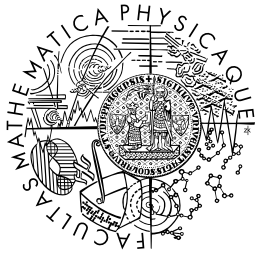

FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

BACHELOR THESIS

Martin Pastyřík

Security of cryptographic schemes for contact tracing

Computer Science Institute of Charles University

Supervisor of the bachelor thesis: Mgr. Pavel Hubáček, Ph.D.

Study programme: Mathematics (B1101)

Study branch: Mathematics for Information Technologies (MMIT)

Prague 2021

I declare that I carried out this bachelor thesis independently, and only with the cited sources, literature and other professional sources. It has not been used to obtain another or the same degree.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In Prague date 27.5.2021 .....

Author's signature

I would like to thank my supervisor, Mgr. Pavel Hubáček, Ph.D., for all the time and energy he put into helping me with this thesis.

Title: Security of cryptographic schemes for contact tracing

Author: Martin Pastyřík

Institute: Computer Science Institute of Charles University

Supervisor: Mgr. Pavel Hubáček, Ph.D., Computer Science Institute of Charles University

Abstract: Due to the Covid-19 pandemic in 2020 there was a big development of contact tracing schemes and applications. In this thesis, we describe the DP3T scheme and some possible attacks against it mainly the replay and relay attacks. In order to resist these attacks, we formally define and construct Pietrzak's Delay-MAC (INDOCRYPT 2020). Using this construction and the definition of a DCT scheme by Danz et al. (IACR Cryptol. ePrint Arch. 2020: 1309), we formally define Pietrzak's (INDOCRYPT 2020) contact tracing scheme, which we call DP4T. Using the security model presented in (IACR Cryptol. ePrint Arch. 2020: 1309), we prove that DP4T is resistant to replay attacks and discuss if the improvement of DP4T presented in (INDOCRYPT 2020) is resilient to relay attacks. Using definitions and properties from (IACR Cryptol. ePrint Arch. 2020: 1309) we discuss privacy of DP4T. We then present two new attacks on DCT schemes and prove that other schemes from literature are not resistant to them. We prove that DP4T is resilient against one of those attacks and discuss the importance of this result to the improvement of DP4T resistant to relay attacks.

Keywords: Contact tracing Covid-19 DP-3T Delay-MAC DP4T

Contents

Introduction

With the spreading pandemic of SARS-CoV-2 in 2020 emerged many societal, technological and other problems which need to be resolved as fast as possible. Among them occurred the need to track the people who were in close contact with an infected individual. It was soon discovered that large-scale contact tracing cannot be performed manually and, thus, mechanical help is needed in this field. And so it became an objective for computer scientists and programmers to design a system to track those people in a reliable and fast way.

However, they had to create a system, which reveals minimum personal information about its users. This had to remain true even if someone would attack the system or its history would be revealed. Moreover, this system had to be resistant to any attempt to misuse it to make its users falsely believe they were in a dangerous contact.

Many schemes were published to address this task. One of the first of them was DP3T [TPH+20]. Due to the rush in defeating the pandemic, it had many cryptographical issues as described, for example, by Vaudenay [Vau20]. On the other hand, it became an inspiration for many other schemes such as GAEN [LI20], a work of the union of the Google and Apple companies. This scheme is used nowadays in most of the tracing apps among Europe, including the Czech smartphone application eRouška [zČr20].

Our results

In this thesis, we start with describing the DP3T scheme and its problems, mainly the replay and relay attack. In chapter 2, we formally define Pietrzak's [Pie20] Delay-MAC, define its security properties and prove that Pietrzak's construction of Delay-MAC meets these properties. Our main contribution here is that we decouple the definition and the construction presented together in [Pie20].

In chapter 3, we use Delay-MAC to formally define Pietrzak's [Pie20] contact tracing scheme which we call DP4T. In section 3.1, using definitions and security model from [DDL+20], we prove that DP4T is resistant to replay attacks. Moreover, we discuss how to adapt our proof to show that a variant of DP4T suggested by Pietrzak [Pie20] is resilient also to relay attacks.

In section 3.2, we study privacy properties of DCT. First, we analyse resilience of DP4T against known attacks on privacy from [DDL+20]. Second, we introduce two new attacks on privacy in DCT. We show that DP4T is resilient to one of these attacks, while other two DCT schemes from the literature are not resilient to either of these attacks. Finally, we discuss why our results suggest that DP4T is a better alternative for a DCT scheme resilient against relay attacks.

1. Decentralized Contact Tracing

While constructing a contact tracing scheme, there are two alternative types of approach. The first one is to collect coordinates of everyone for all the time in one central server. After a user is tested positive, the server can easily compute his contacts and alert them. This approach is correct but it uses one central server with all the information. In other words, it completely gives up privacy of everyone to the server.

Another approach is to create a communication scheme that would allow its users to get the information that they have been in a dangerous contact with someone else revealing neither who was the infected one nor when this contact happened. This was the goal of Decentralized Privacy-Preserving Proximity Tracing (or DP3T for short) in [TPH+20]. This scheme did not turn out to be the safest, but it was one of the first solutions and many other schemes took inspiration from it. One of them is Pietrzak's scheme using Delayed Authentication, presented in [Pie20], which is the focus of this thesis. Another successor is GAEN 1.2, which is used in many contact tracing applications throughout Europe and the United States.

1.1 DP3T

In this section, we describe how DP3T works. Since Pietrzak's work was determined to improve this scheme, it would be best to start with this description.

The protocol has four phases: joining, broadcasting, sharing and checking. There are three participating parties in this scheme: users, a trusted server and a health authority.

Joining: When a user joins the scheme, he randomly generates his initial key sk_0.

Broadcasting: Broadcasting is the normal phase of DP3T, which happens for the majority of the time, i.e., until someone tests positive for the disease.

Every user generates a list of ephemeral identifiers, which we call *nym*s, as follows:

$$nym_1 \| \ldots \| nym_n := \mathsf{PRG}(\mathsf{PRF}(sk_d, \textit{"broadcast key"})),$$

where PRF is a pseudorandom function, PRG is a pseudorandom generator and sk_d is a private key for the day d. This key is computed from the key for day $d-1$ using a hash function $H(sk_{d-1}) = sk_d$.

The *nym*s are then broadcasted by the user in a random order via a BLE (Bluetooth Low Energy) beacon. Moreover, every user stores any *nym* he receives from other broadcasting users alongside with an identifier of the current day. This storing process may differ over implementations, e.g., it can be suitable to check if the broadcasting beacon was close enough and the contact lasted for long enough time.

Sharing: Once a user is confirmed positive by the health authority, he reveals his $sk_{d-\Delta}$ to the central server, where Δ is the time period when person can be asymptotic but infectious. The key is then stored at the server for anybody to download. Note that, in this phase, we need the health authority to confirm that the person is in fact positive and, thus, should alert his contacts. Otherwise, anybody could call himself sick and create false alerts.

Checking: When a user downloads a key sk, he can compute all the keys derived form this key and generate all the *nym*s that the original owner of the key was broadcasting. Then, the user can compare the generated *nym*s with the ones he stored during the broadcasting phase. If there is any intersection between these two sets the user can assume that he was in a close contact with an infectious user and, thus, undergo some preventive procedures like testing or quarantine.

1.2 Attacks on DCT Schemes

We can see that based on communication between two people, someone can negatively affect someone else's life. Indeed, attacks on DP3T were developed very quickly. See, e.g., Vaudenay [Vau20], who described more than ten attacks only few weeks after the publication of DP3T. We describe some of the attacks described in [Vau20] below.

1.2.1 Attacks on Integrity

The first category of attacks on DP3T is the more obvious one. In these attacks an adversary is trying to make his victim believe that the victim was in a close contact with an infectious user, even though it is not true. This is obviously a problem because it could lead to an unnecessary quarantine, lost of trust in the tracing application or even to an intentional overload of testing capacities.

Replay Attack: In the scenario of replay attack, there are three participants. An adversary $\mathcal{A}$, an honest user $\mathcal{U}$, who will be tested positive in a few days, and an honest victim $\mathcal{V}$.

The attack works as follows. $\mathcal{A}$ listens close to $\mathcal{U}$ and eventually receives a $nym_{\mathcal{U}}$ which $\mathcal{U}$ is broadcasting. $\mathcal{A}$ then travels to his victim $\mathcal{V}$ and broadcasts the $nym_{\mathcal{U}}$ next to $\mathcal{V}$. Then $\mathcal{V}$ probably receives it.

Now, after a few days, $\mathcal{U}$ is tested positive. He therefore uploads his key from Δ days back to the server. $\mathcal{V}$, as an honest user, downloads the key and generates all the *nym*s which $\mathcal{U}$ has been broadcasting over the last Δ days. Among them, he generates the $nym_{\mathcal{U}}$ he received from $\mathcal{A}$. This triggers an alert in the application and makes $\mathcal{V}$ proceed as if he was in a close contact with an infected user, which he was not. (Or at least as far as we know.)

Relay Attack: The relay attack does not differ much from the replay attack. In this scenario we use the same three participants as in the previous attack.

Again, $\mathcal{A}$ listens close to $\mathcal{U}$ and eventually receives a $nym_{\mathcal{U}}$ that $\mathcal{U}$ is broadcasting. But this time $\mathcal{A}$ immediately transmits the $nym_{\mathcal{U}}$ to his accomplice $\mathcal{A}'$, who is already close to $\mathcal{V}$. $\mathcal{A}'$ can immediately broadcast the $nym_{\mathcal{U}}$ and $\mathcal{V}$ probably receives and stores it.

Again, after a few days $\mathcal{U}$ is tested positive and he uploads his key. $\mathcal{V}$ downloads this key and generates all the *nym*s $\mathcal{U}$ was broadcasting. $\mathcal{V}$ finds that one of these *nym*s is among the *nym*s, which he has stored and proceeds as if he has been exposed to the infection.

The main difference between these two attacks is that in the relay attack the transmission is performed almost immediately. This turned out to be a big problem for many schemes resistant to the replay attack. This is because in order to resist replay attack the schemes check the time of contact in some way. If the time of receiving a *nym* differs from the time of sending the *nym*, the scheme rejects the *nym* as an attempt to perform a replay attack. But since relay attack happens almost immediately, this approach is not effective against it.

Released Case Attack: In this scenario there is an adversary $\mathcal{A}$ and an honest victim $\mathcal{V}$. $\mathcal{A}$ leverages a positive honest user $\mathcal{U}$, but this time, $\mathcal{A}$ does not approach $\mathcal{U}$ directly.

In the scenario of the released case attack, $\mathcal{A}$ listens to the server and tries to download the key of any infected user as soon as possible. In particular, before $\mathcal{V}$ does. $\mathcal{A}$ then generates a $nym_{\mathcal{U}}$ for the current day, approaches $\mathcal{V}$, and sends him the $nym_{\mathcal{U}}$.

After that, $\mathcal{V}$ downloads the key, generates all the *nym*s and finds out that one of them is the nym_U, which he received. This again leads to security measures for the victim, even though he has never been in contact with $\mathcal{U}$. (Or at least as far as we know.)

Pseudonym Forging Attack: If $\mathcal{A}$ somehow finds out, how $\mathcal{U}$ generates his *nym*s, or more specifically, has a reliable way to predict a next one, $\mathcal{A}$ can generate a $nym_{\mathcal{U}}$ and then send it to his victim $\mathcal{V}$.

Then $nym_{\mathcal{U}}$ is actually a *nym* broadcasted by $\mathcal{U}$. When $\mathcal{U}$ becomes positive, he reveals his keys. $\mathcal{V}$ then downloads them and finds out that one of the *nym*s broadcasted by $\mathcal{U}$ was the $nym_{\mathcal{U}}$ which $\mathcal{V}$ received.

We note that this attack is the only one presented in this section that cannot be successfully used to attack DP3T.

1.2.2 Privacy Disclosing Attacks

The second category of attacks we discuss are the attacks whose goal is to exploit the application to find out some non-trivial information about its users. Such information may contain identity, social groups, visited places or personal time schedules.

This type of attacks is not a big problem for DP3T since it stores minimum personal information about its users. On the other hand, it becomes an issue for

more complex schemes resistant to most of the attacks from the first category, which in exchange, store more exploitable data.

Linking Attack: In this scenario, an adversary is trying to decide, which *nym*s came from the same user. As an example, we can imagine linking a nym_0, whose author we know, to another nym_1, whose author we want to reveal. In DP3T, this is trivial once the author of nym_1 becomes positive and reveals his key.

2. Delay-MAC

To overcome the potential attacks described in chapter 1, Pietrzak [Pie20] introduced an idea of delayed authentication in the form of Delay-MAC. In this chapter, we provide its formal definition, define its basic security properties and prove that Pietrzak's construction of Delay-MAC satisfies these properties.

2.1 Definition

We define Delay-MAC by extending the standard definition of MAC from [KL15]:

Definition 1. A **Message Authentication Code** (or MAC) consists of three polynomial-time algorithms ($\mathsf{MAC.Gen}, \mathsf{MAC.Tag}, \mathsf{MAC.Vrf}$) such that:

- The *key-generation algorithm* $\mathsf{MAC.Gen}$ takes as input the security parameter 1^n and outputs a key k with $|k| \geq n$.
- The *tag-generation algorithm* $\mathsf{MAC.Tag}$ takes as input a key k and a message $m \in \{0,1\}^*$, and outputs a tag t. Since this algorithm may be randomized, we write this as $t \leftarrow \mathsf{MAC.Tag}_k(m)$.
- The deterministic *verification algorithm* $\mathsf{MAC.Vrf}$ takes as input a key k, a message m, and a tag t. It outputs a bit b, with $b = 1$ meaning *valid* and $b = 0$ meaning *invalid*. We write this as $b = \mathsf{MAC.Vrf}_k(m, t)$.

It is required that for every security parameter n, every key k output by $\mathsf{MAC.Gen}(1^n)$, and every $m \in \{0,1\}^*$, it holds that $\mathsf{MAC.Vrf}_k(m, \mathsf{MAC.Tag}_k(m)) = 1$.

If there is a function l such that for every k output by $\mathsf{MAC.Gen}(1^n)$, algorithm $\mathsf{MAC.Tag}_k$ is only defined for messages $m \in \{0,1\}^{l(n)}$, then we call the scheme a *fixed-length MAC for messages of length* $l(n)$.

Pietrzak's idea was to preserve the idea of verifying by a key, but not immediately upon receiving the tag. In the meantime between receiving and verification he wanted to store the tag in a way which would make it almost impossible to find out, what was the message. Moreover, this property must hold even if the set of possible messages is rather small.

We now provide the formal definition of Delay-MAC.

Definition 2. A **Delayed Message Authentication Code (or Delay-MAC)** consists of four polynomial-time algorithms ($\mathsf{DMAC.Gen}$, $\mathsf{DMAC.Send}$, $\mathsf{DMAC.Receive}$, $\mathsf{DMAC.Vrf}$) such that:

- On input a security parameter 1^n, the probabilistic *key generating algorithm* $\mathsf{DMAC.Gen}$ outputs a key k with $|k| \geq n$.
- On input a key k and a message $m \in \{0,1\}^*$, the probabilistic *tag generating algorithm* $\mathsf{DMAC.Send}$ outputs a tag t. We write this as $t \leftarrow \mathsf{DMAC.Send}_k(m)$.
- On input a message m and a tag t, the deterministic *storing algorithm* $\mathsf{DMAC.Receive}$ outputs a stored value st. We write this as $st = \mathsf{DMAC.Receive}(m, t)$

Experiment $\mathrm{Exp}^{\text{MAC-forge}}_{\mathcal{A},\mathsf{MAC}}(n)$:
$k \leftarrow \mathsf{MAC.Gen}(1^n)$, $\mathcal{Q} = \emptyset$
$(m,t) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{MAC.Tag}_k}}(1^n)$
Return $\begin{cases} 1 \text{ if } \mathsf{MAC.Vrf}_k(m,t) = 1 \wedge m \notin \mathcal{Q} \\ 0 \text{ otherwise} \end{cases}$

$\mathcal{O}_{\mathsf{MAC.Tag}_k}(m)$
$\mathcal{Q} = \mathcal{Q} \cup \{m\}$
$t \leftarrow \mathsf{MAC.Tag}_k(m)$
Return t

Figure 2.1: MAC-forge Experiment

- On input a key k and a stored value st, the deterministic *verification algorithm* $\mathsf{DMAC.Vrf}$ outputs a bit b, with $b = 1$ meaning *valid* and $b = 0$ meaning *invalid*. We write this as $b = \mathsf{DMAC.Vrf}_k(st)$.

The most basic property of Delay-MAC is correctness, as defined here:

Definition 3. A Delay-MAC $\mathsf{DMAC} = (\mathsf{DMAC.Gen}, \mathsf{DMAC.Send}, \mathsf{DMAC.Receive}, \mathsf{DMAC.Vrf})$ is **correct** if for every security parameter n, every key k output by $\mathsf{DMAC.Gen}(1^n)$, and every message $m \in \{0,1\}^*$, it holds that:

$$\mathsf{DMAC.Vrf}_k\Big(\mathsf{DMAC.Receive}\big(m, \mathsf{DMAC.Send}_k(m)\big)\Big) = 1.$$

2.2 Integrity

In this section, we define the requirements for a Delay-MAC to be secure, and prove that Pietrzak's construction of Delay-MAC satisfies these conditions.

For a standard MAC to be secure, we require it to be resistant to forging. In other words it must be infeasible for any probabilistic polynomial-time adversary to succeed in the forging experiment described in Figure 2.1. The experiment and the following definition were both presented in [KL15].

Definition 4. A message authentication code $\mathsf{MAC} = (\mathsf{MAC.Gen}, \mathsf{MAC.Tag}, \mathsf{MAC.Vrf})$ is existentially unforgeable under an adaptive chosen-message attack, or just **secure**, if for all probabilistic polynomial-time adversaries $\mathcal{A}$, there is a negligible function $negl$ such that:

$$\Pr[\mathrm{Exp}^{\text{MAC-forge}}_{\mathcal{A},\mathsf{MAC}}(n) = 1] \leq negl(n).$$

The notion of a secure Delay-MAC is analogous to a classic MAC. We allow the adversary to communicate with an oracle $\mathsf{DMAC.Send}_k(\cdot)$, which produces a correct Delay-MAC-tag for any message m. We also store every message the adversary asked for. We consider it a break of the scheme if the adversary is able to output any message m together with a tag t such that:

1. t is a valid tag for the message m (i.e., $\mathsf{DMAC.Vrf}_k(\mathsf{DMAC.Receive}(m,t)) = 1$)

   and

2. the adversary had not previously requested the oracle to tag the message m.

A Delay-MAC satisfying the level of security specified above is said to be unforgeable. Again, we start with the formal definition of the security experiment, which is presented in Figure 2.2.

Now, we define what it means for a Delay-MAC to be secure:

Experiment $\mathbf{Exp}^{\text{DMAC-forge}}_{\mathcal{A},\mathsf{DMAC}}(n)$:
$k \leftarrow_{\mathsf{DMAC.Gen}} (1^n), Q = \emptyset$
$(m,t) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{DMAC.Send}_k}}(1^n)$
$rec \leftarrow \mathsf{DMAC.Receive}(m,t)$
Return $\begin{cases} 1 \text{ if } \mathsf{DMAC.Vrf}_k(rec) = 1 \ \wedge \ m \notin Q \\ 0 \text{ otherwise} \end{cases}$

$\mathcal{O}_{\mathsf{DMAC.Send}_k}(m)$
$Q = Q \cup \{m\}$
$t \leftarrow \mathsf{DMAC.Send}_k(m)$
Return t

Figure 2.2: DMAC-forge Experiment

Definition 5. A Delay-MAC $\mathsf{DMAC} = (\mathsf{DMAC.Gen}, \mathsf{DMAC.Send}, \mathsf{DMAC.Receive}, \mathsf{DMAC.Vrf})$ is existentially unforgeable under an adaptive chosen-message attack, or just **secure**, if for all probabilistic polynomial-time adversaries $\mathcal{A}$, there is a negligible function *negl* such that:

$$\Pr[\mathrm{Exp}^{\text{DMAC-forge}}_{\mathcal{A},\mathsf{DMAC}}(n) = 1] \leq negl(n).$$

2.2.1 Integrity of Pietrzak's Construction

Pietrzak constructed a secure Delay-MAC from any (weakly) computationally binding commitment scheme and a standard secure MAC.

We provided the definition of a secure MAC at the beggining of this chapter. We provide the classical definition of a commitment scheme from [KL15] now.

Definition 6. A (non-iterative) **commitment scheme** is defined by two algorithms $(\mathsf{comGen}, \mathsf{commit})$ such that:

- On input security parameter 1^n, the randomized algorithm comGen outputs public parameters *par*.
- On input parameters *par* and a message $m \in \{0,1\}^n$, the probabilistic algorithm commit outputs a commitment *com*. If we make the randomness used by commit explicit, we denote it ρ and write it as $com = \mathsf{commit}(par, m; \rho)$.

We use the classical definition of the computationally binding property which is a part of the definition of a secure commitment scheme in [KL15] and use the commitment binding experiment displayed in Figure 2.3 also from [KL15].

Definition 7. A commitment scheme $\mathsf{CS} = (\mathsf{comGen}, \mathsf{commit})$ is **computationally binding** if for all probabilistic polynomial-time adversaries $\mathcal{A}$ there is a negligible function *negl* such that:

$$\Pr[\mathrm{Exp}^{\text{Binding}}_{\mathcal{A},\mathsf{CS}}(n) = 1] \leq negl(n).$$

Pietrzak's construction of Delay-MAC is formally described in Figure 2.4.

The correctness of Pietrzak's construction of Delay-MAC is evident. The next theorem proves that the construction is secure. Pietrzak [Pie20] provides a similar statement and a proof sketch in his work. Here, we expand on his proof sketch using the formal definition of Delay-MAC introduced above (Definition 2). In the proof, we use the standard union bound:

Experiment $\mathbf{Exp}^{\mathbf{Binding}}_{\mathcal{A},\mathsf{CS}}(n)$:
$par \leftarrow \mathsf{comGen}(1^n)$
$(com, m, r, m', r') \leftarrow \mathcal{A}(par)$
Return $\begin{cases} 1 \text{ if } m \neq m' \text{ and } \mathsf{commit}(par, m; r) = com = \mathsf{commit}(par, m'; r') \\ 0 \text{ otherwise} \end{cases}$

Figure 2.3: Binding Experiment

Using a MAC $\mathsf{MAC} = (\mathsf{MAC.Gen}, \mathsf{MAC.Tag}, \mathsf{MAC.Vrf})$ and a commitment scheme $\mathsf{CS} = (\mathsf{comGen}, \mathsf{commit})$. Let par output by $\mathsf{comGen}(1^n)$ be publicly known parameters of the commitment scheme.

DMAC.Gen: The DMAC.Gen is constructed as the MAC.Gen algorithm from the MAC function, i.e., on input security parameter n:

$$\mathsf{DMAC.Gen}(1^n) := \mathsf{MAC.Gen}(1^n).$$

DMAC.Send: On input a message m, the DMAC.Send algorithm samples a random coins ρ for commit. The algorithm is then constructed as a combination of commit and MAC. It produces a pair consisting of the MAC-tag $t = \mathsf{MAC}_k(\mathsf{commit}(par, m; \rho))$ and the random coins ρ :

$$\mathsf{DMAC.Send}_k(m) := \Big(\mathsf{MAC}_k\big(\mathsf{commit}(par, m; \rho)\big),\ \rho\Big) = (t, \rho) = tag,$$
$$\text{where } \rho \leftarrow_{\$} \mathcal{R}$$

DMAC.Receive: The DMAC.Receive algorithm takes on input a message m and a tag pair $tag = (t, \rho)$ and outputs a pair of the received t and the commitment $c = \mathsf{commit}(par, m; \rho)$:

$$\mathsf{DMAC.Receive}\big(m, (t, \rho)\big) := \big(t, \mathsf{commit}(par, m; \rho)\big) = (t, c) = st.$$

DMAC.Vrf: The $\mathsf{DMAC.Vrf}_k$ algorithm takes as input a pair $st = (t, c)$ and outputs 1 if and only if $\mathsf{MAC.Vrf}_k(c, t) = 1$:

$$\mathsf{DMAC.Vrf}_k(st) = 1 \quad \Leftrightarrow \quad \mathsf{MAC.Vrf}_k(c, t) = 1.$$

Figure 2.4: Pietrzak's construction of Delay-MAC

Proposition 1 (Union Bound)**.** *Let $A_1, A_2, \ldots, A_n$ be a set of events. Then*

$$\Pr[\bigcup_i A_i] \leq \sum_i \Pr[A_i].$$

Theorem 2. *If the commitment scheme is computationally binding and the MAC is secure then Pietrzak's construction of Delay-MAC defined in Figure 2.4 is secure.*

Proof. Let par output by $\mathsf{comGen}(1^n)$ be publicly known parameters of the commitment scheme.

Suppose to the contrary that there is an adversary $\mathcal{A}$ who can succeed in the Delay-MAC forge experiment with a non-negligible probability. We use him to break either the unforgeability of the MAC or the binding property of the commitment scheme.

We construct an adversary $\mathcal{B}$, who uses $\mathcal{A}$ as his subprogram, simulates for $\mathcal{A}$ the $\mathsf{DMAC.Send}_k(\cdot)$ oracle and is a successful adversary in the $\mathrm{Exp}^{\text{MAC-forge}}_{\mathcal{B},\mathsf{MAC}}$ experiment or the $\mathrm{Exp}^{\text{Binding}}_{\mathcal{B},\mathsf{CS}}$ experiment. The communication then proceeds as follows.

$\mathcal{A}$ sends a message m to $\mathcal{B}$. $\mathcal{B}$ generates a random coins ρ and computes $c = \mathsf{commit}(par, m; \rho)$. Then, $\mathcal{B}$ sends this c to his $\mathsf{MAC.Tag}_k$ oracle. The oracle returns a tag $t = \mathsf{MAC.Tag}_k(c)$. This tag is then sent to $\mathcal{A}$ alongside with the random coins ρ as a pair (t, ρ).

Now, since $\mathcal{A}$ is a successful adversary, he eventually produces a tag-pair $tag^{\mathcal{A}} = (t^{\mathcal{A}}, \rho^{\mathcal{A}})$ and a message $m^{\mathcal{A}}$, such that:

$$\mathsf{DMAC.Vrf}_k(\mathsf{DMAC.Receive}(m^{\mathcal{A}}, (t^{\mathcal{A}}, \rho^{\mathcal{A}}))) = 1.$$

By the construction of $\mathsf{DMAC.Vrf}_k$ and $\mathsf{DMAC.Receive}$, it follows that:

$$t^{\mathcal{A}} = \mathsf{MAC.Tag}_k(\mathsf{commit}(par, m^{\mathcal{A}}; \rho^{\mathcal{A}})).$$

But this means that $\mathcal{B}$ has just produced a message $a = \mathsf{commit}(par, m^{\mathcal{A}}; \rho^{\mathcal{A}})$ with a tag $t^{\mathcal{A}}$ such that $\mathsf{MAC.Tag}_k(a) = t$. Now, there are two possibilities:

1. The value a has not been sent to the $\mathsf{MAC.Tag}_k$ oracle by $\mathcal{B}$ to be tagged. In that case $\mathcal{B}$ produced a new message a with its correct tag $t^{\mathcal{A}}$. In other words $\mathcal{B}$ is able to break the unforgeability of the MAC we used. This violates our assumption that the MAC is secure.

2. The value $a = \mathsf{commit}(par, m^{\mathcal{A}}; \rho^{\mathcal{A}})$ has been sent to the $\mathsf{MAC.Tag}$ oracle to be tagged. But since we know that $\mathcal{A}$ is a successful adversary, the message m could have not been sent to $\mathcal{B}$ to be Delay-MAC-tagged.

   Therefore, $\mathcal{B}$ is able to produce a message $b = \mathsf{commit}(par, m'; r)$, where m' is another message asked by $\mathcal{A}$ and r is a random number generated upon asking for m'. Therefore, $b = a \ \wedge \ m \neq m'$, which is a direct violation of the binding property of the commitment scheme.

In other words, we have just showed that

$$\Pr[\mathrm{Exp}^{\text{DMAC-forge}}_{\mathcal{A},\mathsf{DMAC}}(n) = 1] \leq \Pr[(\mathrm{Exp}^{\text{MAC-forge}}_{\mathcal{B},\mathsf{MAC}}(n) = 1) \ \cup \ (\mathrm{Exp}^{\text{Binding}}_{\mathcal{B},\mathsf{CS}}(n) = 1)].$$

Thus, by union bound (Proposition 1), it holds that

$$\Pr[\mathrm{Exp}^{\text{DMAC-forge}}_{\mathcal{A},\mathsf{DMAC}}(n) = 1] \leq \Pr[\mathrm{Exp}^{\text{MAC-forge}}_{\mathcal{B},\mathsf{MAC}}(n) = 1] + \Pr[\mathrm{Exp}^{\text{Binding}}_{\mathcal{B},\mathsf{CS}}(n) = 1].$$

Since we assume that $\mathcal{A}$ is a successful adversary, there is by definition a non-negligible function $f(n) < \Pr[\mathrm{Exp}^{\text{DMAC-forge}}_{\mathcal{A},\mathsf{DMAC}}(n) = 1]$. This means that one of the probabilities on the right side of the last inequation has to be at least $\frac{1}{2}$ of $\Pr[\mathrm{Exp}^{\text{DMAC-forge}}_{\mathcal{A},\mathsf{DMAC}}(n) = 1] \ > \ f(n)$. But since $\frac{1}{2}$ of a non-negligible function is still non-negligible, we get that there exists a non-negligible function $g(n)$ such that either

$$\Pr[\mathrm{Exp}^{\text{MAC-forge}}_{\mathcal{B},\mathsf{MAC}}(n) = 1] \geq g(n)$$

or

$$\Pr[\mathrm{Exp}^{\mathrm{Binding}}_{\mathcal{B},\mathsf{CS}}(n) = 1] \geq g(n),$$

which is a contradiction to at least one of our assumptions. Thus, Pietrzak's construction of Delay-MAC is secure.

□

2.3 Privacy

In this section we define the privacy property for a Delay-MAC and prove that Pietrzak's construction (Figure 2.4) meets these requirements.

To be able to define privacy, we need to define statistical distance. We use the classical definition of statistical distance from [CDN15].

Definition 8. Let X_0 and X_1 be two random variables defined on the same probability space and with common range D. We define the **statistical distance** between X_0 and X_1 as

$$\delta(X_0, X_1) := \frac{1}{2} \sum_{d \in D} \Big| \Pr[X_0 = d] - \Pr[X_1 = d] \Big|$$

We present the definition of the privacy property of a Delay-MAC, which is similar to the definition of the hiding property of a commitment scheme (Definition 10).

Definition 9. A Delay-MAC $\mathsf{DMAC} = (\mathsf{DMAC.Gen}, \mathsf{DMAC.Send}, \mathsf{DMAC.Receive}, \mathsf{DMAC.Vrf})$ with a message space $\mathcal{M}$ is **statistically private** if for all $m_0, m_1 \in \mathcal{M}$ and a security parameter n there is a negligible function $negl$ such that:

$$\delta(ST_0(n), ST_1(n)) \leq negl(n),$$

where $ST_b(n)$ is defined as:

1. $k \leftarrow \mathsf{DMAC.Gen}(1^n)$,
2. $st = \mathsf{DMAC.Receive}(m_b, \mathsf{DMAC.Send}_k(m_b))$,
3. **output** st.

2.3.1 Privacy of Pietrzak's Construction

Pietrzak [Pie20] constructed his Delay-MAC scheme as described in Figure 2.4. In this construction he uses a commitment scheme with the statistically hiding property.

We use the classical definition of statistical hiding presented as a part of the definition of a secure commitment scheme in [KL15]:

Definition 10. A commitment scheme $(\mathsf{comGen}, \mathsf{commit})$ is **statistically hiding** if for all $m_0, m_1 \in \mathcal{M}$, a security parameter n and a par output by $\mathsf{comGen}(1^n)$ there is a negligible function $negl$ such that

$$\delta(\mathsf{commit}(par, m_0), \mathsf{commit}(par, m_1)) \leq negl(n).$$

We prove that Pietrzak's construction is a private Delay-MAC if it uses a commitment scheme which has the hiding property. Similarly to Theorem 2, an analogous statement to Theorem 4 appears with a proof sketch in [Pie20]. To prove Theorem 4, we need to state Proposition 3 which was presented in [CDN15].

Proposition 3. *Let A be any algorithm, and let X_0, X_1 be any random variables with common range D. Then it holds that*

$$\delta(A(X_0), A(X_1)) \leq \delta(X_0, X_1).$$

Theorem 4. *If the commitment scheme is statistically hiding then Pietrzak's construction of Delay-MAC (Figure 2.4) is private.*

Proof. Let $n \in \mathbb{N}$. Let *par* be output by $\mathsf{comGen}(1^n)$.

For every $m_0, m_1 \in \mathcal{M}$ let $C_0 = \mathsf{commit}(par, m_0)$ and $C_1 = \mathsf{commit}(par, m_1)$ be two random variables. (Note that commit if a randomized algorithm.) Therefore, from the statistical hiding property of the commitment scheme we get that:

$$\delta(C_0, C_1) < negl(n)$$

for a negligible function $negl$. Let A be an algorithm as follows:

$A_n(c)$:
1. $k \leftarrow \mathsf{DMAC.Gen}(1^n)$,
2. $t \leftarrow \mathsf{MAC.Tag}_k(c)$,
3. Return (t, c).

The output of $ST_b(n)$ from Definition 9 is:

$$\mathsf{DMAC.Receive}(m_b, \mathsf{DMAC.Send}_k(m_b)) =$$

$$= (\mathsf{MAC.Tag}_{k_b}(\mathsf{commit}(par, m_b; \rho_b)),\ \mathsf{commit}(par, m_b; \rho_b))$$

for uniformly random ρ_b. By definition of C_b above, ST_b is the same distribution as $A_n(C_b)$. Thus $A_n(C_b) = ST_b(n)$.

By Proposition 3, we get that

$$\delta(A_n(C_0), A_n(C_1)) \leq \delta(C_0, C_1).$$

In conclusion, we get:

$$\delta(ST_0(n), ST_1(n)) = \delta(A_n(C_0), A_n(C_1)) \leq \delta(C_0, C_1) < negl(n).$$

Thus, by definition, Pietrzak's construction of Delay-MAC is private.

□

3. DP4T

In this chapter we formally define Pietrzak's [Pie20] improvement of DP3T which uses Delay-MAC to provide more security properties than classical DP3T. We call this scheme Decentralized Pietrzak's Privacy-Preserving Proximity Tracing (or DP4T for short).

We start with the general definition of a decentralized contact tracing scheme presented in [DDL+20]. This definition uses discrete time measured by days and epochs. A day is the time period when a single daily key is used. An epoch is a part of a day and denotes the time period during which a user broadcasts one *nym*.

Definition 11. A decentralized contact tracing scheme **DCT** is a tuple of algorithms (Init, Rotate, MsgGen, MsgRec, TraceGen, TraceTf, TraceVf) defined as follows:

Init$(1^n) \to k_1$. Outputs an initial day key k_1 and sets CL $\leftarrow \emptyset$.
Rotate$(k_d) \to k_{d+1}$. On input a day key k_d, outputs a new key k_{d+1}.
MsgGen$(k_d, e) \to msg$. Outputs a broadcasted message for key k_d and epoch e.
MsgRec$(\mathbf{CL}, d, e, msg) \to$ **CL'**. On input a contact list CL, the current time d, e and a message msg, outputs an updated contact list CL'.
TraceGen$(\mathbf{Keys}, d_{start}, e_{start}) \to t$. On input a set of day keys Keys $= (k_{d-\Delta}, ..., k_d)$, a starting day d_{start} and epoch e_{start}, outputs a tracing key t for the tracing period $(d_{start}, e_{start}), \ldots, (d, e)$. We assume that the algorithm always checks that $d-\Delta \leq d_{start} \leq d$ and $1 \leq e_{start} \leq e$.
TraceTf$(\mathbf{TL}, t) \to$ **TL'**. On input the current tracing list TL and a new tracing key t, outputs an updated list TL'.
TraceVf$(\mathbf{CL}, \mathbf{TL}) \to \{0, 1\}$. On input of a contact list CL and tracing list TL outputs a bit, where 1 indicates that a match was found.

Remark 1. We had to change the definition a little to describe our scheme better. These changes are solely aesthetic and the model would work exactly the same if we stuck to the original definition. Specifically, in the original definition from [DDL+20], there were algorithms NymGen and NymRec, instead of MsgGen and MsgRec. We wanted to avoid any abuse of notation and decided to use the term *nym* exclusively for the ephemeral identifier. This produced the need to denote any additional information sent along with the *nym*. Therefore, we call a message (or a *msg* for short) all the data any user sends, which contains a *nym* and any additional information.

DP3T and DP4T use a pseudorandom function and a pseudorandom generator. We use the classical definitions of these primitives from [KL15].

Definition 12. Let l be a polynomial and let G be a deterministic polynomial-time algorithm such that for any n and any input $s \in \{0,1\}^n$, the result $G(s)$ is a string of length $l(n)$. We say that G is a **pseudorandom generator** if the following conditions hold:

1. **(Expansion:)** For every n it holds that $l(n) > n$.

2. **(Pseudorandomness:)** For any PPT algorithm D, there is a negligible function $negl$ such that

$$\Big|\Pr[D(G(s)) = 1] - \Pr[D(r) = 1]\Big| \leq negl(n),$$

where the first probability is taken over uniform choice of $s \in \{0,1\}^n$ and the randomness of D, and the second probability is taken over uniform choice of $r \in \{0,1\}^{l(n)}$ and the randomness of D.

We call l the expansion factor of G.

Definition 13. Let $F : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$ be an efficient length-preserving, keyed function. F is a **pseudorandom function** if for all PPT distinguishers D, there is a negligible function $negl$ such that:

$$\Big|\Pr[D^{F_k(\cdot)}(1^n) = 1] - \Pr[D^{f(\cdot)}(1^n) = 1]\Big| \leq negl(n),$$

where the first probability is taken over uniform choice of $k \in \{0,1\}^n$ and the randomness of D, and the second probability is taken over uniform choice of $f \in \mathsf{Func}_n$ and the randomness of D.

Pietrzak's idea was to preserve the basic DP3T scheme and authenticate every time of a user sending a *nym*. This tag is then verified when another user compares the *nym*s he received and the ones he generated from TL. In Figure 3.1 we present Pietrzak's [Pie20] construction following the definition of DCT.

Remark 2. Pietrzak's scheme is slightly different than the one given in Figure 3.1. He uses a Delay-MAC to authenticate the actual time (i.e., measured in minutes) of sending the *msg* instead of the epoch number. But since the time is measured as (d, e) in our model, we authenticate this value instead of minutes.

Moreover, Pietrzak [Pie20] uses one more feature in his scheme, which is rounding times of communicating users to the same value. Particularly, along with *nym* and *tag* in a message between users he also adds the least significant bit of the sender's time. This bit is then used to round the time of the receiver to match in the least significant bit. Thus, it provides synchronization of time for all users whose time differs at most in one minute. This part is unnecessary in our model since we assume all the users to use the same time and the transitions to happen immediately.

Remark 3. Pietrzak proposes yet another scheme which is supposed to be resistant to relay attacks. This one uses the same approach with authenticating some additional information with an ephemeral key. But this time, the additional information is the time and the coordinates of the user. Again, together with *tag* and *nym* the sending user sends the least significant bits of both coordinates to allow the receiving user to accordingly round his values.

Protocol 1. Decentralized Pietrzak's Privacy-Preserving Proximity Tracing (DP4T):

using Delay-MAC $\mathsf{DMAC} = (\mathsf{DMAC.Gen}, \mathsf{DMAC.Send}, \mathsf{DMAC.Receive}, \mathsf{DMAC.Vrf})$, a hash function H, a pseudorandom function PRF and a pseudorandom generator PRG:

Init$(1^n, e_{\max})$:
- $k'_1 \leftarrow_R \{0,1\}^n$
- Choose a random permutation $\mathrm{MAP}_1 : \{1, \ldots, e_{\max}\} \to \{1, \ldots, e_{\max}\}$
- Return $k_1 = (k'_1, \mathrm{MAP}_1)$.

Rotate(k_d) with $k_d = (k'_d, \mathrm{MAP}_d)$:
- $k'_{d+1} \leftarrow H(k'_d)$
- Choose a random permutation $\mathrm{MAP}_{d+1} : \{1, \ldots, e_{\max}\} \to \{1, \ldots, e_{\max}\}$.
- Return $k_{d+1} = (k'_{d+1}, \mathrm{MAP}_{d+1})$.

MsgGen(k_d, e):
- $nym_{d,1} \| \ldots \| nym_{d,e_{max}} \leftarrow \mathsf{PRG}(\mathsf{PRF}(k'_d, \textit{"broadcast key"}))$
- $k^E_{d,1} \| \ldots \| k^E_{d,e_{max}} \leftarrow \mathsf{PRG}(\mathsf{PRF}(k'_d, \textit{"secret key"}))$
- $tag \leftarrow \mathsf{DMAC.Send}_{k^E_{d,\mathrm{MAP}_d(e)}}((d, e))$
- Return $msg = (nym_{d,\mathrm{MAP}_d(e)},\ tag)$

MsgRec(CL,d, e, msg), *for* $m = (nym, tag)$:
- $st \leftarrow \mathsf{DMAC.Receive}((d, e), tag)$
- Return $\mathrm{CL}' = \mathrm{CL} \cup (nym, st)$

TraceGen(Keys, d_s, e_s):
- Retrieve $k_{d_{start}} \in$ Keys with $k_{d_{start}} = (k'_{d_{start}}, \mathrm{MAP}_{d_{start}})$
- Return $t = (k'_{d_{start}}, d_{start}, d-1)$

TraceTf(TL,t):
- Return $\mathrm{TL}' = \mathrm{TL} \cup t$

TraceVf(CL,TL):
- For each (k'_{d_i}, d_i, d_{end}) in TL:
  - Set $d^* = d_i,\ k^*_{d^*} = k'_{d_i}$
  - While $d^* \leq d_{end}$ do:
    - $nym^*_{d^*,1} \| \ldots \| nym^*_{d^*,e_{max}} \leftarrow \mathsf{PRG}(\mathsf{PRF}(k^*_{d^*}, \textit{"broadcast key"}))$
    - $k^{*E}_{d^*,1} \| \ldots \| k^{*E}_{d^*,e_{max}} \leftarrow \mathsf{PRG}(\mathsf{PRF}(k^*_{d^*}, \textit{"secret key"}))$
    - Add $(nym^*_{d^*,e_j}, k^{*E}_{d^*,e_j})$ to Nyms for $e_j = 1,\ \ldots, e_{max}$
    - $k^*_{d^*+1} \leftarrow H(k^*_{d^*}),\ d^* = d^* + 1$
- Set $b = 0$
- For each $(nym, st) \in$ CL and each $(nym', k') \in$ Nyms
  - If $nym = nym'$ then:
    - If $\mathsf{DMAC.Vrf}_{k'}(st) = 1$ then:
      - ▷ Set $b = 1$
- Return b

Figure 3.1: Construction of DP4T

$\underline{\mathcal{O}_{\text{NewUser}}()}$
set $l \leftarrow l + 1$ and $\mathcal{U} \leftarrow \mathcal{U} \cup l$
store $k_d^l \leftarrow_R \text{Init}(1^n)$, $\text{CL}[l] = \emptyset$

$\underline{\mathcal{O}_{\text{GetMsg}}(u_S)}$
abort if $u_S \notin \mathcal{U}$
$msg \leftarrow \text{MsgGen}(k_d^u, e)$
$\mathcal{Q}_{\text{msg}} = \mathcal{Q}_{\text{msg}} \cup \{(u_S, \mathcal{A}, d, e, msg)\}$
return msg

$\underline{\mathcal{O}_{\text{RecMsg}}(u_R, msg)}$
abort if $u_R \notin \mathcal{U}$
$\text{CL}'[u_R] \leftarrow \text{NymRec}(\text{CL}[u_R], d, e, msg)$
$\mathcal{Q}_{\text{msg}} = \mathcal{Q}_{\text{msg}} \cup \{(\mathcal{A}, u_R, d, e, msg)\}$

$\underline{\mathcal{O}_{\text{SendMsg}}(u_S, u_R)}$
abort if u_S or $u_R \notin \mathcal{U}$
$msg \leftarrow \text{MsgGen}(k_d^{u_S}, e)$
$\text{CL}'[u_R] \leftarrow \text{MsgRec}(\text{CL}[u_R], d, e, msg)$
$\mathcal{Q}_{\text{msg}} = \mathcal{Q}_{\text{msg}} \cup \{(u_S, u_R, d, e, msg)\}$

$\underline{\mathcal{O}_{\text{Rotate}}(X)}$ with $X \in \{day, epoch\}$
if $X = epoch$ and $e < e_{max}$:
  set $e = e + 1$
if $X = day$:
  $\forall u \in \mathcal{U} : k_{d+1}^u \leftarrow \text{Rotate}(k_d^u)$
  Set $d = d + 1$ and $e = 1$

$\underline{\mathcal{O}_{\text{TraceGen}}(u, d_{start}, e_{start})}$
abort if $u \notin \mathcal{U}$, set $\mathcal{U} \leftarrow \mathcal{U} \backslash u$
$t \leftarrow \text{TraceGen}(\text{KEYS}^u, d_{start}, e_{start})$
$\text{TL}' \leftarrow \text{TraceTf}(\text{TL}, t)$
$\{(d_i, e_j)\} \leftarrow \text{Validity}(d, t)$
$\forall (d_i, e_j) : \mathcal{O}_{\text{pos}} = \mathcal{O}_{\text{pos}} \cup \{(u, d_i, e_j)\}$

$\underline{\mathcal{O}_{\text{UploadTrace}}(t)}$
$\text{TL}' \leftarrow \text{TraceTf}(\text{TL}, t)$
$\{(d_i, e_j)\} \leftarrow \text{Validity}(d, t)$
$\forall (d_i, e_j) : \mathcal{O}_{\text{pos}} = \mathcal{O}_{\text{pos}} \cup \{(\mathcal{A}, d_i, e_j)\}$

$\underline{\mathcal{O}_{\text{GetTL}}()}$
return TL

Figure 3.2: Oracles, adversary is given access to in $\text{Exp}_{\mathcal{A},\text{DCT}}^{\text{Integrity}}$

3.1 Integrity

In this section, we prove that Pietrzak's improvement of DP3T is beneficial and helps the scheme achieve a strong notion of integrity, i.e., resilience to all attacks captured by our security model except against the relay attacks. For the formal analysis, we use the security model presented in [DDL+20].

3.1.1 Security Model

Integrity is presented in a model which simulates the adversary's communication with honest users and communication among the users themselves.

The adversary plays a game where he can communicate with oracles described in Figure 3.2. He eventually outputs a user u^* who is considered the victim of the attack.

In the game, we measure time as $(d, e) \in \mathbb{N}^2$ where d denotes the current day and e denotes the current epoch. We set $(d, e) = (0, 0)$ at the beginning of the experiment. Time is united for every participant (i.e., the adversary and users) of the experiment and is only changed by the adversary by calling the $\mathcal{O}_{\text{Rotate}}$ oracle.

The game keeps lists $\mathcal{Q}_{\text{msg}}$ and $\mathcal{Q}_{\text{pos}}$, which are essential for the experiment. In $\mathcal{Q}_{\text{msg}}$, there are stored all the messages generated during the experiment alongside with their sender, receiver, and the time of this transition in a form of a pair (d, e) denoting the day and the epoch of this communication. Every encounter is

stored as one element. If a msg was sent to more participants, then it is stored individually every time.

The second list is $\mathcal{Q}_{\text{pos}}$ which stores information about who was marked as infectious at a certain time. More precisely, it stores the name of a user alongside with the time (d, e) for every epoch the user was considered infectious.

We describe what each of the oracles in Figure 3.2 does on high-level:

NewUser: The oracle $\mathcal{O}_{\text{NewUser}}()$ creates a new user for the game and generates him a key. It also adds the new user to the list of users, which we denote $\mathcal{U}$. We also denote l the current number of users.

GetMsg: The oracle $\mathcal{O}_{\text{GetMsg}}(u_S)$ generates the message msg, which is broadcasted by user u_S at the current epoch. The oracle also adds this msg with the information about current time to the list of sent messages $\mathcal{Q}_{\text{msg}}$ as if it was sent to the adversary.

RecMsg: The oracle $\mathcal{O}_{\text{RecMsg}}(u_R, msg)$ simulates the receiving process of a user u_R for the message msg. It also stores this message in the list $\mathcal{Q}_{msg}$ as it was sent by the adversary to u_R at current epoch.

SendMsg: The oracle $\mathcal{O}_{\text{SendMsg}}(u_S, u_R)$ simulates the contact of two users u_S and u_R. The user u_S generates a msg and u_R receives it. This contact is then registered in the $\mathcal{Q}_{\text{msg}}$ list.

Rotate: The oracle $\mathcal{O}_{\text{Rotate}}(X)$ increases time and, if needed, rotates the key of every user. The adversary can decide, if he wants to increase only the value of epoch or if he wants to start a new day.

TraceGen: The oracle $\mathcal{O}_{\text{TraceGen}}(u, d_{start}, e_{start})$ simulates the revelation of keys by a user who was tested positive. It generates his tracing information, transforms it as an honest server would do and adds the information about the user u being positive from time (d_{start}, e_{start}) to the list of positive users $\mathcal{Q}_{\text{pos}}$. The Validity function is there to choose the days when the keys in the tracing information t are considered valid.

UploadTrace: The oracle $\mathcal{O}_{\text{UploadTrace}}(t)$ simulates the adversary calling himself positive with tracing information t at the current time. Again, the Validity function is there to choose the days when the keys in t are considered valid.

GetTL: The oracle $\mathcal{O}_{\text{GetTL}}()$ simulates the adversary downloading the list of tracing information from the server.

The goal of the adversary is to make his victim u^* believe that there was a close contact between the victim and a positive user. The adversary has to do this without leveraging any trivial possibilities. This definition of integrity and the experiment described in Figure 3.3 were presented in [DDL+20].

Definition 14. A DCT scheme provides **strong** and **weak integrity** respectively if for all efficient adversaries $\mathcal{A}$ and a security parameter n there is a negligible function $negl$ such that

$$\Pr[\text{Exp}^{\text{Integrity}}_{\mathcal{A},\text{DCT}}(n) = 1] \leq negl(n).$$

Experiment $\mathbf{Exp}^{\mathbf{Integrity}}_{\mathcal{A},\mathbf{DCT}}(n)$:
$d \leftarrow 1, e \leftarrow 1, l \leftarrow 0, \mathrm{TL} \leftarrow \emptyset$
$u^* \leftarrow_R \mathcal{A}^{\mathcal{O}_{\mathrm{NewUser}},\mathcal{O}_{\mathrm{Rotate}},\mathcal{O}_{\mathrm{GetMsg}},\mathcal{O}_{\mathrm{RecMsg}},\mathcal{O}_{\mathrm{SendMsg}},\mathcal{O}_{\mathrm{TraceGen}},\mathcal{O}_{\mathrm{UploadTrace}},\mathcal{O}_{\mathrm{GetTL}}}(1^n)$
retrieve current tracing list TL, and the contact list $\mathrm{CL}[u^*]$
return 1 if $\mathrm{TraceVf}(\mathrm{CL}[u^*], \mathrm{TL}) = 1$ and
$\forall(\mathcal{P}, u^*, d_i, e_j, msg) \in \mathcal{Q}_{\mathrm{msg}}$ with $d_i \in [d - \lambda, d]$ it holds that:
1. if $\mathcal{P} = u_S : \nexists(u_S, d_i, e_j) \in \mathcal{Q}_{\mathrm{pos}}$
2. if $\mathcal{P} = \mathcal{A}$
   (a) if $\exists(u_S, \mathcal{A}, d_i, e_j, msg) \in \mathcal{Q}_{\mathrm{msg}} : \nexists(u_S, d_i, e_j) \in \mathcal{Q}_{\mathrm{pos}}$
   (b) if $\nexists(u_S, \mathcal{A}, d_i, e_j, msg) \in \mathcal{Q}_{\mathrm{msg}} : \nexists(\mathcal{A}, d_i, e_j) \in \mathcal{Q}_{\mathrm{pos}}$

Weak Integrity: condition 2a is relaxed to replay attacks by removing the epoch:
2.a*) if $\exists(u_S, \mathcal{A}, d_i, *, msg) \in \mathcal{Q}_{\mathrm{msg}} : \nexists(u_S, d_i, *) \in \mathcal{Q}_{\mathrm{pos}}$

Figure 3.3: Integrity Experiment

To make these conditions more understandable we describe them here:

Condition 1. ensures that the adversary cannot win by simply putting his victim in a close contact with a positive honest user. This constraint is needed because this type of communication is meant to happen in any DCT scheme.

Condition 2.a) forbids the adversary from performing a relay attack. The condition states that if there was a msg, sent by a user u_S to $\mathcal{A}$ at a time (d, e), and this msg was then forwarded by $\mathcal{A}$ to the victim u^* at the same time, the user u_S could not be considered positive at the time (d, e). In other words, If the adversary resends a msg immediately, it cannot be from a user, who is eventually tested positive for that time. This condition is presented in [DDL$^+$20], because all the schemes analyzed in there are not resistant to relay attacks.

Condition 2.b) forbids $\mathcal{A}$ from sending his victim a msg, which was generated by $\mathcal{A}$ from a key k, and then uploading this key to the tracing list. This type of attack would require $\mathcal{A}$ to be tested positive by the health authority. But we assume the health authority to be trusted. Therefore, the only option, how $\mathcal{A}$ can perform this attack is to actually become positive, which we cannot consider a successful attack.

Finally, condition 2.a*) forbids the adversary from performing not only a relay attack, but even a replay attack during one day. The idea is the same as for the condition 2.a). If $\mathcal{A}$ forwards a message msg from a user u_S to the victim u^* in one day, it cannot be from a user who eventually tested positive for that day. This condition is used in [DDL$^+$20] in the context of weaker schemes, in order to prove that these schemes are resistant to other attacks captured by their model *except relay and replay attacks.*

3.1.2 Integrity of DP4T

Before proving that Pietrzak's construction provides integrity, we need to present two more definitions securing the properties of the primitives used in the scheme. Both Definition 15 and Definition 16 were presented in [DDL$^+$20].

Definition 15. A pseudorandom function PRF is **key-preimage resistant**, if for all probabilistic polynomial adversaries $\mathcal{A}$ there is a negligible function *negl*

such that

$$\Pr[(k', x) = \mathcal{A}^{\mathsf{PRF}(k,\cdot)}(1^n) : \mathsf{PRF}(k, x) = \mathsf{PRF}(k', x)] \leq negl(n),$$

where the probability is over the randomness of $\mathcal{A}$ and $k \leftarrow \{0,1\}^n$.

Definition 16. A pseudorandom generator $\mathsf{PRG} : \{0,1\}^n \rightarrow \{0,1\}^{r \cdot l}$ is **partial preimage resistant**, if for all efficient adversaries $\mathcal{A}$ and any seed s uniformly randomly chosen from $\{0,1\}^n$ there is a negligible function $negl$ such that

$$\Pr[\mathsf{PRG}(s)[i] = \mathsf{PRG}(\mathcal{A}(1^n, \mathsf{PRG}(s)))[j] \text{ for some } i, j] \leq negl(n),$$

where $\mathsf{PRG}(x)[i]$ is the i-th block of the output of $\mathsf{PRG}(x)$ of length r.

Now, we can prove that DP4T provides strong integrity. The first part of the proof is analogous to the proof of weak integrity of the basic DP3T scheme in [DDL+20].

Theorem 5. *The DP4T scheme (defined in Figure 3.1) satisfies strong integrity if H is a random oracle, pseudorandom function* PRF *is key-preimage resistant, pseudorandom generator* PRG *is partial preimage resistant and* $\mathsf{DMAC} = (\mathsf{DMAC.Gen}, \mathsf{DMAC.Send}, \mathsf{DMAC.Receive}, \mathsf{DMAC.Vrf})$ *is secure.*

Proof. First, we prove that the basic DP3T algorithm provides weak integrity. Note that in DP3T a *msg* contains only a *nym*.

Suppose to the contrary that there exists a PPT adversary $\mathcal{A}$ who can succeed in the integrity experiment with a non-negligible probability. Therefore, $\text{TraceVf}(\text{CL}[u^*], \text{TL}) = 1$. In other words, the victim has received a *nym* which was then also generated from one of the keys in TL. Let us split this situation to two cases:

1. The victim has received the *nym* generated by an honest user. But by the condition 1) or 2.*a*) from Figure 3.3, this implies that the honest user was not positive at this time. In other words, his key k for the day of this contact is not in the list TL. But we know that the *nym* was generated from a key k' which is in the list TL.

   This means that the adversary or another honest user (accidentally) generated the key k' such that for a broadcast key BR either

   - $\mathsf{PRF}(k', BR) = \mathsf{PRF}(k, BR)$, which is negligible since the used PRF is key-preimage resistant.

     or

   - $\exists\, i, j \in [1, e_{max}]$ such that $y[i] = y'[j]$ for $y' \leftarrow \mathsf{PRG}(\mathsf{PRF}(k', BR))$ and $y \leftarrow \mathsf{PRG}(\mathsf{PRF}(k, BR))$, which we know is negligible since the used PRG is partial preimage resistant.

2. The *nym* was generated by an adversary $\mathcal{A}$. But from the condition 2.*b*) we know that the adversary was not considered positive at that time. In other words, any key k' computed by $\mathcal{A}$ could not be in the TL list. This means that $\mathcal{A}$ was able to predict an outcome of a PRG or a PRF which we know is negligible from the pseudorandomness of these oracles.

From the first part and the fact that the condition 2.a) from Figure 3.3 forbids $\mathcal{A}$ to perform a relay attack, we know that if the adversary was successful, he had to perform a replay attack within one day. In other words, he had to compute a tag $t^{\mathcal{A}}$ such that

$$\mathsf{DMAC.Vrf}_k(\mathsf{DMAC.Receive}((d, e^{\mathcal{A}}), t^{\mathcal{A}})) = 1$$

for some $(d, e^{\mathcal{A}}) \neq (d, e)$, where (d, e) is the epoch when nym was originally broadcasted by an honest user. But this means that $\mathcal{A}$ was able to produce a valid Delay-MAC-tag $t^{\mathcal{A}}$ for a message $(d, e^{\mathcal{A}})$ with no information about a key k, which we know is negligible because the Delay-MAC is secure.

□

Remark 4. We actually have to generate a special k^{*E}_{d,e_j} for each epoch to prevent the replay attack. Imagine, if we used only one key to MAC all the times in a day d. An adversary $\mathcal{A}$ could then collect two messages (nym_1, tag_1), (nym_2, tag_2) sent by an honest user $\mathcal{U}$ at different epochs (d, e_1), (d, e_2) respectively.

$\mathcal{A}$ could then send (nym_1, tag_2) at time (d, e_2) to his victim $\mathcal{V}$. Assuming $\mathcal{U}$ eventually becomes positive and his keys become revealed, this message would get verified by $\mathcal{V}$. Thus, the adversary succeeded in the replay attack.

This obviously requires $\mathcal{A}$ to be able to transmit the tag_2 at the time (d, e_2) to $\mathcal{V}$. With this ability we can probably anticipate him to be able to just perform the relay attack. To justify this attack, let us assume that the tracing application does not give its user just the information that he was in contact with a positive user, but explicitly states how many times. In this situation the described attack could easily make the victim believe that he was in a close contact with as much as n infectious people for n the number of epochs in a day.

Remark 5. As stated in Remark 3, Pietrzak [Pie20] presents yet another scheme. This scheme is even resistant to relay attacks. The proof would be exactly the same as for the Theorem 5. It would assume that the location of the contact with $\mathcal{U}$ was different from the location of the contact with $\mathcal{V}$ and thus $\mathcal{A}$ had to produce a valid Delay-MAC-tag for a new message.

However, even this scheme does not provide resistance against a relay attack performed on two users who are close to each other, but would never actually meet (e.g., they are in different rooms of a same building).

Experiment $\mathbf{Exp}^{\text{MsgUnlink}}_{\mathcal{A},\text{DCT}}(n)$:
$k_1^0 \leftarrow_R \text{Init}(1^n), k_1^1 \leftarrow_R \text{Init}(1^n), d = 1$
$(e^*, st) \leftarrow_R \mathcal{A}^{\mathcal{O}_{\text{Rotate}},\mathcal{O}_{\text{MsgGen}}}(1^n)$
Store current day as challenge day d^*.
$b \leftarrow_R \{0,1\}$
$msg_b \leftarrow \text{MsgGen}(k_{d^*}^b, e^*)$
$b^* \leftarrow_R \mathcal{A}^{\mathcal{O}_{\text{Rotate}},\mathcal{O}_{\text{MsgGen}},\mathcal{O}_{\text{TraceGen}}}(st, msg_b)$
Return $\begin{cases} 1 \text{ if } b^* = b \text{ and } (d^*, e^*) \notin Q. \\ 0 \text{ otherwise.} \end{cases}$

$\underline{\mathcal{O}_{\text{Rotate}}()}$
$k_{d+1}^0 \leftarrow_R \text{Rotate}(k_d^0)$
$k_{d+1}^1 \leftarrow_R \text{Rotate}(k_d^1)$
Set $d = d + 1$.

$\underline{\mathcal{O}_{\text{MsgGen}}(u, e)}$
$msg_{d,e}^u \leftarrow \text{MsgGen}(key_d^u, e)$
$Q = Q \cup \{(d, e)\}$
Return $msg_{d,e}^u$.

$\underline{\mathcal{O}_{\text{TraceGen}}(u, d_{\text{start}}, e_{\text{start}})}$
strong: abort if $(d_{\text{start}}, e_{\text{start}} < (d^*, e^* + 1))$
weak: abort if $(d_{\text{start}}, e_{\text{start}} < (d^* + 1, 1))$
$t^u \leftarrow \text{TraceGen}(\text{KEYS}^u, d_{\text{start}}, e_{\text{start}})$
Return t^u.

Figure 3.4: Message Unlinkability Experiment

3.2 Privacy

The second property we would like a DCT scheme to have is privacy. In other words, there is no way how to misuse the application to track somebody or to find out any other information about a contact beyond the fact that it happened. This is the biggest difference between a DCT scheme and a centralized scheme. In a centralized scheme, there is a central server which is intentionally constructed to know when and where every contact happened.

3.2.1 Message Unlinkability

The first property we consider is the message unlinkability presented in [DDL+20]. Note that in [DDL+20], they call this property pseudonym unlinkability. But as discussed in Remark 1, we call the sent information a message instead of a pseudonym.

The property requires that no PPT adversary is able to link two different messages (or pseudonyms in their notation) to one person during the broadcasting phase. The formal definition uses the experiment described in Figure 3.4 and was presented in [DDL+20].

Definition 17. A DCT scheme provides **strong** and **weak message unlinkability** respectively, if for all efficient PPT adversaries $\mathcal{A}$ there is a negligible function *negl* such that

$$\Pr[\text{Exp}^{\text{MsgUnlink}}_{\mathcal{A},\text{DCT}}(n) = 1] \leq 1/2 + negl(n).$$

DP3T has the weak message unlinkability property as shown by [DDL+20].

DP4T uses a message, containing a *nym* and a Delay-MAC-tag. As the *tag* is an output of an authentication scheme, it is possible that it leaves some information about its sender. Therefore, the *tag*s can, in principle, be used to link distinct messages of one user. However, this property depends on the MAC which is used to construct the Delay-MAC in DP4T. If this MAC is constructed, for example, as a HMAC by a hash function, it could result in DP4T having the weak message unlinkability property. We leave this as a question for future research.

DP4T cannot aim for strong message unlinkability since its *nym*s are still generated from a single key and they are broadcasted in a random order during the day. This implies that we have to let the verifying users generate the *nym*s

Experiment $\mathbf{Exp}^{\text{TraceUnlink}}_{\mathcal{A},\text{DCT}}(n)$:
$k_1^0 \leftarrow_R \text{Init}(1^n), k_1^1 \leftarrow_R \text{Init}(1^n), d = 1$
$(e^*, st) \leftarrow_R \mathcal{A}^{\mathcal{O}_{\text{Rotate}},\mathcal{O}_{\text{MsgGen}}}(1^n)$
store current day as challenge day d^*
$b \leftarrow_R \{0,1\}$
$msg_b \leftarrow \text{MsgGen}(k^b_{d^*}, e^*)$
$b^* \leftarrow_R \mathcal{A}^{\mathcal{O}_{\text{Rotate}},\mathcal{O}_{\text{MsgGen}},\mathcal{O}_{\text{TraceBoth}}}(st, msg_b)$
return 1 if $b^* = b$ and
Strong Unlinkability: $(d^*, e^*) \notin Q$
Weak Unlinkability: $(d^*, *) \notin Q$

$\underline{\mathcal{O}_{\text{TraceBoth}}(d^0_{\text{start}}, e^0_{\text{start}}, d^1_{\text{start}}, e^1_{\text{start}})}$
abort if $(d^u_{\text{start}}, e^u_{\text{start}}) > (d^*, e^*)$
$t^0 \leftarrow \text{TraceGen}(\text{KEYS}^0, d^0_{\text{start}}, e^0_{\text{start}})$
$t^1 \leftarrow \text{TraceGen}(\text{KEYS}^1, d^1_{\text{start}}, e^1_{\text{start}})$
$TL \leftarrow \text{TraceTf}(\text{TraceTf}(\emptyset, t^0), t^1)$
return TL

Figure 3.5: Trace Unlinkability Experiment

for every epoch in every day. Thus, any adversary can trivially link two *nym*s for different epochs generated form the same key, even though the original owner of this key was considered infectious only in one of those epochs.

Post-compromise security

[DDL+20] also defined the property of *post-compromise security*, which requires that a DCT scheme is message unlinkable even if one of its daily keys is compromised (i.e., revealed to an adversary). DP4T obviously does not fulfill this property since all its keys are generated from one starting key. Thus, leaking of the key for day d^* compromises the secrecy of all the keys for days $d \geq d^*$.

3.2.2 Trace Unlinkability

The second type of unlinkability presented in [DDL+20] is *trace unlinkability*. It requires that no PPT adversary can link distinct messages of a user after the keys, which were used to generate both these messages, were made public. The formal definition uses the experiment described in Figure 3.5 and was presented in [DDL+20].

Definition 18. A DCT scheme provides **strong** and **weak trace unlinkability** respectively, if for all efficient $\mathcal{A}$ there is a negligible function *negl* such that

$$\Pr[\text{Exp}^{\text{TraceUnlink}}_{\mathcal{A},\text{DCT}}(n) = 1] \leq 1/2 + negl(n).$$

Obviously, DP4T does not fulfill this definition, since all its *nym*s in one day are generated from one daily key. In particular, strong integrity (definition 14) of DP4T prevents trace unlinkability as we show below.

Theorem 6. *DP4T does not achieve trace unlinkability in any form if the pseudorandom generator* PRG *is partial preimage resistant and the pseudorandom function* PRF *is key-preimage resistant.*

Proof. Let us construct an adversary $\mathcal{A}$ who proves this theorem. For the first challenge we use an adversary as folows: $\mathcal{A}^{\mathcal{O}_{\text{Rotate}},\mathcal{O}_{\text{MsgGen}}}(1^n)$ constructed as follows:

1. Call $\mathcal{O}_{\text{Rotate}}()$ until $d \neq 1$.
2. Set $st = \{(0, msg^0)\}$, where $msg^0 \leftarrow \mathcal{O}_{\text{MsgGen}}(0, 1)$.

3. Set $st = st \cup \{(1, msg^1)\}$, where $msg^1 \leftarrow \mathcal{O}_{\text{MsgGen}}(1, 1)$.
4. Call $\mathcal{O}_{\text{Rotate}}()$.

Now, the current day d is set as d^* and a random bit b is generated. Then the challenge message msg_b is computed. Let us construct the next adversary as follows:

$\mathcal{A}^{\mathcal{O}_{\text{Rotate}}, \mathcal{O}_{\text{MsgGen}}, \mathcal{O}_{\text{TraceBoth}}}(st, msg_b)$:

1. $TL \leftarrow \mathcal{O}_{\text{TraceBoth}}(1, 1, 1, 1)$.
2. Obtain keys k^x and k^y form TL.
3. Extract the pseudonym nym^0 from msg^0.
4. Extract the pseudonym nym^1 from msg^1.
5. Generate $nym^x_{(1,1)} \| \ldots \| nym^x_{(2,e_{\max})}$ using k^x.
6. Generate $nym^y_{(1,1)} \| \ldots \| nym^y_{(2,e_{\max})}$ using k^y.
7. Find $nym^c_{(1,e)}$ such that $nym^c_{(1,e)} = nym^0$, set $c = 0$.
8. Find $nym^d_{(1,e)}$ such that $nym^d_{(1,e)} = nym^1$, set $d = 1$.
9. Find $nym^f_{(2,e)}$ such that $nym^f_{(1,e)} = nym_b$, set $b = f$.
10. **Return** b.

This adversary can distinguish b with the probability $(1-p)$, where p is the probability that for any $u \neq v$ or $d_1 \neq d_2$ or $e_1 \neq e_2$ holds that $nym^u_{(d_1,e_1)} = nym^v_{(d_2,e_2)}$, which we know is negligible since PRG has the partial preimage resistance property and PRF has the key-preimage resistance property.

□

3.2.3 Privacy Disclosure by a Dishonest Server Owner

The idea of DCT was to prevent a server owner from obtaining any non-trivial information about the users of a scheme. In this section, we formalize two new attacks on DCT schemes, which show that a server owner can, in fact, use a DCT scheme to acquire some information. Note that we assume a server owner who is capable of obtaining (e.g., by using police or secret service) any user's history of contacts stored by the scheme.

Contact Identity Disclosure by a Server Owner

The first attack we present is the identity leaking attack. In this attack, a dishonest server owner chooses an honest user $\mathcal{V}$ and tries to reveal the identity of the users $\mathcal{V}$ was in contact with.

Note that we restrict the set of $\mathcal{V}$'s contacts only to the set of users who were tested positive. This can be justified by the fact that, in real life, the server owner often has a strong influence on the health authority (they can even be the same organization) and, thus, the server owner can easily make any user declared positive if needed.

We present our formal definition of resistance to an identity leaking attack, which uses the experiment described in Figure 3.6:

Definition 19. A DCT scheme is **identity-leak resistant** if for every probabilistic polynomial-time adversary $\mathcal{A}$ there is a negligible function $negl$ such that

$$\Pr[\text{Exp}^{\text{IdentityReveal}}_{\mathcal{A},\text{DCT}}(n) = 1] \leq \frac{1}{2} + negl(n).$$

Experiment $\mathbf{Exp}^{\text{IdentityReveal}}_{\mathcal{A},\text{DCT}}(n)$:
$k_1^0 \leftarrow_R \text{Init}(1^n),\ k_1^1 \leftarrow_R \text{Init}(1^n),\ d^* = 1$
$b^* \leftarrow_R \{0,1\}$
$e^* \leftarrow_R \{1,\ldots,e_{\max}\}$
$msg_{b^*} \leftarrow \text{MsgGen}(k_d^{b^*}, e^*)$
$CL \leftarrow \text{MsgRec}((\emptyset, d^*, e^*, msg_b)$
$t_0 \leftarrow \text{TraceGen}(k_1^0, d^*, 1)$
$t_1 \leftarrow \text{TraceGen}(k_1^1, d^*, 1)$
$b^{\mathcal{A}} \leftarrow_R \mathcal{A}(CL, (0,t_0), (1,t_1))$
Return $\begin{cases} 1 \text{ if } b^{\mathcal{A}} = b^* \\ 0 \text{ otherwise} \end{cases}$

Figure 3.6: Identity Reveal Experiment

Theorem 7. *Any correct DCT scheme with at least weak integrity, is not identity-leak resistant.*

Proof. In the experiment, the adversary gets the keys k_0 and k_1. Since the scheme is correct, we know that one of the keys can be used to verify the msg_{b^*}. Since the scheme has the weak integrity property, we know that the probability of both the keys being usable to verify the msg_{b^*} is negligible. Therefore, the adversary can try to verify the msg_{b^*} by one key at a time and output the bit of the key which succeeds in the verification. This approach has the probability of success $1 - \mu(n)$ for some negligible μ which we know is greater than $\frac{1}{2} + negl(n)$ and, thus, the DCT scheme is not identity-leak resistant.

□

Note that this attack requires the adversary to have access to very specific information. In DP4T and other successors of DP3T the central server has exactly this type of information which makes it suitable to be the adversary. On the other hand, it could be possible to construct a scheme which does not use a central server and, thus, provides no possibility that an adversary would get the type of information he needs. Therefore, in that scheme the identity leaking attack as described in this thesis would not even make sense. We leave the question of identity-leak resistant DCT schemes open for future research.

Time of Contact Disclosure

In this section, we present the second new attack on DCT schemes, where a dishonest server owner tries to determine when a contact happened. We call this attack a time revealing attack. We know from the previous section that a dishonest server owner can determine every pair of users who were in contact. Therefore, we can simply consider a scenario, where the server owner gets the history of a victim containing only one contact, and tries to determine when this contact happened.

We present our formal definition of resistance to a time revealing attack, which uses the experiment defined in Figure 3.7.

Definition 20. A DCT scheme is **time-reveal resistant** if for every polynomial

Experiment $\mathbf{Exp}^{\text{TimeReveal}}_{\mathcal{A},\text{DCT}}(n)$:
$d = 1,\ k_d \leftarrow_R \text{Init}(1^n)$
$(d^{\mathcal{A}}, e^0), (d^{\mathcal{A}}, e^1) \leftarrow \mathcal{A}(k_d, \Delta)$
$b \leftarrow_R \{0, 1\}$
While $d < d^{\mathcal{A}}:\ \text{Rotate}(k_d)$
$msg \leftarrow \text{MsgGen}(k_{d^{\mathcal{A}}}, e^b)$
$CL \leftarrow \text{MsgRec}(\emptyset, d^{\mathcal{A}}, e^b, msg)$
While $d < \Delta:\ \text{Rotate}(k_d)$
$t \leftarrow \text{TraceGen}(\{k_1, \ldots, k_\Delta\}, 1, 1)$
$b^{\mathcal{A}} \leftarrow \mathcal{A}(CL, t)$
Return $\begin{cases} 1 \text{ if } b = b^{\mathcal{A}} \\ 0 \text{ otherwise} \end{cases}$

Figure 3.7: Time Reveal Experiment

adversary $\mathcal{A}$ there is a negligible function *negl* such that

$$\Pr[\text{Exp}^{\text{TimeReveal}}_{\mathcal{A},\text{DCT}}(n) = 1] \leq \frac{1}{2} + negl(n).$$

In the experiment, we let the adversary know the tracing information generated by a positive user. Then, we give the adversary a history generated by the scheme for a victim who was in contact with the now positive user. The goal of the adversary is to determine the epoch of this contact.

We argue that DP4T is resistant to this type of attack unlike some other schemes which are considered secure.

GAEN 1.2: We start by briefly describing the GAEN 1.2 [LI20] scheme as it was described in [DDL+20]. This scheme is a successor of DP3T and is very similar in many ways. Similarly to DP3T, this scheme broadcasts *nym*s which are randomly generated from private daily keys. The first difference is that in GAEN 1.2 a daily key is not derived from the key for the previous day but is chosen uniformly random every day. When a user becomes positive, he shares all the keys he used in last Δ days and not only the one for the first day.

Integrity of GAEN 1.2 stems from the way it stores its messages, which contain only a *nym* and no additional information. The *nym*s are stored alongside with the time of receiving them. Note that this time is not masked or encrypted. When sharing his keys after being tested positive, a user also shares an information about when he was using each of the keys. This makes the GAEN 1.2 scheme achieve the strong integrity property as shown in [DDL+20]. On the other hand, this approach makes the scheme trivially vulnerable to a disclosure of the time of contact.

Note that this attack is not exclusive to the sever owner and can be performed by anyone who can obtain the application history (e.g., a thief).

Theorem 8. *GAEN 1.2 is* ***not*** *time-reveal resistant.*

Proof. In GAEN 1.2, the CL contains tuples (nym, d, e). Therefore, any adversary can easily extract the time (d, e) and win.

□

DP3T-UNLINK: Another successor of DP3T is DP3T-UNLINK from the authors of DP3T. We describe this scheme as it is described in [DDL+20]. This scheme chooses a different key (or a seed as they call it) for every *nym* it generates. Consequently, when a user becomes positive, he shares all of those seeds alongside with the information when were the seeds used. The messages this scheme sends also contain only the *nym*s and no additional information.

DP3T-UNLINK does not store the time of contact as a raw data. The scheme stores $H(nym, d, e)$ for a hash function H, a time (d, e), and an identifier nym. Its integrity stems from the trust put into the server. The server is responsible for generating all the *nym*s and computing their corresponding hash. The users then download only the list of hashes. Therefore, the users can never actually see the keys. This approach makes this scheme very robust. It achieves strong integrity as shown in [DDL+20]. On the other hand, security of this scheme strongly relies on the trust to the server. Therefore, we prove that a malicious server owner can succeed in a time revealing attack.

Theorem 9. *DP3T-UNLINK is* ***not*** *time-reveal resistant.*

Proof. An adversary $\mathcal{A}$ can generate all the possible outputs of the hash function same as the server does when preparing the *nym*s to be revealed. While doing this, $\mathcal{A}$ can mark which seed was used to compute each of these hash values. In the end, he outputs the time of the seed which generated the challenging nym. This approach will have the same probability of failure as the probability of the scheme generating the same nym for two different epochs, which we know is negligible since DP3T-UNLINK has the strong integrity property.

□

DP4T: Unlike other schemes, DP4T hides the additional information by $\mathsf{DMAC.Receive}$. It also sends its *nym*s in a random order. If the used DMAC is private, it makes any PPT adversary unable to determine in which epoch the communication happened.

Theorem 10. *If the used Delay-MAC* $\mathsf{DMAC} = (\mathsf{DMAC.Gen}, \mathsf{DMAC.Send}, \mathsf{DMAC.Receive}, \mathsf{DMAC.Vrf})$ *is statistically private, then DP4T is time-reveal resistant.*

Proof. In this scheme, an adversary $\mathcal{A}$ gets $CL = (nym, st)$ such that $st = \mathsf{DMAC.Receive}((d, e), tag)$. The value of nym is completely independent of time in DP4T. Therefore, it does not give $\mathcal{A}$ any information about time. We assume the DMAC to be statistically private. Therefore the probability that the st value could give $\mathcal{A}$ any information about the time is negligible. Thus, the probability of $\mathcal{A}$ succeeding in the experiment is $\frac{1}{2} + negl(n)$ for some negligible function $negl$. Thus, DP4T is time-reveal resistant.

□

Note that by statistical privacy of Delay-MAC the time-reveal resistance holds even if the adversary is computationally unbounded.

Remark 6. As we can see, the approach GAEN and DP3T-UNLINK used to overcome a replay attack made them vulnerable to revealing the time of a contact. This observation is important because it justifies, why we should not try to make these schemes even rela y attack resistant. In both cases this step would be trivial, as it would only require storing some coarse measured location (e.g., coordinates) alongside the time of contact. On the other hand, as we have just proven, this step would allow any malicious server owner to disclose not only who were its users in contact with but even when and where. The idea of decentralized contact tracing was to prevent the server owner from obtaining exactly this type of information. Therefore, this extension would jeopardize the founding idea of DCT.

On the other hand, if we do this improvement in DP4T, we do not show any new information. This is because the hiding property of Delay-MAC. In Pietrzak's second scheme, a sender tags not only his time (d, e), but a tuple (d, e, x, y), where x, y are some coarsely measured coordinates. This approach makes the scheme resistant to relay attacks as discussed in Section 3.1. Using the same argument as in the last proof, we can show that an adversary can get no information about the tuple (d, e, x, y) from the history of an honest user.

Conclusion

In this thesis, we formally defined Delay-MAC introduced by Pietrzak [Pie20] and the decentralized contact tracing scheme he builds using Delay-MAC, which we call DP4T. Using security model from [DDL+20], we proved that DP4T achieves strong integrity property as defined in [DDL+20] and, thus, is resilient to replay attacks. We discussed resilience of Pietrzak's [Pie20] improvement of DP4T against relay attacks.

We left as a question for future research how to construct DP4T to be message unlinkable as defined in [DDL+20]. We proved that DP4T is not trace unlinkable as defined in [DDL+20].

We presented two new attacks on DCT schemes. We proved that no correct DCT scheme with at least weak integrity property (as defined in [DDL+20]) is resilient against our identity revealing attack. We proved that unlike other schemes from the literature, DP4T is resilient to our time revealing attack.

We leave for future research how to use these results to construct a DCT scheme which would achieve both integrity and privacy.

Bibliography

[CDN15] Ronald Cramer, Ivan Bjerre Damgård, and Jesper Buus Nielsen. *Secure Multiparty Computation and Secret Sharing*. Cambridge University Press, 1 edition, 2015.

[DDL+20] Noel Danz, Oliver Derwisch, Anja Lehmann, Wenzel Pünter, Marvin Stolle, and Joshua Ziemann. Security and privacy of decentralized cryptographic contact tracing. *IACR Cryptol. ePrint Arch.*, 2020, 2020.

[KL15] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography and Second Edition*. CRC Press, 2015.

[LI20] Google LLC and Apple Inc. Exposure notification, 2020. https://covid19.apple.com/contacttracing.

[Pie20] Krzysztof Pietrzak. Delayed authentication: Preventing replay and relay attacks in private contact tracing. In Karthikeyan Bhargavan, Elisabeth Oswald, and Manoj Prabhakaran, editors, *Progress in Cryptology - INDOCRYPT 2020 - 21st International Conference on Cryptology in India, Bangalore, India, December 13-16, 2020, Proceedings*, volume 12578 of *Lecture Notes in Computer Science*, pages 3–15. Springer, 2020.

[TPH+20] Carmela Troncoso, Mathias Payer, Jean-Pierre Hubaux, Marcel Salath, James Larus, Edouard Bugnion, Theresa Stadler Wouter Lueks, Apostolos Pyrgelis, Sylvain Chatel, Daniele Antonioli, Ludovic Barman, Kenneth Paterson, Srdjan Capkun, David Basin, Jan-Beutel, Dennis Jackson, Bart Preneel, Nigel Smart, Dave Singelee, Aysajan Abidin, Seda Guerses, Michael Veale, Cas Cremers, Michael Backes, Nils Ole Tippenhauer, Reuben Binns, Ciro Cattuto, Alain Barrat, Giuseppe Persiano, Dario Fiore, Manuel Barbosa, and Dan Boneh. Decentralized privacy-preserving proximity tracing, 2020. https://github.com/DP-3T/documents.

[Vau20] Serge Vaudenay. Analysis of DP3T. *IACR Cryptol. ePrint Arch.*, 2020:399, 2020.

[zČr20] Ministerstvo zdravotnictví České republiky. eRouška, 2020. https://erouska.cz/audit-kod.

List of Figures