture this dynamic, we anticipate that the protocol will achieve higher detection rates in the long run, as user adoption increases, ultimately enhancing its long-term efficacy.

Acknowledgments. This work received funding from the EU research project SWARMCHESTRATE (project no. 101135012).

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Abdolmaleki, B., Michalas, A., Rabaninejad, R., Ramacher, S., Slamanig, D.: Attribute-based threshold issuance anonymous counting tokens and its application to sybil-resistant self-sovereign identity. Cryptology ePrint Archive, Paper 2024/1024 (2024), https://eprint.iacr.org/2024/1024
2. Abdolmaleki, B., Michalas, A., Rabaninejad*, R., Ramacher, S., Slamanig, D.: Attribute-based threshold issuance anonymous counting tokens and its application to sybil-resistant self-sovereign identity. IACR Communications in Cryptology **3**(1) (2026). https://doi.org/10.62056/ayivr-zn4
3. Abtahi, A., et al.: Dp-act: Decentralized privacy-preserving asymmetric digital contact tracing. Proceedings on Privacy Enhancing Technologies (2024)
4. Altmann, S., et al.: Acceptability of app-based contact tracing for covid-19: Cross-country survey study. JMIR mHealth and uHealth **8**(8), e19857 (2020)
5. Apple, Google: Privacy-Preserving Contact Tracing. http://www.apple.com/covid19/contacttracing (2020)
6. Auerbach, B., Chakraborty, S., Klein, K., Pascual-Perez, G., Pietrzak, K., Walter, M., Yeo, M.: Inverse-sybil attacks in automated contact tracing. In: Cryptographers' Track at the RSA Conference. pp. 399–421. Springer (2021)
7. Ayalon, O., et al.: Exploring privacy and incentives considerations in adoption of covid-19 contact tracing apps. In: USENIX Security Symposium (2023)
8. Benhamouda, F., Raykova, M., Seth, K.: Anonymous counting tokens. In: ASIACRYPT (2023)
9. Beskorovajnov, W., Dörre, F., Hartung, G., Koch, A., Müller-Quade, J., Strufe, T.: Contra corona: Contact tracing against the coronavirus by bridging the centralized–decentralized divide for stronger privacy. In: ASIACRYPT (2021)
10. California Health Care Foundation: https://www.chcf.org/blog/preliminary-research-suggests-covid-19-warning-app-slowed-transmission-virus/
11. Canetti, R., Kalai, Y.T., Lysyanskaya, A., Rivest, R.L., Shamir, A., Shen, E., Trachtenberg, A., Varia, M., Weitzner, D.J.: Privacy-preserving automated exposure notification. Cryptology ePrint Archive (2020)
12. Cevik, M., Kuppalli, K., Kindrachuk, J., Peiris, M.: Virology, transmission, and pathogenesis of sars-cov-2. bmj **371** (2020)
13. Chan, E.Y., Saqib, N.U.: Privacy concerns can explain unwillingness to download and use contact tracing apps when covid-19 concerns are high. Computers in Human Behavior **119**, 106718 (2021)
14. Cicala, F., et al.: Pure: A framework for analyzing proximity-based contact tracing protocols. ACM Computing Surveys (CSUR) (2021)
15. Cramer, R.: Modular design of secure yet practical cryptographic protocols. Ph. D.-thesis, CWI and U. of Amsterdam **2** (1996)

16. Cramer, R., et al.: Proofs of partial knowledge and simplified design of witness hiding protocols. In: Annual International Cryptology Conference (1994)
17. Dodis, Y., Yampolskiy, A.: A verifiable random function with short proofs and keys. In: Public Key Cryptography-PKC (2005)
18. DP-3T GitHub: Privacy and security risk evaluation of digital proximity tracing systems. `https://github.com/DP-3T/documents/blob/master/` (2024)
19. Fan, B., Andersen, D.G., Kaminsky, M., Mitzenmacher, M.D.: Cuckoo filter: Practically better than bloom. In: Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies. pp. 75–88 (2014)
20. Fuchsbauer, G., Gay, R.: Weakly secure equivalence-class signatures from standard assumptions. In: Public-Key Cryptography–PKC (2018)
21. Fuchsbauer, G., et al.: Structure-preserving signatures on equivalence classes and constant-size anonymous credentials. Journal of Cryptology (2019)
22. Génois, M., Barrat, A.: Can co-location be used as a proxy for face-to-face contacts? EPJ Data Science **7**(1), 1–18 (2018)
23. Givehchian, H., et al.: Evaluating physical-layer ble location tracking attacks on mobile devices. In: IEEE Symposium on Security and Privacy (SP) (2022)
24. Givehchian, H., et al.: Practical obfuscation of ble physical-layer fingerprints on mobile devices. In: IEEE Symposium on Security and Privacy (SP) (2023)
25. Griffy, S., Lysyanskaya, A.: Pacific: Privacy-preserving automated contact tracing scheme featuring integrity against cloning. Cryptology ePrint Archive (2023)
26. Günther, C.U., Pietrzak, K.: Deniability in automated contact tracing: Impossibilities and possibilities. Proceedings on Privacy Enhancing Technologies (2024)
27. Hassandoust, F., et al.: Individuals' privacy concerns and adoption of contact tracing mobile applications in a pandemic: A situational privacy calculus perspective. Journal of the American Medical Informatics Association **28**(3), 463–471 (2021)
28. Johansson, M.A., et al.: Sars-cov-2 transmission from people without covid-19 symptoms. JAMA network open **4**(1), e2035057–e2035057 (2021)
29. Kales, D., Rechberger, C., Schneider, T., Senker, M., Weinert, C.: Mobile private contact discovery at scale. In: 28th USENIX Security Symposium (2019)
30. Kaptchuk, G., et al.: How good is good enough? quantifying the impact of benefits, accuracy, and privacy on willingness to adopt covid-19 decision aids. Digital Threats: Research and Practice (DTRAP) **3**(3), 1–18 (2022)
31. Kate, A., Huang, Y., Goldberg, I.: Distributed key generation in the wild. Cryptology ePrint Archive (2012)
32. Morio, K., Esiyok, I., Jackson, D., Künnemann, R.: Automated security analysis of exposure notification systems. In: 32nd USENIX Security Symposium (2023)
33. Naor, M., Pinkas, B., Reingold, O.: Distributed pseudo-random functions and kdcs. In: Stern, J. (ed.) Advances in Cryptology - EUROCRYPT (1999)
34. Silde, T., Strand, M.: Anonymous tokens with public metadata and applications to private contact tracing. In: International Conference on Financial Cryptography and Data Security. pp. 179–199. Springer (2022)
35. Tedeschi, P., Bakiras, S., Di Pietro, R.: Spreadmenot: A provably secure and privacy-preserving contact tracing protocol. IEEE Transactions on Dependable and Secure Computing **20**(3), 2500–2515 (2022)
36. Troncoso, C., et al.: Decentralized privacy-preserving proximity tracing. arXiv preprint arXiv:2005.12273 (2020)
37. Troncoso, C., et al.: Deploying decentralized, privacy-preserving proximity tracing. Communications of the ACM (2022)