Terrorist Attacks for Fake Exposure Notifications in Contact Tracing Systems[*]

Gennaro Avitabile, Daniele Friolo, and Ivan Visconti

DIEM, University of Salerno, Italy
{gavitabile,dfriolo,visconti}@unisa.it

Abstract

In this work we show that an adversary can attack the integrity of contact tracing systems based on Google-Apple Exposure Notifications (GAEN) by leveraging blockchain technology. We show that through smart contracts there can be an on-line market where infected individuals interested in monetizing their status can upload to the servers of the GAEN-based systems some keys (i.e., TEKs) chosen by a non-infected adversary. In particular, the infected individual can anonymously and digitally trade the upload of TEKs without a mediator and without running risks of being cheated. This vulnerability can therefore be exploited to generate large-scale *fake* exposure notifications of at-risk contacts with serious consequences (e.g., jeopardizing parts of the health system, affecting results of elections, imposing the closure of schools, hotels or factories).

As main contribution, we design a smart contract with two collateral deposits that works, in general, on GAEN-based systems. We then also suggest the design of a more sophisticated smart contract, using DECO, that could be used to attack in a different way GAEN-based systems (i.e., this second smart contract can succeed even in case GAEN systems are repaired making ineffective the first smart contract).

Our work shows how to realize with GAEN-based systems (in particular with Immuni and SwissCovid), the terrorist attack to decentralized contact tracing systems envisioned by Vaudenay.

Keywords: Contact Tracing, GAEN, Smart Contracts

1 Introduction

During the COVID-19 pandemic, several governments have decided to use digital contact tracing systems in addition to other practices to contain the spread of SARS-CoV-2. The reason is that digital contact tracing could help in notifying at-risk exposures to individuals that have been in close proximity to people who subsequently tested positive to SARS-CoV-2. This could be very useful especially when the involved individuals do not know each other. If digital contact tracing systems worked perfectly, they would certainly be effective in alerting at-risk individuals who, following some prescribed procedures (e.g., informing doctors, staying at home in self-quarantine), may significantly limit the spread of the virus. Such systems have been highly recommended by some governments and in some cases (e.g., in Switzerland) an alert received by a contact tracing smartphone application allows to get a test for free.

[*]This paper is the full version of an ACNS Publication [AFV21].

The most used contact tracing systems rely on Google-Apple Exposure Notifications (GAEN), a feature offered by recent updates of iOS and Android and therefore available on a large fraction of currently used smartphones. These systems are widely used in Europe (e.g., Austria, Belgium, Germany, Ireland, Italy, Poland, Spain, Switzerland) and cross-border compatibility has recently been implemented[1].Moreover, in the US, several states have adopted GAEN-based systems. GAEN allows to run decentralized contact tracing where there is very low control from governments, and this makes attacks from third parties generally simpler to mount and harder to mitigate.

GAEN-based contact tracing systems. The approach of GAEN-based contact tracing systems is to use Bluetooth Low Energy (BLE) to detect close proximity contacts among smartphones. Each smartphone broadcasts random pseudonyms via BLE, and this information is received by smartphones in close proximity along with some encrypted metadata. If a citizen is tested positive and decides to notify others, she will upload a set of secret keys named *Temporary Exposure Keys* (we will refer to them as TEKs in the remainder of the paper) corresponding to previous days in which she was presumably contagious. Starting from a TEK, it is possible to generate all the pseudonyms broadcast by a user during a day. The receivers of such pseudonyms will then manage to decrypt the stored metadata to then evaluate a risk factor[2]. The TEKs are disseminated to the users via a back-end server that periodically posts a list of digitally signed TEKs. A detailed description of GAEN can be found at `https://covid19.apple.com/contacttracing`.

An important point is that GAEN evaluates the reported TEKs if and only if the digital signature verifies successfully under a public key that has been previously communicated by the developers to Apple and Google. Google motivates this requirement saying that it ensures that keys received by the devices are actually from the authorized server and not from malicious third parties[3]. Theoretically, one could also rely on server authentication using TLS, but the use of Content Delivery Networks (CDNs) to disseminate TEKs (e.g., the CDN used by Immuni is operated by Akamai, while the SwissCovid's one is operated by Amazon) requires protection against malicious modifications operated by the CDN itself. Unfortunately, as we will see next, this requirement paves the way for the development of dark economies where TEKs to be uploaded by infected users are traded through smart contracts.

False positives due to attacks. Since BLE was not originally designed to detect a precise distance among devices, the evaluation of the risk factor is prone to significant errors. To this regard, Leith and Farrell recently evaluated the reliability of BLE for digital contact tracing in several real-world scenarios [LF20b].

While false positives due to BLE limitations in measuring distance can indiscriminately affect all individuals using the smartphone apps, a much more concerning threat allowing to direct false positive alerts to specific targets has been pointed out in prior work (e.g., see [Vau20a, Pie20]). Indeed, GAEN-based contact tracing systems[4] can be heavily abused through replay attacks. In this case, the pseudonyms sent by an individual considered at risk (e.g., a person who is taking a test) are transmitted by an adversary to a different location in order to create a fake proximity contact. The attack can have a specific target but can also be performed at large

[1]EU eHealth Network: European Proximity Tracing. An Interoperability Architecture `https://lasec.epfl.ch/people/vaudenay/swisscovid/swisscovid-ana.pdf`.

[2]For example, metadata include information useful to estimate the distance among the smartphones which clearly impacts on estimating the risk of a contact.

[3]Google: Exposure Notification Reference Key Server `https://google.github.io/exposure-notifications-server/`.

[4]Sometimes for brevity we will just say GAEN systems.

scale. Recently, in [RGK20] Gennaro et al. discussed how the capability of running such attacks at large scale can be used to put a category of citizens in quarantine with the consequence of severely compromising the results of an election. In general, the malicious generation of false positives can be harmful in various ways, the health system can be overloaded of requests that can penalize those citizens who instead are really affected by the virus. A student can cause the complete closure of a school or university and similar attacks can be directed to shops, malls, gyms, post offices, restaurants, factories.

Risks related to replay attacks were already known back in April 2020, and GAEN systems have a pretty large time window (about 2 hours)[5] for pseudonyms to be replayed successfully. Nevertheless, governments have so far considered unlikely that such attacks can produce enough damage to cancel out the positive effects of genuine notifications of at-risk contacts. This is could be due to complications involved in the attack. Indeed, an adversary may not want to get herself infected, or it could not be easy to identify, and be in physical proximity with, an individual that soon will report to be infected.

In [Vau20b], Vaudenay envisioned the possibility of using smart contracts to realize a *terrorist* attack against decentralized systems, therefore the attack could potentially apply to GAEN-based systems as well. In this case, the attacker would spread on his targets some pseudonyms, subsequently promising through a smart contract a reward to whoever uploads the corresponding keys. Therefore, an infected individual who participates in the contract will cash a reward, and false positive alerts will raise on the smartphones of the targets selected by the terrorist. More details are discussed Sec. 1.2.

1.1 Our Contribution

In this paper, we show that the terrorist attack envisioned by Vaudenay can be concretely mounted against currently deployed GAEN-based contact tracing systems. In particular, we have analyzed its concrete feasibility with respect to two systems, such as Immuni [Imm20], used in Italy and SwissCovid [Swi20], used in Switzerland. We expect several other deployed GAEN systems to suffer from the same vulnerabilities.

More generally, our work shows how to attack the integrity of currently deployed GAEN-based contact tracing systems by leveraging blockchain technology. A very alarming side of our contribution is that current systems can be compromised without the need for the attacker to get infected, or to be with high probability in close proximity to individuals that will be soon detected positive and will upload the keys. Our attacks consist of smart contracts to establish a mediator-free market where parties, without knowing each other, without meeting in person and without running risks to be cheated, can abuse exposure notifications procedures of GAEN systems. We give a brief description of the mentioned smart contracts in the following.

Trading TEKs exploiting publicly verifiable lists of *infected* TEKs. As a main contribution we show a smart contract named Take-TEK that allows a buyer (i.e., the adversary willing to spread false positive alerts) to decide the TEKs that will be uploaded by a seller (i.e., the infected individual that is willing to monetize her right to upload TEKs to the servers of the GAEN system). The smart contract requires the buyer to deposit the amount of cryptocurrency (we will call it prize) that he is willing to give to the seller. The seller instead will deposit an amount of cryptocurrency in order to reserve a time slot in which she will try to upload the TEKs. In case she does not manage to complete the upload of the TEKs, the deposit will be assigned to the buyer. The deposit of the seller is therefore useful to make unlikely that a seller

[5]Google: Exposure Notification Specification https://blog.google/documents/69/Exposure_Notification_-_Cryptography_Specification_v1.2.1.pdf.

tries to prevent other sellers from completing the job. Additionally, we can hide the TEKs so that, even observing all transactions, it is not clear which TEKs have been traded using the smart contract among the many TEKs jointly published by the server of the contact tracing system during a slot.

Take-TEK crucially relies on the server publishing such lists of TEKs along with a signature verifiable with a publicly known public key. We show that the Take-TEK attack can be deployed to generate fake false positive alerts w.r.t. both Immuni and SwissCovid. Indeed, both systems follow strongly the design of GAEN and announce such signed lists of TEKs using ECDSA signatures.

Regardless of Immuni and SwissCovid making available or not their signature public keys, we have successfully extracted the public keys from previously released signatures. Therefore, Take-TEK can be instantiated to attack both (and possibly many more) systems. More details are discussed in Sec. 2.

Trading TEKs without publicly verifiable signatures: DECO. One might think that realizing the terrorist attack via smart contracts (e.g., Take-TEK) crucially relies on exploiting those signed lists of TEKs under a known (or extractable) public key. At first sight, a fix to such vulnerabilities consists of hiding the public keys and to use a signature scheme such that it is hard to extract the public key from signed messages. However, we show that things are actually more complicated for designers of contact tracing systems. In particular, we show another way to buy/sell TEKs that follows a completely different approach. The key idea is requiring the seller to prove that a TLS session with the server led to a successful upload of the buyer's TEKs. The only requirements on the communication between smartphone app and server are that 1) both the TEKs and the positive (or negative) outcome of the upload procedure are part of the exchanged application data in the TLS session, and 2) the upload phase consists of just one request made by the client and the response of the sever (e.g., as it is in SwissCovid). At first sight, the attack seems very hard to realize since notoriously TLS produces deniable communication transcripts when it comes to application data (i.e., exchanged messages are only authenticated and not digitally signed). However, we exploit a very recent work of Zhang et al. [ZMM$^+$]. They show how to build a fully decentralized TLS oracle, named DECO, for commonly used ciphersuites. Further details are described in Sec. 3.

Remark on the actual work done by our smart contracts. Both our smart contracts provide full guarantees to both seller and buyer at the expense of running some cryptographic operations that can obviously produce transaction costs. Nevertheless, if we make an additional optimization based on pragmatism, the expensive computations may happen very rarely in practice. Indeed, we notice that the main computational cost for those smart contracts consists of checking at the very end that the seller has completed the task of uploading TEKs correctly. We observe that a buyer can check that TEKs are published by the server on his own. As a result, he would be satisfied in finding out that the trade has been completed successfully. Therefore, it is natural to expect that the buyer would give his approval to the smart contract to transfer the money to the seller avoiding the execution of expensive computations, and therefore saving transaction costs[6]. Since this behavior would be visible in the wild, the reputation of the buyer would also benefit from such approvals and more sellers would want to run contracts

[6]Obviously, the smart contract can be adjusted so that, in case the buyer does not give his approval and the seller shows that she completed successfully her part of the contract, the expensive transactions costs due to the lack of help from the buyer are charged to the wallet of the buyer. A simple way to realize this could be asking for an additional deposit made by the buyer which could clearly cover the transaction costs of the seller in case the buyer does not give his approval and the seller shows that she successfully completed the upload procedure.

with him. Moreover, a (somewhat irrational) buyer that refuses to speed up the execution of the smart contract would anyway not stop the final transfer of the deposited money to the seller. As a result, the buyer would only get a worse reputation. In conclusion, the expensive work done by our smart contracts belongs to pieces of code that would rarely be executed in practice.

1.2 Related Work

The design of GAEN is very similar to the low-cost design of DP-3T[7], and thus several vulnerabilities identified in prior work generally apply to both systems. Tang [Tan20] observes that DP-3T is vulnerable to identification attacks and presents an accurate survey about contact tracing systems. In [Vau20a], Vaudenay reports both privacy and security issues. The most famous privacy attack is the so-called *Paparazzi* attack. Basically, it is possible through passive antennas to track infected individuals over a certain time window[8] during which pseudonyms are linkable.

Regarding security issues, Vaudenay extensively considers false alert injection attacks, where the adversary manages to raise false alerts on the smartphone apps of targeted victims. Within this category, there are *replay* and *relay* attacks. GAEN is vulnerable to relay attacks and to replay attacks carried out within two hours. Vaudenay in [Vau20a] and Pietrzak in [Pie20] proposed, back in April 2020, some solutions to defeat these attacks, but they have not been included neither in DP-3T nor in GAEN designs so far. Baumgärtner et al. [BDF+20] provide empirical evidence of the concrete feasibility of both Paparazzi and replay attacks. Pietrzak et al. [ACK+21] analyze inverse-sibyl attacks in which multiple adversaries cooperate to use the same pseudonyms. If one of the attackers gets to upload his TEKs, many false alerts may be raised. This attack could be used in combination with either the replay attack or our smart-contract based attacks in order to increase the number of affected targets. Iovino et al. [IVV21] concretely demonstrate the possibility to inject false alerts by replaying released TEKs. In particular, pseudonyms associated with already published TEKs are transmitted to smartphones whose clock is corrupted in order to make them believe these pseudonyms are valid for risk matching. They also show that several apps may publish TEKs that are still valid. These TEKs can be used to generate false alerts without the need of corrupting smartphones' clocks.

Several GAEN-based systems are currently used in the world for digital contact tracing. Vaudenay and Vuagnoux, and later Dehaye and Reardon, extensively evaluated SwissCovid [VV20, DR20b, DR20a], confirming some vulnerabilities showed in previous works and elucidating new ones.

Finally, another class of attacks leading to false alerts involves bribing. Vaudenay envisions various possibilities for the development of dark economies [Vau20b] which could support false alert injection attacks, allowing them to be carried out at very large scales. In particular, the *Lazy Student* attack is connected to replay attacks. It is based on a dark economy where a hunter (i.e., seller) collects pseudonyms of individuals who will likely become infected later on, and deposits them on a smart contract. If the TEKs corresponding to such pseudonyms are uploaded to the server of the contact tracing system, the hunter gets a reward paid by a buyer (i.e., the lazy student). If replay attacks are doable, the buyer can use them to make target victims' apps raise false alerts. This dark economy is sustainable only if the smart contract

[7]Decentralized Privacy-Preserving Proximity Tracing `https://github.com/DP-3T/documents/blob/master/DP3T%20White%20Paper.pdf`

[8]In GAEN, depending on the particular application, this time may amount to up 14 days if the adversary colludes with the authorities, and to one day assuming TEKs are properly mixed and anonymized prior to publication.

has a way to check that pseudonyms were actually reported to the official server. Another form of dark economy described by Vaudenay is the *terrorist* attack. It involves users reporting pseudonyms that differ from the ones used during the previous days. In fact, in both Immuni and SwissCovid there is no mechanism forcing users to upload genuine TEKs. Again, a TEK could be posted on a smart contract automatically issuing a reward to whoever reports it to the contact tracing system. This purchase may lead to a massive amount of fake notifications, without relying on replay attacks.

On the (missing) risk assessment of the terrorist attack. The huge impact of false injection attacks seems to have gone unnoticed or just ignored. In [LHML20] the cybersecurity risks of contact tracing systems are reviewed and compared using a subjective scoring scheme. The report considers injection of false alerts notifications by only mentioning replay attacks or trivial attacks such as recruiting people with symptoms, while the terrorist attack is not even mentioned.

Vaudenay and Vuagnoux expressed these and other concerns in their analysis of SwissCovid [VV20]. The Swiss National Cyber Security Center (NCSC) answered to their criticism seemingly downplaying those risks. The possible development of dark economies was ignored[9], and a recap table on security issues reports on SwissCovid marks the concerns expressed by Vaudenay as addressed, including false alert injection attacks[10]. Nevertheless, no solution or mitigation to such problems is reported.

Bribing attacks on smart contracts. As we discuss in Section 2, our smart contracts make possible to trade TEKs reducing at a minimum the risks related to interacting with a dangerous entity such as a criminal. Bribery attacks on smart contracts for different scenarios have been proposed in the context of bribing miners in Ethereum and Bitcoin [MHM18, LK17, TJS16, VTL17, NKW21].

2 Trading TEKs in GAEN Systems

The GAEN API has been created to provide an efficient platform for exposure notifications on top of which countries can easily develop digital contact tracing systems. GAEN is supposed to solve various technical problems (e.g., changing BLE MAC address synchronously with the rotation of pseudonyms, keeping BLE advertisements on in background) on a large fraction of available smartphones[11]. At the same time, the API is so inflexible that it forces anyone who is willing to benefit from it to adopt a specific design for contact tracing. What is left in the hand of the developers is merely the creation of the graphical interface, the choice of some parameters and the realization of a server to gather and spread data about infected users and, more importantly, an authentication mechanism to avoid the upload of data by non-infected users.

Whenever a user is tested positive, she is given the right to upload her TEKs to the server so that the other users can be notified a risk of infection. The mechanism can be implemented

[9]Swiss National Cyber Security Center: Security Issue Submission [INR-4434]. Detailed analysis. `https://www.melani.admin.ch/dam/melani/de/dokumente/2020/INR-4434_NCSC_Risk_assessment.pdf.download.pdf/INR-4434_NCSC_Risk_assessment.pdf`

[10]Swiss National Cyber Security Center: SwissCovid Proximity Tracing System - Public Security Test, page 8 `https://www.melani.admin.ch/dam/melani/de/dokumente/2020/SwissCovid_Public_Security_Test_Current_Findings.pdf.download.pdf/SwissCovid_Public_Security_Test_Current_Findings.pdf`

[11]Indeed, see the case of UK that tried to develop a system without GAEN but had to give up `https://www.bbc.com/news/technology-53095336`.

in different ways. For example, a simple method consists of a code generated by the app that is given first to the health operator in order to activate it on the server. Then, once the server has authorized the code, the app will upload the TEKs along with the code (e.g., Immuni follows this approach). More complex mechanisms may be put in place. However, the attack we show next works for every GAEN-based contact tracing system under some natural assumptions that we will discuss later.

In order to evaluate the contagion risk, GAEN provides appropriate methods that take as input two files containing the last TEKs and the related signature. The matching is not performed if the signature does not verify under a public key previously known to Google and Apple. The first file is named `export.bin` and contains, along with other fields, a list of TEKs belonging to infected users that have decided to perform the upload procedure. Each TEK has also a date attached, which indicates when such TEK was used. The second file, named `export.sig`, contains a digital signature of the file `export.bin`[12]. An example of `export.bin` is reported in App C.

2.1 Take-TEK Smart Contract: Buying/Selling TEKs Uploads

To simplify the description, we will refer to the TEKs file published by the server as a list of pairs of values. In each pair, the first value is a TEK and the second value is the corresponding date of usage date. Let the seller $\mathcal{P}$ be an infected user who would like to monetize her right to upload TEKs, and buyer $\mathcal{B}$ someone who is interested in paying $\mathcal{P}$ in order to upload TEKs of his interest. If the seller can prove she acted as promised, this selling process can be executed remotely remaining automated, anonymous, and scalable. GAEN's design requiring the list of TEKs to be signed makes the verification easy to the smart contract, and greatly facilitates such trades. The trade can be performed using a blockchain capable of executing sufficiently powerful smart contracts (e.g., Ethereum). Such smart contract guarantees that $\mathcal{P}$ gets an economic compensation if and only if $\mathcal{P}$ uploads to the server the TEKs specified by $\mathcal{B}$.

The high-level functioning of the smart contract is as follows. (1) $\mathcal{B}$ creates the smart-contract posting a list of TEKs with the related date, and deposits a prize to be redeemed by a seller. (2) An interested $\mathcal{P}$ also makes a small deposit to declare her intention to upload the TEKs specified by $\mathcal{B}$ (the purpose of this small deposit is explained later). After having made this deposit, (3) $\mathcal{P}$ has a specified amount of time to complete the upload procedure. Before the time runs out, $\mathcal{P}$ must provide a list of TEKs which includes all the pairs $(\mathsf{tek}, \mathsf{date})$ specified by $\mathcal{B}$, along with a valid signature under the server's public key. If $\mathcal{P}$ manages to do so, she gets a reward, otherwise both deposits go back to $\mathcal{B}$.

By making a deposit, the seller reserves a time slot during which she can perform the upload. Such deposit protects the buyer from denial of service (DoS) attacks by sellers who actually do not have the right to upload TEKs. Here, as in the remainder of the paper, with the word DoS we mean attacks carried out by fake sellers which prevent honest sellers from participating to the trade.

We name the above smart contract Take-TEK and the attack that leverages the use of this smart contract Take-TEK attack. The time window given to $\mathcal{P}$ must be wide enough to take into account that new TEKs are not continuously released by the server, in fact, several hours may pass between the submission of a TEK and its publication. Obviously, the amounts of both deposits will be significantly higher than transaction fees. A custom software is needed to upload arbitrary TEKs. However, this simple software may be developed even by other entities

[12]Apple: Setting Up an Exposure Notification Server `https://developer.apple.com/documentation/exposurenotification/setting_up_an_exposure_notification_server`. Google: Exposure Key export file format and verification `https://developers.google.com/android/exposure-notifications/exposure-key-file-format`

(not just the buyers), and publicly distributed on the Internet or other sources (e.g., Darknet). Therefore, all the seller would need to do is just running a software on a smartphone/computer; something that is easily doable by a large fraction of the infected citizens willing to gain money[13]. Additionally, the time given to the seller to complete the upload after having been tested positive must be long enough to reserve a slot on the blockchain (i.e., enough to wait that the transaction related to the seller's deposit gets confirmed) and subsequently send the TEKs via the custom software.

Attack description. $\mathcal{B}$ and $\mathcal{P}$ owns wallets $\mathsf{pk}_{\mathcal{B}}$ and $\mathsf{pk}_{\mathcal{P}}$ respectively. The buyer has no assurance that the seller is actually an infected person, and she is not just a malicious party trying to slow down the buyer's plan. Thus, some collateral must be deposited by $\mathcal{P}$ too. The seller will lose the collateral deposit in case she is not able to prove that she sent the buyer's TEKs to the server S. We use a signature scheme $(\mathtt{Gen}_{\mathsf{S}}, \mathtt{Sign}_{\mathsf{S}}, \mathtt{Ver}_{\mathsf{S}})$. The protocol description is depicted in Fig. 1 and a brief overview of the main functions follows below.

$\mathtt{Constructor}(\mathbf{T}_{\mathcal{B}}, \mathsf{vk}_{\mathsf{S}}, t, d_{\mathcal{P}})$: It takes as input a set of tuples $\mathbf{T}_{\mathcal{B}} := (\mathsf{tek}_i^{\mathcal{B}}, \mathsf{date}_i^{\mathcal{B}})_{i\in[n]}$ with $n \leq \mathsf{maxteks}$ [14], where tek_i is the i-th TEK of the buyer and date_i is the associated date, the verification key vk_{S} to be used to verify the signature of the TEKs list, a timestamp t, indicating the maximum time the seller has to provide the correct list and signature, and the collateral value $d_{\mathcal{P}}$ that the seller must deposit.

$\mathtt{Deposit}()$: must be triggered by $\mathcal{B}$ and takes as input a quantity p of coins as the payment for the seller.

$\mathtt{Promise}()$: must be triggered by the seller $\mathcal{P}$ by sending a quantity of collateral deposit $d_{\mathcal{P}}$ as a payment when invoked.

$\mathtt{SendTeks}(\mathbf{T}_{\mathsf{KS}}, \sigma_T)$: must be triggered by the seller $\mathcal{P}$ to provide a list of TEKs together with its signature σ_T. Let the list released by the server be $\mathbf{T} = (\mathsf{tek}_i, \mathsf{date}_i)_{i\in[N]}$, where N is the number of published TEKs. It checks that:

- $\mathbf{T}_{\mathcal{B}} \subseteq \mathbf{T}$ and $\mathtt{Ver}_{\mathsf{S}}(\mathbf{T}, \sigma_{\mathbf{T}}; \mathsf{vk}_{\mathsf{S}}) = 1$.

If the checks passes, $d_{\mathcal{B}}$ coins are transferred to the seller's wallet $\mathsf{pk}_{\mathcal{P}}$.

2.2 On the Practicality of Take-TEK Attack

Various proposed upload authorization mechanisms include manual steps (e.g., SwissCovid uses an authorization code, termed covidcode, which lasts for 24 hours) which, in order to function properly, naturally give the seller enough time to perform the steps mentioned in the section above. For example, if a code is communicated to the infected user via a phone call, she should be given a fairly large amount of time to write down the code and insert it in the app later on (the needs of people with disabilities and of elder people must be taken into account). Even systems that have fairly strict requirements on the time by which the upload procedure must be completed should allow for errors and recovery procedures, which may give additional time to the future seller. For example, Immuni requires that the infected user dictates, via phone call, a code that appears on her device. After that, the user must complete the upload within two

[13]COVID-19 by itself caused a global economic crisis which led to lower wages and job losses. More details at `https://en.wikipedia.org/wiki/COVID-19_recession`.

[14]The maximum number of TEKs that can be uploaded in one shot depends on the particular contact tracing system.

Take-TEK Attack

We consider two entities: the seller $\mathcal{P}$ and the buyer $\mathcal{B}$, with wallets $\mathsf{pk}_{\mathcal{B}}$ and $\mathsf{pk}_{\mathcal{P}}$ respectively. The protocol works as follows:

1. $\mathcal{B}$ invokes the constructor, taking as input the buyer's TEKs list $\mathbf{T}_{\mathcal{B}}$, the server verification key vk_{S} that will be used to verify the signed TEKs list, a timestamp t, and a value $d_{\mathcal{P}}$ indicating the minimal amount that $\mathcal{P}$ must deposit in order to participate. After having created the contract, $\mathcal{B}$ triggers the function `Deposit` to deposit the prize p aimed for the seller who uploads $\mathbf{T}_{\mathcal{B}}$ to the server.

2. $\mathcal{P}$ deposits her collateral by triggering the function `Promise`. Now the seller has at most time t to send a TEKs list $\mathbf{T}$ signed by the server.

3. If $\mathcal{P}$, before time t, triggers the function `SendTeks` submitting a signed TEKs list $\mathbf{T}$ such that it satisfies conditions $\mathbf{T}_{\mathcal{B}} \subseteq \mathbf{T}$ and $\mathtt{Ver}_{\mathsf{S}}(\mathbf{T}, \sigma_{\mathbf{T}}; \mathsf{vk}_{\mathsf{S}}) = 1$, the collateral deposit $d_{\mathcal{P}}$ of $\mathcal{P}$ and the prize p are transferred to $\mathcal{P}$'s wallet. Otherwise, if t seconds have passed, they are moved to $\mathcal{B}$'s wallet.

Figure 1: The steps followed by buyer $\mathcal{B}$ and seller $\mathcal{P}$ to carry out the Take-TEK attack.

minutes. If this does not happen, the procedure must be repeated. Additionally, the system should be tolerant. People should have the opportunity to perform the upload procedure later on if they are unable to do it in that precise moment. It is worth noting that strict requirements on the upload phase reduce users' privacy. A clear example is Immuni, where the medical operator, by checking whether a code has been used or is instead expired, gets to know whether or not the infected user actually uploaded her TEKs.

Implementation. We implemented our results as a smart contract for Ethereum, published it in a public repository[15] and tested it locally. Since Ethereum does not use `ECDSA-SHA256` (i.e., the one used in GAEN) for built-in transaction signature verification, there is the need to use specific solidity smart contract libraries[16] which lead to extra gas usage. Considering the change of 206 dollars per single ETH token on the 20th of July 2020, signature verification costs around 11 dollars (1235000 of gas). In order to compute the full cost, one should add about 0.4 dollars (45000 of gas) for each TEK that is contained in the export.bin file[17]. Note that our smart contract can handle export files large as the maximum data that an Ethereum transaction can handle at most. Details on how to deal with such limitation can be found in App. D.

2.3 Subtleties in the Wild

In 2.1 we gave a high-level overview of how TEKs uploads can be sold safely via blockchains. However, there are some subtleties we overlooked for the sake of simplicity. We first analyze the

[15]Code available at `https://github.com/danielefriolo/TEnK-U`.

[16]The one we used for signature verification is available at `https://github.com/tdrerup/elliptic-curve-solidity`.

[17]The cost of 45000 of gas includes TEK extraction, hashing of the export file for signature verification, checking if the stored TEKs are in the extracted ones. To simplify the gas evaluation, we assume that $\mathcal{B}$ stores only one TEK in the contract.

advantages for adversaries when using automated trade compared to already known attacks. Then, we consider certain problems that arise while trying to concretely mount our attack against deployed GAEN-based contact tracing systems. We also show how these difficulties are easily tackled if very small modifications to our attack are made.

Advantages of automated trade (for an adversary). One might think that malicious injection of fake TEKs is inherent in decentralized contact tracing systems since there is no control over the smartphone used by infected individuals and thus, when the time of the upload comes, the infected person can always use a smartphone belonging to someone else.

While it is true that such simple attacks are very hard to tackle, they have limited impact for at least two main reasons: 1) the buyer must handover his smartphone to the seller, and this requires physical proximity; 2) sellers and buyers must trust each other since an illegal payment must be performed without being able to rely on justice in case of missing payment or aborted upload of keys. Indeed, even if in need of money, people are generally afraid of dealing with criminals since they may get scammed or threatened. Additionally, the buyer might expose the sellers' identities to the authorities in case he gets arrested or legally persecuted. Equally, the buyer may share the same concern with respect to an unreliable seller. It goes without saying that some citizens are prone to violate the rules[18] when they believe that risks are low compared to the advantages.
As such, attacks involving the exchange of smartphones, or the usage of a malicious app uploading TEKs sent by a criminal contacted directly by the infected citizen, do not scale and their damage may be considered tolerable. Having a mechanism which allows this trade to happen remotely, in anonymity and ensuring no party is cheated, solves all the above problems for parties willing to abuse contact tracing systems. In fact, it provides a framework for large-scale black markets of TEKs. The seller would not feel threatened in any way and could easily earn money, on the other hand, the buyers would benefit from a larger set of users to be in business with, therefore succeeding in many possible attack scenarios. Other systems for black markets based on reputations could also be used, but they are clearly less appealing than the transparency and usability of mediator-free smart contracts.

A worry-free seller. The effectiveness of a digital contact tracing system is strictly related to various factors among which the percentage of active population using them. Appropriate measures should be taken to earn citizens' trust since it is the only way to guarantee broad adoption. With this in mind, the European Commission released a series of recommendations in relation to data protection stating the need of identifying solutions that are the least intrusive and comply with the principle of data minimization [Eur20]. A similar recommendation has been given by the Chaos Computer Club (CCC)[19], the Europe's largest ethical hackers association, which explicitly states that "data which is no longer needed must be deleted". Corona-Warn, the German contact-tracing system, declares to be fully compliant with CCC's guidelines[20]. Many other systems are inspired by similar principles. For example, the Italian system Immuni also declares that data is deleted when no longer needed[21], as well as the Swiss system SwissCovid which also specifies a retention period for the TEKs and the upload authorization codes [22]. In

[18] The infected person also commits a violation by allowing the injection of fake TEKs.

[19] 10 requirements for the evaluation of "Contact Tracing" apps `https://www.ccc.de/en/updates/2020/contact-tracing-requirements`.

[20] Criteria for the Evaluation of Contact Tracing Apps `https://github.com/corona-warn-app/cwa-documentation/blob/ec703906c109bd7c3cc84bc361b7e703b20650ea/pruefsteine.md`.

[21] `https://github.com/immuni-app/immuni-documentation`.

[22] Corona-Warn-App Solution Architecture `https://github.com/corona-warn-app/cwa-documentation/blob/master/solution_architecture.md`.

its recommendation to build a verification server authenticating the uploaded TEKs, Google states that identifiable information should not be associated with uploaded data[23].
The adoption of the above measures ensures that uploaded data do not link to, nor identify a particular individual. This is very important considering that GAEN systems are vulnerable to the Paparazzi attack[24] [Vau20a].

Evaluation of seller's risks. Considering the above data minimization principles, are the seller and the buyer at risk of being legally persecuted for a trade that may be deemed as illegal? The answer seems to be no. If data is handled as specified above, there would be no way to associate the seller to its uploaded TEKs at a later time. Data exchanged during the attack would also not directly compromise neither the buyer nor the seller[25].

However, there is a problem for a seller who really wants to minimize the chance of getting caught. In fact, since the TEKs proposed by the buyer are posted in clear on the blockchain, authorities may become aware of them and activate ad-hoc procedures monitoring the incriminated TEKs and exploiting the upload authorization process to identify the guilty seller. This, in fact, does not seem to directly contradict the data minimization principle when national security is at stake. If the server getting the TEKs upload monitors the requests (e.g., by storing connection logs) without colluding with the health authority, the seller could be easily incriminated after the TEKs have been detected in the smart contract by just looking at her IP stored together with such request. However, in this case, the usage of an anonymity service like Tor [DMS04] can easily reduce the chance of getting caught. If the authorities are colluding, the upload authorization codes (e.g., the covidcode) may be associated with the identities of infected users, and TEKs could be in turn mapped to a precise individual via such codes. However, by slightly increasing the complexity of the smart contract, such risk may be completely avoided. It suffices for the buyer to encrypt his TEKs with a public key provided by the seller, who then will use a non-interactive zero-knowledge (NIZK) proof system to prove that the TEKs encrypted under the specified public key are indeed contained in the list signed with the server's public key. This requires an additional interaction with the buyer, who has to publish the encrypted TEKs (see App. B for more details). Once again, the seller is protected by a timer which assigns her all the deposits if the buyer does not reply. Efficient Ethereum implementations of NIZK proofs (see App. A.3) are known in literature, like NIZKs for Σ-protocols [Wil18] or zk-SNARKs [Sem20, ZoK20, ZkD20].

Even if the buyer decides to claim the authorship of the attack at a later point in time (e.g., as it usually happens for terrorist attacks) by opening the encrypted values on the blockchain to published TEKs, the seller would not be at risk if data was handled according to the principles of data economy and anonymity. Any evidence based on contact tracing data would be a clear indicator that those principles have been violated. This could result in a big disincentive in using the app, since citizens may think (probably rightfully) that data could also be abused for other reasons, perhaps for mass surveillance purposes. Finally, we want to point out that even if several researchers raised the concern about the possible birth of black markets [Vau20b, RGK20], we did not find any document related to any contact-tracing system, either issued by governments

[23]Google: Exposure Notification Verification Server `https://developers.google.com/android/exposure-notifications/verification-system`.

[24]In *Paparazzi* attack, through passive antennas one can link pseudonyms used by an infected user tracing him over the duration of a TEK or for more days if the TEKs are linked. Therefore leaving open the possibility to link such data to a person's real identity would be extremely incautious.

[25]In this analysis, we refer only to contact tracing system data and messages exchanged via the blockchain during the execution of the attack. We do not take into account border-line situations as, for example, the case where there is only a single infected individual. We also ignore additional information that may help investigators figuring out who the seller is, for example how the money are spent after the trade.

or national security agencies, which deeply evaluates these risks. To the best of our knowledge, no risk analysis ever mentions to monitor the dark web and blockchains looking for suspicious smart contracts. It goes by itself that if blockchains are not monitored, all extra measures taken in this paragraph to protect the seller are not necessary.

Other subtleties. There are two other subtleties with limited impact to consider for the actual realization of the attack. We describe them in App. E.1 and App. E.2 and shortly mention here:

- *Extracting public keys from signatures*: Generally, servers' public keys do not seem to have been made publicly available neither by Google and Apple, nor by the countries which deployed GAEN-based contact tracing systems[26]. However, the signature algorithm used (i.e., ECDSA) allows to retrieve this public key starting from a pair of signed messages.
- *Updates of public keys*: The structure of the `export.bin` file allows for updates of the used digital signature key (see App. C). Therefore, it might happen that, after the seller makes the deposit and accepts to upload the buyer's TEK, the server, by coincidence, decides to use a new key which was never used before, thus producing a signature that is not verifiable under the public key posted on the smart contract. However, by making a slight modification to the smart contract, it is possible to handle also this unfortunate event. Moreover, keys have changed very rarely in export files up to now.

3 Connecting Smart Contracts to TLS Sessions

The Take-TEK attack relies on the fact that a digital signature is used to authorize uploads. Additionally, the ability to extract the public key from signed messages may also play a key role. Therefore, one might think that to protect GAEN systems the public key should remain hidden and the signature scheme should be such that extracting the public key from message-signature pairs is hard. In this way, due to the inability of allowing a smart contract to verify that a TEK is officially in a list of infected TEKs, the attack would fail. However, things are not so easy. The previous smart contract exploited the public verifiability of the signatures because this is what is used in GAEN systems. If a different method is used, it might be abused again. Indeed, we show that TLS oracles can be used to prove to a smart contract that an upload was successfully performed, without relying on signatures of TEKs.

3.1 Decentralized Oracles

Recently, Zhang et al. [ZMM$^+$], introduced the concept of Decentralized Oracles. Roughly, an oracle is an entity that can be queried by a client to interact with a TLS server and help the client proving statements about the connection transcript. Previously known oracle constructions rely on trusted/semi-trusted execution environments [ZCC$^+$16], thus not giving any help in our case. DECO [ZMM$^+$] is the first work where a fully-decentralized construction is proposed for specific ciphersuites such as CBC-HMAC and AES-GCM coupled with DH key exchange with ephemeral secrets. We recall that a TLS connection is divided in two parts: a handshake phase where key exchange is performed, and a phase during which the transferred data is encrypted/decrypted by the client/server using the key exchanged in the previous phase. GAEN servers usually accept Elliptic-Curve Diffie-Hellman key Exchange (ECDHE) for the first phase, while for the second

[26]Once a contact tracing system handles his public key to Google, it can completely rely on GAEN APIs to perform signature verification without storing the public key in clear to the app source code (see `https://developers.google.com/android/exposure-notifications/exposure-key-file-format` for more details).

phase some servers accept only AES-GCM (e.g., Immuni), whereas others, like SwissCovid's one, accept also CBC-HMAC as a ciphersuite. To guarantee the integrity of data, the plaintext is usually compressed and a MAC on the compressed string is calculated using a key derived from the DH exchanged key.

Decentralized Key-Exchange. We provide below an informal description of the key-exchange in DECO for ECDHE that is called Three Party Handshake (3PHS).

We assume three entities: a prover $\mathcal{P}$, a verifier $\mathcal{V}$ and a server S. $\mathcal{P}$ and $\mathcal{V}$ jointly act as a TLS client. The overall idea of DECO is that the prover and verifier, after performing some two-party computations, compute shares of the exchanged key, while the server computes the entire key without even noticing that $\mathcal{P}$ and $\mathcal{V}$ are two distinct interacting entities.

When using CBC-HMAC, the keys $k_{\mathcal{P}}^{\mathtt{MAC}}$, $k_{\mathcal{V}}^{\mathtt{MAC}}$ (such that $k_{\mathcal{P}}^{\mathtt{MAC}} + k_{\mathcal{V}}^{\mathtt{MAC}} = k^{\mathtt{MAC}}$) are learned by $\mathcal{P}$ and $\mathcal{V}$ respectively, while $k^{\mathtt{Enc}}$ is only known to $\mathcal{P}$. When using AES-GCM, the same key is used for both encryption and MAC, therefore both $\mathcal{P}$ and $\mathcal{V}$ just get a share of it. While $\mathcal{P}$ and $\mathcal{V}$ only learn their secret shares of the key, the server S gets to know both $k^{\mathtt{Enc}}$ and $k^{\mathtt{MAC}}$.

Let G be an EC group generator. The key exchange phase works as follows:

- $\mathcal{P}$ establishes a TLS connection with the server S.
- When receiving the DH share $Y_{\mathsf{S}} = s_{\mathsf{S}} \cdot G$ from S, $\mathcal{P}$ forwards it to $\mathcal{V}$.
- $\mathcal{V}$ samples a DH secret $s_{\mathcal{V}}$ and sends his DH share $Y_{\mathcal{V}} = s_{\mathcal{V}} \cdot G$ to $\mathcal{P}$.
- $\mathcal{P}$ samples her DH secret $s_{\mathcal{P}}$, calculates her DH share $Y_{\mathcal{P}} = s_{\mathcal{P}} \cdot G$, calculates the combined DH share $Y = Y_{\mathcal{P}} + Y_{\mathcal{V}}$, and sends Y to S.

Finally, S computes the DH exchanged key as $Z = s_{\mathsf{S}} \cdot Y$. $\mathcal{P}$ and $\mathcal{V}$ will compute their secret shares of Z as $Z_{\mathcal{P}} = s_{\mathcal{P}} \cdot Y_{\mathsf{S}}$ and $Z_{\mathcal{V}} = s_{\mathcal{V}} \cdot Y_{\mathsf{S}}$. Note that $Z_{\mathcal{P}} + Z_{\mathsf{S}} = Z$, where $+$ is the EC group operation. Now that $\mathcal{P}$ and $\mathcal{V}$ have secret shares of EC points, they use secure two-party computation (2PC) to evaluate a PRF (that we call TLS-PRF) to derive the keys $k_{\mathcal{P}}^{\mathtt{MAC}}$ and $k_{\mathcal{V}}^{\mathtt{MAC}}$. The authors face and solve several challenges in order to derive keys efficiently via 2PC. We do not cover this part, a more detailed description can be found in [ZMM$^+$].

Encrypted communication. At the end of the 3PHS, $\mathcal{P}$ and $\mathcal{V}$ have to engage in a 2PC protocol to correctly calculate the MAC and the encryption on the plaintext to be sent to the server, without revealing the shares to each other. Privacy of the plaintext is also ensured with respect to $\mathcal{V}$. For CBC-HMAC, the encryption of such plaintext is computed exclusively by $\mathcal{P}$ who holds the encryption key. The authors [ZMM$^+$] provide hand-optimized protocols which are much more efficient then the ones obtained by directly applying 2PC techniques. The 2PC protocol for AES-GCM is a lot slower than the one for CBC-HMAC since for AES-GCM $\mathcal{P}$ and $\mathcal{V}$ must cooperate also for the encryption.

Proving statements. An important feature of DECO is that $\mathcal{P}$, when the communication with S ends, can prove, in zero knowledge, statements on the communication transcript in a clever and efficient way. However, to make their protocol practical for our goal, we do not try to maintain the transcript private. As a result, we will not discuss this part of DECO which can be found in [ZMM$^+$]. In the following, we describe how to adapt DECO to our scenario.

3.2 A Smart Contract Oracle

Our goal is to make the smart contract play the role of the DECO verifier. In this way, the smart contract would be able to verify that the intended communication between the seller and the server took place and to reward the seller accordingly. Unfortunately we can not just plug DECO into a smart contract for several reasons. For example, DECO requires to run intensive 2PC related tasks, to sample random values and to maintain a private state[27].

Therefore, we keep running the DECO protocol off-chain but we find a way to connect the DECO run between the prover and the verifier to the state of the smart contract, so that the smart contract will eventually be able to act as an impartial judge punishing the malicious party when a deviation from the prescribed honest behavior is detected. In particular, the seller acts as a prover and the buyer as a verifier, and we guarantee no party is able to cheat (i.e., the seller is paid if and only if she performs the upload of the requested TEKs) by binding the off-chain execution to the state of the smart contract itself. Furthermore, we guarantee privacy of the messages exchanged between the server and the prover only until their TLS connection is open. After the communication ends, the seller proves that she acted honestly by providing the application-level messages exchanged with the server, along with the corresponding MAC tags w.r.t. the MAC key which is bound to the smart contract. To be more specific, the smart contract freezes a share of the MAC key and the seller has to show a communication transcript (i.e., the messages exchanged with the server and corresponding MAC tags) which is consistent with such share. Privacy of the upload request message to be sent to the server is crucial while the TLS session is open because the verifier may abort the protocol and use the authorization token of the prover to upload data by himself without paying out the promised reward. On the other hand, making the communication public after it took place does not endanger the prover, apart from the considerations made in Section 2.3, and makes the verification procedure much more practical. What we need is that the shares of the prover and the verifier are kept private until the end of the protocol, and then revealed to the smart contract, along with other information, for verification and reward paying. In addition, the TLS session timeout should be big enough to allow for the 2PC execution. To this regard, Zhang et. al already verified the practical feasibility of their protocol [ZMM$^+$]. Obviously, $\mathcal{P}$ must know how to reach $\mathcal{V}$ to carry out the protocol. To address concerns regarding anonymity, $\mathcal{V}$ may set up a Tor hidden service[28]. Using hidden services may significantly slow down the process, however we found both Immuni and SwissCovid servers to give a generous time out window of two hours[29]. Another point to consider is that upload authorization tokens may have a limited duration. For example, in SwissCovid,the smartphone, by interacting with an appropriate server (different from the TEKs upload server, called CovidCode-Service), exchanges the covid code for a signed JWT token that is valid for 5 minutes[30]. Then, this token is sent by the smartphone to the server along with the TEKs to complete the upload. Thus, the upload message, containing the TEKs and the authorization token, must be computed and sent to the server within 5 minutes from the reception of the JWT token. Given the high efficiency of DECO when CBC-HMAC is used, even when bandwidth is limited [MMZ$^+$20], it is reasonable to think that the attack is feasible in SwissCovid. In Immuni instead, no signed token is used. In fact, the upload must be completed within 2 minutes after the infected user has communicated the code to the health

[27]Keeping a private state inside a smart contract is not possible and computationally intensive operations generate high costs.

[28]More on Tor hidden services can be found at `https://2019.www.torproject.org/docs/onion-services`.

[29]Interestingly, in June the timeout of a TLS session with both Immuni and SwissCovid upload servers was limited to 5 minutes, but it has been then extended to two hours.

[30]See CovidCode-Service configuration `https://github.com/admin-ch/CovidCode-Service/blob/develop/src/main/resources/application-prod.yml`.

operator. Therefore, in Immuni the attack would less likely be operative, especially with Tor, given that the slower AES-GCM ciphersuite is required (see App. G).

Protocol description. From now on, we refer to the seller and the buyer as prover $\mathcal{P}$ and verifier $\mathcal{V}$ respectively; we denote the server as S. In the following, we explain the detailed attack for the CBC-HMAC ciphersuite. When creating the smart contract, $\mathcal{V}$ also posts the DH share $Y_{\mathcal{V}} = s_{\mathcal{V}} \cdot G$ he is willing to use during the 3PHS, along with requested TEKs (and dates).

First, $\mathcal{P}$ transacts on the smart contract to reserve a time slot of duration t_1 by which a DECO protocol run must be performed together with $\mathcal{V}$ and S, and the data needed to redeem the reward must be posted on the smart contract by $\mathcal{P}$. If time t_1 elapses, $\mathcal{P}$ loses her slot. This reservation mechanism is needed to prevent $\mathcal{V}$ from getting back the reward while an honest $\mathcal{P}$ performs the upload of the requested TEKs. In fact, the verifier could also act as a prover and simulate a reward-paying interaction with the server to the smart contract, which would have no mean to distinguish it from a fake one. By adding a reservation mechanism, we are sure a malicious $\mathcal{V}$ cannot play a simulated transcript in the smart contract while honest $\mathcal{P}$ is performing with him the DECO protocol run. Furthermore, since the communication for the upload between the server and the prover consists of just a single query followed by a single response, it is not possible for a cheating verifier to make the timer expire avoiding to pay the prover while at the same time the upload of the TEKs successfully completes. In fact, once $\mathcal{V}$ cooperates with $\mathcal{P}$ to build a valid request, S will reply to $\mathcal{P}$ independently of what $\mathcal{V}$ does, thus giving $\mathcal{V}$ all she needs to redeem the reward.

When executing the 3PHS, $\mathcal{P}$ checks that the value $Y'_{\mathcal{V}}$ sent by $\mathcal{V}$ during the handshake corresponds to the value $Y_{\mathcal{V}}$ posted on the smart contract. This prevents $\mathcal{V}$ from providing an erroneous DH share and blaming $\mathcal{P}$ for it. If this is not the case, $\mathcal{P}$ aborts. Since no upload message has been sent to the server yet, no party gains advantage from this operation. If $\mathcal{V}$'s share is correct (i.e., $Y_{\mathcal{V}} = Y'_{\mathcal{V}}$), parties engage in the communication with S and jointly compute the MAC (via 2PC as in [ZMM$^+$]) on the upload request m_c generated by $\mathcal{P}$. If the connection ends successfully[31], the elected $\mathcal{P}$ posts (only who reserved this slot is allowed to post this message) on the smart contract the following:

- The entire communication transcript, that is (m_c, m_s) together with the MACs (θ_c, θ_s), calculated by the client(s) $\mathcal{P} \leftrightarrow \mathcal{V}$ and the server S.
- The prover's secret $s_{\mathcal{P}}$.
- The DH share of the server Y_{S} received during the 3PHS.

Then, the smart contract starts a timer t_2 indicating the maximum time $\mathcal{V}$ has to reveal his secret $s_{\mathcal{V}}$. In case $\mathcal{V}$ does not do that, the prize is automatically transferred to the seller $\mathcal{P}$. If $\mathcal{V}$ reveals $s_{\mathcal{V}}$, the smart contract does the following:

- Check that $Y_{\mathcal{V}} = s_{\mathcal{V}} \cdot G$ and if not, transfer the prize to $\mathcal{P}$.
- If the check passes, reconstruct the secret Z from $s_{\mathcal{V}}, s_{\mathcal{P}}, Y_s$, and apply TLS-PRF to derive the MAC key $k^{\mathtt{MAC}}$.

[31]This can be inferred from the communication. For example, as in SwissCovid (see SwissCovid Server Controller: `https://github.com/DP-3T/dp3t-sdk-backend/blob/a730a5b276591e5cc8b6c609e2b0ba29c6069eb6/dpppt-backend-sdk/dpppt-backend-sdk-ws/src/main/java/org/dpppt/backend/sdk/ws/controller/GaenController.java`), S may reply $\mathcal{P}$ with either a success message such as "200 OK" or an error message.

Now the smart contract has everything it needs to check that the fields inside message m_c (from the prover to the server) are correct (i.e., the buyer's TEK are present), the response message (from the server to the prover) is positive, and that the MACs (θ_c, θ_s) verify w.r.t. $k^{\texttt{MAC}}$. If all the checks pass, the prize is transferred to $\mathcal{P}$, otherwise $\mathcal{P}$ gains no prize and the deposit is returned back to $\mathcal{V}$.

As mentioned before, $\mathcal{V}$ is not encouraged to provide a different public key w.r.t. the one he used in DECO execution, otherwise $\mathcal{P}$ will just abort. On the other hand, the prover is not able to earn a reward without uploading the promised TEKs. In fact, the probability for the prover to come up with a pair (m'_c, θ'_c) (resp. (m'_s, θ'_s)) that verifies under the key $k'^{\texttt{MAC}}$ derived from $Z' = Z'_{\mathcal{P}} + Z'_{\mathcal{V}}$ with $Z'_{\mathcal{P}} := s'_{\mathcal{P}} \cdot Y'_{\mathsf{S}}$ and $Z_{\mathcal{V}} := s_{\mathcal{V}} \cdot Y'_{\mathsf{S}}$ is negligible due to the fact that $s_{\mathcal{P}}$ is fixed and honestly generated, thus randomizing Z', hence $k'^{\texttt{MAC}}$.

A note on DoS attacks. It is important to prevent DoS attacks run by sellers who actually do not have the right to upload TEKs and end up by just wasting the buyer's precious time. In the previous discussion this protection is not provided: before sending the jointly computed message (m_c, θ_c), the seller can decide to not forward the message to the server. Now, the buyer has to open his commitment to show his secret $s_{\mathcal{V}}$ in order to not lose the prize. As a result, the committed value cannot be used in other runs. To address this issue, the smart contract can be modified to handle multiple sessions. Instead of storing $Y_{\mathcal{V}}$ as a single DH contribute, the buyer stores the root of a Merkle tree. Now, when the seller interacts with the contract to reserve a session, a session id (a simple counter j suffices) is assigned to her: the DH contribute used in the 3HPS will correspond now to the j-th leaf of the Merkle tree. Now, when the buyer has to open his secret $s_{\mathcal{V}}$, he also reveals the path of the Merkle tree from the root to the leaf j. The smart contract will now verify that the contribute is correctly derived from the root by following a path with correct openings. Let us consider a Merkle root committing to 2^k elements, thus allowing the buyer to open as many sessions. For a k large enough, a malicious seller should spend a considerable amount of money in order to reserve all the sessions.

AES-GCM. Carrying out the attack when AES-GCM ciphersuite is required is more involved. A discussion on this is reported in App F.1.

4 Conclusion

In our work we showed that the terrorist attack, previously envisioned by Vaudenay, is concretely realizable against GAEN systems with the aid of cryptographic tools and a blockchain capable of executing smart contracts (e.g., Ethereum). In particular, the Take-TEK attack exploits the fact that the list of infected TEKs, published by the server daily, has always a digital signature attached to it. Such signature allows the smart contract to easily verify that the upload was performed as requested by the terrorist. Even beyond the use of signatures, we have shown a different instantiation of the terrorist attack using DECO. In conclusion, we advise protocol designers not to look at the effects of a specific realization, but to prove the protocol secure against any automated instantiation of a terrorist attack. Our work shows that the power of blockchain technology to trade digital assets is still overlooked even when critical features are digitized.

5 Acknowledgments

We thank the first author of DECO [ZMM$^+$] Fan Zhang for all the clarifications about their paper, Stephen Farrell of the TACT project [LF20a] for his help on how to gather contact tracing data, Serge Vaudenay and Martin Vuagnoux for useful information about the implementation and configuration of SwissCovid. This research is supported by the European Union's Horizon 2020 research and innovation programme under grant agreement No 780477 (project PRIViLEDGE).

References

[ACK$^+$21] Benedikt Auerbach, Suvradip Chakraborty, Karen Klein, Guillermo Pascual-Perez, Krzysztof Pietrzak., Michael Walter, and Michelle Yeo. Inverse-sybil attacks in automated contact tracing. In *Proc. of CT-RSA*, volume To appear, 2021.

[AFV21] Gennaro Avitabile, Daniele Friolo, and Ivan Visconti. Tenk-u: Terrorist attacks for fake exposure notifications in contact tracing systems. *Applied Cryptography and Network Security 2021*, To appear, 2021.

[BDF$^+$20] Lars Baumgärtner, Alexandra Dmitrienko, Bernd Freisleben, Jonas Höchst, Mira Mezini, Markus Miettinen, Thien Duc Nguyen, Alvar Penning, Filipp Roos, Ahmad-Reza Sadeghi, Michael Schwarz, and Christian Uhl. Mind the GAP: Security & privacy risks of contact tracing apps. In *TrustCom 2020, Security Track*, pages 458–467, 2020.

[DMS04] Roger Dingledine, Nick Mathewson, and Paul F. Syverson. Tor: The second-generation onion router. In *USENIX*, pages 303–320, 2004.

[DR20a] Paul-Olivier Dehaye and Joel Reardon. Proximity tracing in an ecosystem of surveillance capitalism. *CoRR*, abs/2009.06077, 2020.

[DR20b] Paul-Olivier Dehaye and Joel Reardon. Swisscovid: a critical analysis of risk assessment by swiss authorities. *CoRR*, abs/2006.10719, 2020.

[Eur20] European Commission. Guidance on apps supporting the fight against COVID 19 pandemic in relation to data protection. *Official Journal of the European Union*, 2020.

[Imm20] Immuni Team. Immuni's high-level description. `https://github.com/immuni-app/immuni-documentation`, 2020. Accessed: 2020-08-23.

[IVV21] Vincenzo Iovino, Serge Vaudenay, and Martin Vuagnoux. On the effectiveness of time travel to inject covid-19 alerts. In *Proc. of CT-RSA*, volume To appear, 2021.

[LF20a] Dough Leith and Stephen Farrell. Testing apps for COVID-19 tracing (TACT). `https://down.dsg.cs.tcd.ie/tact/`, 2020. Accessed: 2020-08-23.

[LF20b] Douglas J. Leith and Stephen Farrell. Coronavirus contact tracing: evaluating the potential of using bluetooth received signal strength for proximity detection. *Comput. Commun. Rev.*, 50(4):66–74, 2020.

[LHML20] Franck Legendre, Mathias Humbert, Alain Mermoud, and Vincent Lenders. Contact tracing: An overview of technologies and cyber risks. *CoRR*, abs/2007.02806, 2020.

[LK17] Kevin Liao and Jonathan Katz. Incentivizing blockchain forks via whale transactions. In *Financial Cryptography*, pages 264–279, 2017.

[MHM18] Patrick McCorry, Alexander Hicks, and Sarah Meiklejohn. Smart contracts for bribing miners. In *Financial Cryptography*, pages 3–18, 2018.

[MMZ$^+$20] Deepak Maram, Harjasleen Malvai, Fan Zhang, Nerla Jean-Louis, Alexander Frolov, Tyler Kell, Tyrone Lobban, Christine Moy, Ari Juels, and Andrew Miller. Candid: Can-do decentralized identity with legacy compatibility, sybil-resistance, and accountability. *IACR Cryptol. ePrint Arch.*, 2020:934, 2020.

[NKW21] Tejaswi Nadahalli, Majid Khabbazian, and Roger Wattenhofer. Timelocked bribing. In *Financial Cryptography*, volume To appear, 2021.

[Pie20] Krzysztof Pietrzak. Delayed authentication: Preventing replay and relay attacks in private contact tracing. In *Proc. of INDOCRYPT*, pages 3–15, 2020.

[RGK20] Adam Krellenstein Rosario Gennaro and James Krellenstein. Exposure notification system may allow for large-scale voter suppression. `https://static1.squarespace.com/static/5e937afbfd7a75746167b39c/t/5f47a87e58d3de0db3da91b2/1598531714869/Exposure_Notification.pdf`, 2020. Accessed: 2020-08-23.

[Sem20] Semaphore Team. Semaphore. `https://semaphore.appliedzkp.org/`, 2020. Accessed: 2020-09-15.

[Swi20] Swiss Federal Office of Public Health. New coronavirus: Swisscovid app and contact tracing. `https://www.bag.admin.ch/bag/en/home/krankheiten/ausbrueche-epidemien-pandemien/aktuelle-ausbrueche-epidemien/novel-cov/swisscovid-app-und-contact-tracing/datenschutzerklaerung-nutzungsbedingungen.html#-11360452`, 2020. Accessed: 2020-08-23.

[Tan20] Qiang Tang. Privacy-preserving contact tracing: current solutions and open questions. *CoRR*, abs/2004.06818, 2020.

[The03] The OpenSSL Project. OpenSSL: The open source toolkit for SSL/TLS. `www.openssl.org`, 2003.

[TJS16] Jason Teutsch, Sanjay Jain, and Prateek Saxena. When cryptocurrencies mine their own business. In *Financial Cryptography*, pages 499–514, 2016.

[Vau20a] Serge Vaudenay. Analysis of DP3T. *IACR Cryptol. ePrint Arch.*, 2020:399, 2020.

[Vau20b] Serge Vaudenay. Centralized or decentralized? the contact tracing dilemma. *IACR Cryptol. ePrint Arch.*, 2020:531, 2020.

[VTL17] Yaron Velner, Jason Teutsch, and Loi Luu. Smart contracts make bitcoin mining pools vulnerable. In *Financial Cryptography*, pages 298–316, 2017.

[VV20] Serge Vaudenay and Martin Vuagnoux. Analysis of swisscovid. `https://lasec.epfl.ch/people/vaudenay/swisscovid/swisscovid-ana.pdf`, 2020. Accessed: 2020-08-23.

[Wet] Dirk Wetter. testssl.sh. `https://testssl.sh/`.

[Wil18] Zachary J. Williamson. Aztec. `https://github.com/AztecProtocol/AZTEC/blob/master/AZTEC.pdf`, 2018. Accessed: 2020-09-15.

[Yan19] H. Yang. *EC Cryptography Tutorials - Herong's Tutorial Examples.* Herong's Tutorial Examples. Herong Yang, 2019.

[ZCC+16] Fan Zhang, Ethan Cecchetti, Kyle Croman, Ari Juels, and Elaine Shi. Town crier: An authenticated data feed for smart contracts. In *ACM CCS*, 2016.

[ZkD20] ZkDAI Team. Zkdai. `https://github.com/atvanguard/ethsingapore-zk-dai`, 2020. Accessed: 2020-09-15.

[ZMM+] Fan Zhang, Deepak Maram, Harjasleen Malvai, Steven Goldfeder, and Ari Juels. DECO: liberating web data using decentralized oracles for TLS. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *Proc. of CCS '20*, pages 1919–1938. ACM.

[ZoK20] ZoKrates Team. Zokrates. `https://zokrates.github.io/`, 2020. Accessed: 2020-09-15.

A Tools

A.1 MACs and Signature Schemes

A *Message Authentication Code* consists of a tuple of algorithms $(\texttt{Gen}, \texttt{Tag}, \texttt{Ver})$ such that

$\texttt{Gen}(1^\lambda)$: Takes as input the security parameter and outputs a key k in the key space $\mathcal{K}$.

$\texttt{Tag}(m; k)$: Takes as input a message m in the message space $\mathcal{M}$ and a key k, and outputs a tag θ.

$\texttt{Ver}(m, \theta; k)$: Takes as input a message m and a key k, and outputs 1 iff θ is a correct tag on m under key k.

It must satisfy the following properties:

- **Completeness**: The probability that $\texttt{Ver}((m, \theta); k)$ outputs 1 for an honestly generated tag $\theta \leftarrow \texttt{Tag}(m; k)$ is 1.
- **Unforgeability**: The probability that an adversary, knowing only challenge message m^* and having access to an oracle giving back tags θ_i on messages $m_i \neq m^*$ (for all $i \in [n]$ with n polynomially bounded in the security parameter), outputs a pair (m^*, θ^*) such that $\texttt{Ver}(m^*, \theta^*; k) = 1$ is negligible.

A *Signature Scheme* consists of a set of algorithms $(\texttt{Gen}, \texttt{Sign}, \texttt{Ver})$, such that

$\texttt{Gen}(1^\lambda)$: Takes as input the security parameter and outputs a pair $(\mathsf{sk}, \mathsf{vk})$ sampled from the key space, where sk is the signing key and vk the verification key.

$\texttt{Sign}(m; \mathsf{sk})$: Takes as input a message m in the message space $\mathcal{M}$ and a signing key sk, and outputs a signature σ_m on that message.

$\texttt{Ver}(m, \sigma; \mathsf{vk})$: Takes as input a pair (m, σ) and the verification key vk, and outputs 1 if the signature σ correctly verifies under vk.

It must satisfy the following properties:

- **Completeness**: The probability that $\mathtt{Ver}((m, \sigma); \mathsf{vk})$ outputs 1 for an honestly generated signature $\sigma \leftarrow \mathtt{Sign}(m; \mathsf{sk})$ is 1.
- **Unforgeability**: The probability that an adversary, knowing only the challenge message m^* and having access to an oracle giving back signatures σ_i on messages $m_i \neq m^*$ (for all $i \in [n]$ with n polynomially bounded in the security parameter), outputs a pair (m^*, σ^*) such that $\mathtt{Ver}(m^*, \sigma^*; \mathsf{vk}) = 1$ is negligible.

A.2 Public-Key Encryption Schemes

A Public-Key Encryption Scheme is a tuple of algorithms $(\mathtt{Gen}, \mathtt{Enc}, \mathtt{Dec})$ such that

$\mathtt{Gen}(1^\lambda)$: Takes as input the security parameter, outputs a couple $(\mathsf{pk}, \mathsf{sk})$ of keys sampled in the key spaces.

$\mathtt{Enc}(m; \mathsf{pk})$: Takes as input a message m in the message space and a public key pk, and outputs the ciphertext c in the ciphertext space.

$\mathtt{Dec}(c; \mathsf{sk})$: Takes as input a ciphertext c and a secret key, and output a message m'.

A PKE scheme is CPA-Secure if the following properties are satisfied

- **Completeness** The probability that $m = m'$, where $m' \leftarrow \mathtt{Dec}(c; \mathsf{sk})$ with $c \leftarrow_{\$} \mathtt{Enc}(m; \mathsf{pk})$ for an honestly generated pair $(\mathsf{pk}, \mathsf{sk}) \leftarrow_{\$} \mathtt{Gen}(1^\lambda)$ is 1.
- **CPA-Security** The probability that an attacker, after choosing two messages (m_0, m_1), giving them to a challenger, and receiving back the encryption of one of the two (chosen by the challenger flipping a coin), can distinguish which of the two messages were encrypted, is negligible.

A.3 NIZK proofs

In a *zero-knowledge proof* system an entity $\mathcal{P}$, called prover, can prove to another entity, called verifier, that an NP-statement x is in some language $\mathcal{L}$ (i.e., there exists at least a witness w such that the relation $\mathcal{R}_\mathcal{L}(x, w)$ for the language $\mathcal{L}$ is satisfied) without revealing a single bit of information on the used witness. Informally, the following properties must be satisfied by a zero-knowledge proof system:

- **Completeness:** The probability that an honest prover $\mathcal{P}$ (i.e. computing the proof by providing a valid (x, w) such that $\mathcal{R}_\mathcal{L}(x, w) = 1$) convinces the verifier $\mathcal{V}$ about the validity of the statement is 1.
- **Soundness:** The probability that a cheating prover convinces the verifier that a statement x is not in the language $\mathcal{L}$ is negligible.
- **Zero Knowledge:** If the statement x is true, the verifier learns no more information other than the fact that the statement is true. This concept is formalized by showing that there exists an efficient simulator that, given only the statement, can produce a protocol transcript that is indistinguishable from a real protocol execution.

A proof is said to be non-interactive when the interaction consists solely on a message sent by the prover to the verifier. A *zero-knowledge proof of knowledge* is a zero-knowledge proof where the prover shows that he actually *knows* a witness for the statement x and this is formalized by showing an efficient extractor that gives a witness in output. When we refer to NIZK proofs throughout the paper we usually intend NIZK proofs of knowledge.

In the random oracle model, both prover and verifier access to a cryptographic hash function that in the security proof is modeled as a random oracle. The simulator for the zero-knowledge property and the extractor for the proof of knowledge property have the power to program the random oracle.

B Adding Seller's Privacy

As discussed in Section 2.3, using publicly posted TEKs is dangerous for the seller due to possible risks of incrimination. This could disincentivize the seller to utilize such smart contract mechanism. To guarantee seller's privacy, in all of our attacks we can enrich our playground by assuming the existence of a CPA-Secure PKE encryption scheme $(\mathtt{Gen}, \mathtt{Enc}, \mathtt{Dec})$ and a NIZK proof system. The proposed protocols can be modified as follows:

- When the buyer creates the smart contract, he waits that a seller $\mathcal{P}$ is elected before providing his TEKs. When $\mathcal{P}$ is elected, $\mathcal{B}$ posts his TEKs encrypted with $\mathcal{P}$'s public key $\mathsf{pk}_{\mathcal{P}}$, by triggering an algorithm $\mathtt{SendBuyerTeks}(\mathbf{C}_{\mathcal{B}})$ where $\mathbf{C}_{\mathcal{B}} = (c_1, \dots, c_n)$, with $c_i \leftarrow\!\!\$\ \mathtt{Enc}(t_i)$ for each $t_i \in \mathbf{T}_{\mathcal{B}}$. TEKs are pairs $t_i = (\mathsf{tek}_i, \mathsf{date}_i)$.

- When the signed TEKs list is available, the seller triggers $\mathtt{SendTeks}(\mathbf{T}, \sigma_T, \Pi, \tilde{\mathbf{T}})$, where $\mathbf{T} = (\tilde{t_1}, \dots, \tilde{t_N})$ are the published TEKs, σ_T the corresponding signature, and $\Pi = (\pi_1, \dots, \pi_n)$ is a sequence of proofs in which π_i is a NIZK proof that the prover knows $t_i \leftarrow \mathtt{Dec}(c_i; \mathsf{sk}_{\mathcal{P}})$ and that at least one element $\tilde{t_j}$ in a subset $\tilde{\mathbf{T}} \subseteq \mathbf{T}$ such that $|\tilde{\mathbf{T}}| > |\mathbf{T}_{\mathcal{B}}|$ is equal to t_i. The smart contract checks all the proofs, and if all of them verify, transfer the prize to the seller.

Now the only information that an external observer can deduce by looking at the proofs is that all the encrypted buyer's TEKs are indeed inside the list (or in a subset of them). Depending on how the date field is handled it may be also necessary to encrypt it and to prove a slightly more complicated statement. To be sure that an observer cannot pinpoint the buyer's TEKs precisely, it is sufficient that the proofs use as a statement a subset of the published TEKs that contains at least one more TEK w.r.t the buyer's TEKs (proving on a subset and not on the entire list can be beneficial in terms of proof size and efficiency). The only harmful case is when the number of published keys matches with the number of the buyer's keys. We can argue that this condition happens quite rarely, considering that one external more key is sufficient to guarantee buyer's safety, and if GAEN recommendations are followed, a decent amount of keys should be present in the list.

C GAEN Export Files

An example of an `export.bin` file for Immuni, the Italian contact tracing app is reported below. The meaning of the main fields is commented on the side. The `start_timestamp` and `end_timestamp` are expressed in UTC seconds,
`rolling_start_interval_number` is expressed in 10 minutes increments from UNIX epoch. The `export.sig` contains the digital signature of the `export.bin` file, along with the field

`signature_infos`.
The content description of the `export.bin` file follows.

```
start_timestamp: 1591254000 //start of the time window of included keys
end_timestamp: 1591268399 //end of the time window of included keys.
region: "222"  batch_num: 1 batch_size: 1
signature_infos {
verification_key_version: "v1" //version of used verification key
verification_key_id: "222"
signature_algorithm: "1.2.840.10045.4.3.2"
1: "it.ministerodellasalute.immuni"}
keys {
key_data: ".." //base64 encoded TEK
transmission_risk_level: 8
rolling_start_interval_number: 2651616 //date of usage of TEK
rolling_period: 144}...
```

D Implementation Improvements

As noted in Section 2, our smart contract implementation of Take-TEK can handle export files large as the maximum transaction size at most. This limitation can be overcome by making the smart contract accepting the file split in multiple chunks (a transaction for each chunk), and then extracting the keys and verifying the signature by hashing the concatenation of all the stored chunks. A trivial solution to this problem can be to store $n-1$ chunks in the smart contract, and when the seller sends the n-th chunk, the smart contract performs the concatenation, extracts the keys, and verifies the signature. Unfortunately, storing data in a smart contract is the most expensive operation in terms of gas cost, and storing such a big piece of data in a smart contract state may be too expensive. However, exploiting the Merkle-Damgård construction used by SHA to hash multiple blocks, way less amount of data needs to be stored. Let us define `Hash` as the hashing algorithm and H_i as the hash of the i-th chunk C_i. TEKs extraction and signature verification in the chunk-based mechanism can be done in the following way:

- The seller divides the export file in different chunks in such a way that, when each chunk is hashed, the hash climbs up to the same level of the Merkle tree of the other hashed chunks.
- When the seller sends a new chunk to the smart contract, the latter extracts all the TEKs contained in the chunk, checks which of the buyer's TEK are present in the chunk and stores this information[32]. After that, it hashes the chunk and stores the hashed value H_i.
- When the last chunk is sent to the smart contract by the seller (together with the signature of the entire export file), the smart contract extracts the last pieces of information, checks if the TEKs contained in the last chunk cover the not yet appeared buyer's TEKs, computes its hash H_n, hashes its concatenation with the previously stored hashed chunks (i.e. it calculates $H_{\mathtt{out}} = \mathtt{Hash}(H_1, \ldots, H_n)$) and triggers the signature verification procedure giving the value $H_{\mathtt{out}}$ and the signature file as input.

[32]During the chunk splitting, some TEKs may be cut in half. The smart contract should take care of the first and the last bits of each chunk and reconstruct the missing information.

As can be noticed, the application of the hashing algorithm to the concatenation of the H_is makes the hashing algorithm climbing up to the root of the Merkle tree, thus giving the expected hash of the entire file as output. Now the amount of bits needed to be stored is around $|H| \cdot n = 512 \cdot n$, vs $|C_i| \cdot n$ (usually the maximum transaction size, and so C_i in our case, is around 44 Kbytes in Ethereum).

E Other Subtleties: Details

E.1 Extracting Public Keys from Signatures

Take-TEK (cfr., Section 2.1) requires that the server's public key is known to both the involved parties. This guarantees that the buyer is sure the reward is paid only to sellers who actually upload data to the contact tracing system, and that honest sellers are sure they will be able to satisfy the conditions to be paid, namely obtaining a valid digital signature for reward redemption. A Github issue asking for the public key of the Italian contact tracing app was opened on the 7th of June 2020 and it has still not been addressed at the time of writing. SwissCovid Android app contains a configuration file specifying the production version of the bucket public key (the value `BUCKET_PUBLIC_KEY` can be found in `https://github.com/DP-3T/dp3t-app-android-ch/blob/master/app/backend_certs.gradle`) that is used to perform signature verification outside GAEN. Anyway, as we can notice with Immuni, this is not a requirement. One might think that keeping the verification keys secret may prevent attacks as the one of Section 2.1. However, it turns out that it is actually not the case. In fact, since GAEN uses ECDSA, starting from a signature and the related message we can recover two candidate public keys, one of which will match the actual one with overwhelming probability. A practical example showing this procedure can be found in [Yan19]. Such message/signature pairs are generally made publicly available and are easily accessible by appropriately querying the server of the specific contact tracing system. Multiple pairs per day may be released. A comprehensive description on how to get this data has been provided by the Testing Apps for COVID-19 Tracing (TACT) project, along with scripts to automate the downloading process [LF20a]. We also practically performed the extraction procedure, successfully extracting the keys for both SwissCovid and Immuni.

E.2 Updates of Public Keys

There is a subtle technical problem with the attack described in Section 2.1. The digital signature keys that the server uses may change over time. In fact, as shown in App C, the `export.bin` file includes a field indicating a version for the verification key. This field follows a progressive numeration, that is the first version is termed v1, the second one v2 and so on. This means that the server may change the verification key it uses, perhaps within a set of keys that have been pre-shared with Google and Apple. Therefore, it might happen that, after the seller makes the deposit and accepts to upload the buyer's TEK, the server, by coincidence, decides to use a new key which was never used before, thus producing a signature that is not verifiable under the public key posted on the smart contract.

However, by making a slight modification to the smart contract, it is possible to handle also this unfortunate event. Having realized that she would be unable to redeem the reward, the seller might activate a special recovery condition. After this, the buyer will be able to collect both deposits if and only if he manages to provide a pair of export files which have an `end_timestamp` (cfr., App. C) subsequent to the time of the recovery request and verify under the public key originally posted on the smart contract; otherwise the deposits are returned to the original owners. Obviously, enough time should be given to the buyer to provide the export files, similarly to what happens to the seller after her deposit.

This event is certainly very annoying for the seller and might play as disincentive to join the trade, but taking a look at real-world data one realizes that this is a relatively rare event. We considered several countries which are currently using a digital contact tracing system, namely: Italy, Switzerland, Austria, Germany, Ireland, Northern Ireland, Denmark, Latvia, Canada and US Virginia. Until January 13th 2021 (last time we checked), only US Virginia and Italy have switched to the second version of the verification key. In particular, the change to the Italian system dates back to the 15th of June 2020[33] and no modifications have been made since then. Notably, some countries' systems, like Switzerland and Germany's ones, are active from several months now and the verification key has not changed at all. To the best of our knowledge, the criteria by which the verification key should change is not documented anywhere.

F Further Notes on Our Smart Contract Oracle

F.1 CBC-HMAC vs AES-GCM

Differently from CBC-HMAC, AES-GCM relies on the same key for both encryption and MACs. The impact of AES-GCM is twofold: 1) more computation is needed to perform the required 2PC to calculate messages from/to the server, due to the AES algorithm itself, 2) the prover does not learn the encryption key after 3PHS, meaning that both encryption and decryption must be done via 2PC as well. On the smart contract side, this difference boils down to a lack of fairness. After $\mathcal{V}$ and $\mathcal{P}$ have calculated together the upload message and sent it then to S, $\mathcal{V}$ could decide not to help the prover to decrypt the server's response. Now, $\mathcal{P}$ has no witness in her hands to give to the smart contract in order to prove that she has correctly performed the TEKs upload. As a result, she cannot redeem the prize. The problem can be easily solved by giving to the smart contract the burden of decrypting the server's ciphertext. In our approach, $\mathcal{V}$ must commit to his key and open it later. When this happens, the server reconstructs the MAC/encryption key, decrypts the ciphertext, does the necessary checks, and pay the prize to $\mathcal{P}$. The CBC-HMAC version of DECO is way faster then the AES-GCM one. However, looking at practical evaluations made by the authors [ZMM$^+$, MMZ$^+$20] it is reasonable to think that all their solutions may fit in the time window given by contact tracing servers (e.g., 2 hours in Immuni and SwissCovid) for the TLS connection (see App. G), even when hiding $\mathcal{V}$ through Tor hidden services. What is less likely is that, in the case of Immuni which uses AES-GCM and requires the upload to be completed within two minutes, the upload request message (m_c, θ_c) is computed and sent to the server in time; especially when the prover and the verifier communicate via Tor.

G TLS Connections with Immuni and SwissCovid

In this section we show useful information about TLS sessions established running on the client side the tools `openssl` [The03], and `testssl` [Wet] in order to connect to the TEKs upload servers of Immuni and SwissCovid.

Immuni. Using `testssl` to connect to `upload.immuni.gov.it` one can see that the server accepts TLS 1.2 connections only and the preferred ciphersuite is `ECDHE-RSA-AES256-GCM-SHA384, 256 bit ECDH (P-256)`. Moreover, one can also use `ECDHE-RSA-AES128-GCM-SHA256`.

[33]This change occurred in the 4th export file.

Using `openssl` to connect to `upload.immuni.gov.it` one can see all the parameters of the established TLS session, including in particular a timeout of 7200 seconds (i.e., 2 hours) that is very large and thus beneficial for our attacks. For completeness we show here the content of the standard output (we replace some potentially identifying data by "...").

```
openssl s_client -connect upload.immuni.gov.it:443 -cipher ECDHE-RSA-AES256-GCM-SHA384

CONNECTED(00000003)
depth=2 C = IT, L = Milan, O = Actalis S.p.A./03358520967, CN = Actalis Authentication Root
CA
verify error:num=19:self signed certificate in certificate chain
verify return:1
depth=2 C = IT, L = Milan, O = Actalis S.p.A./03358520967, CN = Actalis Authentication Root
CA
verify return:1
depth=1 C = IT, ST = Bergamo, L = Ponte San Pietro, O = Actalis S.p.A./03358520967, CN =
Actalis Organization Validated Server CA G2
verify return:1
depth=0 C = IT, ST = ROMA, L = ROMA, O = Sogei S.p.A., OU = Server Sicuri, CN =
upload.immuni.gov.it
verify return:1
---
Certificate chain
 0 s:C = IT, ST = ROMA, L = ROMA, O = Sogei S.p.A., OU = Server Sicuri, CN =
upload.immuni.gov.it
   i:C = IT, ST = Bergamo, L = Ponte San Pietro, O = Actalis S.p.A./03358520967, CN =
Actalis Organization Validated Server CA G2
 1 s:C = IT, ST = Bergamo, L = Ponte San Pietro, O = Actalis S.p.A./03358520967, CN =
Actalis Organization Validated Server CA G2
   i:C = IT, L = Milan, O = Actalis S.p.A./03358520967, CN = Actalis Authentication Root CA
 2 s:C = IT, L = Milan, O = Actalis S.p.A./03358520967, CN = Actalis Authentication Root CA
   i:C = IT, L = Milan, O = Actalis S.p.A./03358520967, CN = Actalis Authentication Root CA
---
Server certificate
-----BEGIN CERTIFICATE-----
MIIGYzCCBUugAwIBAgIQRnoupUMoTHxpWMpARKffOTANBgkqhkiG9w0BAQsFADCB
lTELMAkGA1UEBhMCSVQxEDAOBgNVBAgMB0JlcmdhbW8xGTAXBgNVBAcMEFBvbnRl
IFNhbiBQaWV0cm8xIzAhBgNVBAoMGkFjdGFsaXMgUy5wLkEuLzAzMzU4NTIwOTY3
MTQwMgYDVQQDDCtBY3RhbGlzIE9yZ2FuaXphdGlvbiBWYWxpZGF0ZWQgU2VydmVy
IENBIEcyMB4XDTIwMDUxMzA3MTcyN1oXDTIxMDUxMzA3MTcyN1oweTELMAkGA1UE
BhMCSVQxDTALBgNVBAgMBFJPTUExDTALBgNVBAcMBFJPTUExFTATBgNVBAoMDFNv
Z2VpIFMucC5BLjEWMBQGA1UECwwNU2VydmVyIFNpY3VyaTEdMBsGA1UEAwwUdXBs
b2FkLmltbXVuaS5nb3YuaXQwggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIB
AQC/2cqmDJieHJ8dMOzT/dlPpLGCAfWukniW5eFdNexZK3qxpESrrBm27OT6llLv
Oui/laRlqOHwg+2xycf1+aIGFiTO8dKuVyoCdJxCiQqCNF9dtMpgA69DgfYv/4o6
GwXvyx0PjQ/eF5+wJbjO1hIGYDm83J0WNbmHhqA8NzHrqP5Q554aNYrLnzXDVpdN
3I3Gdo/KlUkuH5RUutYhbVZand8uf069MFROzL1xifdHLVqCwYrNRkyc6BqyCVoV
c2f1TEQZ9T9OQxijXkMdXXwNkUXKS6O/SVtGUiUm2KgQ098XqOzEs6U/0aWVyFPt
YMSSh8hpT3bR3eaLjI2yMX/5AgMBAAGjggLIMIICxDAMBgNVHRMBAf8EAjAAMB8G
A1UdIwQYMBaAFGL+uyeKZETtaJZaWHmh21omrf+7MH4GCCsGAQUFBwEBBHIwcDA7
BggrBgEFBQcwAoYvaHR0cDovL2NhY2VydC5hY3RhbGlzLml0L2NlcnRzL2FjdGFs
aXMtYXV0aG92ZzIwMQYIKwYBBQUHMAGGJWh0dHA6Ly9vY3NwMDkuYWN0YWxpcy5p
dC9WQS9BVVRIT1YtRzIwHwYDVR0RBBgwFoIUdXBsb2FkLmltbXVuaS5nb3YuaXQw
UQYDVR0gBEowSDA8BgYrgR8BFAEwMjAwBggrBgEFBQcCARYkaHR0cHM6Ly93d3cu
YWN0YWxpcy5pdC9hcmVhLWRvd25sb2FkMAgGBmeBDAECAjAdBgNVHSUEFjAUBggr
BgEFBQcDAgYIKwYBBQUHAwEwSAYDVR0fBEEwPzA9oDugOYY3aHR0cDovL2NybDA5
LmFjdGFsaXMuaXQvUmVwb3NpdG9yeS9BVVRIT1YtRzIvZ2V0TGFzdENSTDAdBgNV
```

```
HQ4EFgQUFCzbrEIZXwf4JgUy4YCgmpuPKJcwDgYDVR0PAQH/BAQDAgWgMIIBBQYK
KwYBBAHWeQIEAgSB9gSB8wDxAHYARJRlLrDuzq/EQAfYqP4owNrmgr7YyzG1P9Mz
lrW2gagAAAFyDO5cFgAABAMARzBFAiBc/J6oayZGC43Uoec5S432UxCy/AmXaX2P
0gDXEUjJxwIhAKr6mS90XWxe/wa599GmXLD0FYG7QFYYt3Hw2ef/7hk6AHcA9lyU
L9F3MCIUVBgIMJRWjuNNExkzv98MLyALzE7xZOMAAAFyDO5cVQAABAMASDBGAiEA
sdib2FsWyErV+T3IgJqnlw0quecJ8nlerqxHSi+jX+MCIQDySdDQ5ssmiu3pW9MY
60td+s/UOb6oIekdqZKCYTv1aDANBgkqhkiG9w0BAQsFAAOCAQEAD0DHgxyVPgl+
IOwRl6huoOiaFseBfR5dBHTyPa/axCCZxwtZNU8rkPPWfHp36e4iSbOHwEmMjAEr
h9lWR786ohUerN9EUd98Xais/RgJ0uN1TZfQM72nmgw0hYciy0MyUmULUSbbPSDs
JL5zs3pn2E7oCoagdNS14kpp/LDGo8iwitTK7XYtdOu/SAv1k9WfLjY3tR+hNAfJ
R9FTxyRYNFDOaWlwMgDISAdS3WHU8yJ6QntFTpLQ3vEHvTgswe+pTFzz2Yxl3CFk
D0AbbJ7AcBWDbAr+9H6GDF8uQ7om5SVm0CIUqdQRlG6RMetdz+36QPWuNiCV4c4O
gP7cycp+/Q==
-----END CERTIFICATE-----
subject=C = IT, ST = ROMA, L = ROMA, O = Sogei S.p.A., OU = Server Sicuri, CN =
upload.immuni.gov.it

issuer=C = IT, ST = Bergamo, L = Ponte San Pietro, O = Actalis S.p.A./03358520967, CN =
Actalis Organization Validated Server CA G2

---
No client certificate CA names sent
Peer signing digest: SHA256
Peer signature type: RSA
Server Temp Key: ECDH, P-256, 256 bits
---
SSL handshake has read ... bytes and written ... bytes
Verification error: self signed certificate in certificate chain
---
New, TLSv1.2, Cipher is ECDHE-RSA-AES256-GCM-SHA384
Server public key is 2048 bit
Secure Renegotiation IS supported
Compression: NONE
Expansion: NONE
No ALPN negotiated
SSL-Session:
    Protocol  : TLSv1.2
    Cipher    : ECDHE-RSA-AES256-GCM-SHA384
    Session-ID: ...
    Session-ID-ctx:
    Master-Key: ...
    PSK identity: None
    PSK identity hint: None
    SRP username: None
    Start Time: ...
    Timeout   : 7200 (sec)
    Verify return code: 19 (self signed certificate in certificate chain)
    Extended master secret: no
---
```

SwissCovid. Using `testssl` to connect to `www.pt1.bfs.admin.ch` one can see that the server accepts TLS 1.2 connections only and accepts, interestingly, also the CBC-HMAC ciphersuite, therefore allowing a more efficient attack using DECO. We report here a text extracted from the standard output.

```
ECDHE-RSA-AES128-GCM-SHA256 ECDHE-RSA-AES256-GCM-SHA384
ECDHE-RSA-AES128-SHA256 ECDHE-RSA-AES256-SHA384
ECDHE-RSA-AES128-SHA

Cipher Suite Name (OpenSSL)   KeyExch.   Encryption  Bits  Cipher Suite Name (IANA/RFC)
-----------------------------------------------------------------------
ECDHE-RSA-AES256-GCM-SHA384   ECDH 384   AESGCM   256   TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
ECDHE-RSA-AES256-SHA384       ECDH 384   AES      256   TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384
ECDHE-RSA-AES128-GCM-SHA256   ECDH 384   AESGCM   128   TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
ECDHE-RSA-AES128-SHA256       ECDH 384   AES      128   TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256
ECDHE-RSA-AES128-SHA          ECDH 384   AES      128   TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA
```

Using `openssl` to connect to `www.pt1.bfs.admin.ch` with the CBC-HMAC ciphersuite, one can see all the parameters of the established TLS session, including in particular a timeout of 7200 seconds (i.e., 2 hours), that is very large and thus beneficial for our attacks. For completeness we show here the standard output (we replace some potentially identifying data by "...").

```
openssl s_client -connect www.pt1.bfs.admin.ch:443 -cipher ECDHE-RSA-AES128-SHA

CONNECTED(00000003)
depth=2 C = BM, O = QuoVadis Limited, CN = QuoVadis Root CA 2 G3
verify error:num=19:self signed certificate in certificate chain
verify return:1
depth=2 C = BM, O = QuoVadis Limited, CN = QuoVadis Root CA 2 G3
verify return:1
depth=1 C = BM, O = QuoVadis Limited, CN = QuoVadis Global SSL ICA G3
verify return:1
depth=0 C = CH, ST = Bern, L = Bern, O = Bundesamt fuer Informatik und Telekommunikation
(BIT), OU = Swiss Government PKI, CN = www.pt1.bfs.admin.ch
verify return:1
---
Certificate chain
 0 s:C = CH, ST = Bern, L = Bern, O = Bundesamt fuer Informatik und Telekommunikation
(BIT), OU = Swiss Government PKI, CN = www.pt1.bfs.admin.ch
   i:C = BM, O = QuoVadis Limited, CN = QuoVadis Global SSL ICA G3
 1 s:C = BM, O = QuoVadis Limited, CN = QuoVadis Global SSL ICA G3
   i:C = BM, O = QuoVadis Limited, CN = QuoVadis Root CA 2 G3
 2 s:C = BM, O = QuoVadis Limited, CN = QuoVadis Root CA 2 G3
   i:C = BM, O = QuoVadis Limited, CN = QuoVadis Root CA 2 G3
---
Server certificate
-----BEGIN CERTIFICATE-----
MIIHrDCCBZSgAwIBAgIUfE2HPqjnQrTf9FnR+q9p2lVpz3cwDQYJKoZIhvcNAQEL
BQAwTTELMAkGA1UEBhMCQk0xGTAXBgNVBAoTEFF1b1ZhZGlzIExpbWl0ZWQxIzAh
BgNVBAMTGlF1b1ZhZGlzIEdsb2JhbCBTU0wgSUNBIEczMB4XDTIwMDQyMTEwNDMw
OVoXDTIyMDQyMTEwNTMwMFowgakxCzAJBgNVBAYTAkNIMQ0wCwYDVQQIDARCZXJu
MQ0wCwYDVQQHDARCZXJuMT4wPAYDVQQKDDVCdW5kZXNhbXQgZnVlciBJbmZvcm1h
dGlrIHVuZCBUZWxla29tbXVuaWthdGlvbiAoQklUKTEdMBsGA1UECwwUU3dpc3Mg
R292ZXJubWVudCBQS0kxHTAbBgNVBAMMFHd3dy5wdDEuYmZzLmFkbWluLmNoMIIB
IjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA35huk0exPmA6JIyWxvxJWWvG
cB4AkeLloyKTf0nFr5NA4QxIhX8YmksDLL+HfkY+4Kf58PuWT00li0OKpF4h9b4E
q5sOLFTJEkCmwLoZuP5E7dKFs0tXLfR/mH/5CqQAdLW8Tnhn2ZJ5YFvSDzjSOYuX
irRr5N3Y/FKrHa7ggYJKSdjvT25BIjtkimgEORAzqEwrgxgFgJ7rRqVVZ9G8I5q7
```

```
y3RIwYmJ4Qf/aF3R1iyYUxbcMQvk8G1sS/JEJ+MBALwgWXVNhswQSiOPcVtlSpNk
TGNB73SBbrzGEdozZqYQnoS378wOwuOiepY+K/hZx1D/euIe7CCxBG7XGKs+nQID
AQABo4IDJTCCAyEwCQYDVR0TBAIwADAfBgNVHSMEGDAWgBSzEom1qUs1vBUA8IDp
2HiH8RN8djBzBggrBgEFBQcBAQRnMGUwNwYIKwYBBQUHMAKGK2h0dHA6Ly90cnVz
dC5xdW92YWRpc2dsb2JhbC5jb20vcXZzc2xnMy5jcnQwKgYIKwYBBQUHMAGGHmh0
dHA6Ly9vY3NwLnF1b3ZhZGlzZ2xvYmFsLmNvbTAfBgNVHREEGDAWghR3d3cucHQx
LmJmcy5hZG1pbi5jaDBRBgNVHSAESjBIMEYGDCsGAQQBvlgAAmQBATA2MDQGCCsG
AQUFBwIBFihodHRwOi8vd3d3LnF1b3ZhZGlzZ2xvYmFsLmNvbS9yZXBvc2l0b3J5
MB0GA1UdJQQWMBQGCCsGAQUFBwMCBggrBgEFBQcDATA6BgNVHR8EMzAxMC+gLaAr
hilodHRwOi8vY3JsLnF1b3ZhZGlzZ2xvYmFsLmNvbS9xdnNzbGczLmNybDAdBgNV
HQ4EFgQU2F1pjrbfsqvpeWohTrbIn/4jRRQwDgYDVR0PAQH/BAQDAgWgMIIBfgYK
KwYBBAHWeQIEAgSCAW4EggFqAWgAdwCkuQmQtBhYFIe7E6LMZ3AKPDWYBPkb37jj
d80OyA3cEAAAAXGcXsluAAAEAwBIMEYCIQDJlsbMecVS6415SAqAI6ZqmRQLJq9M
U1dWJb/8fKJ2+gIhAJSksFLfhwuihMum/cONGOBw1SnPODhBNLW1zAGeQnggAHUA
VhQGmi/XwuzT9eG9RLI+x0Z2ubyZEVzA75SYVdaJ0N0AAAFxnF7JcgAABAMARjBE
AiBNZUPkn8ArJVVnyRlthxUILagDylovYL393EbkQKXimwIgfdrGHf5n6Sjla2CC
P96wGdbOTls19hDMlYZK3W+eIQcAdgBvU3asMfAxGdiZAKRRFf93FRwR2QLBACkG
jbIImjfZEwAAAXGcXsncAAAEAwBHMEUCIGgZ4z5MgpIlcLvBkge/BEqJ/7sYT3ze
IXMrVizfwj2TAiEAyJYfJ6D9AYxRyvDcTwnWfHXers3SNbi7s0PuX5lkWsMwDQYJ
KoZIhvcNAQELBQADggIBAC3O3iZDQYcKZ+DyAx4HSzwLpIa5yMiungbkmQuN7RYO
40pPIiADc/V/P/x+cDSuttJa8eoUq9zXEA9VW+ETFOWszf5WE31+MjasmTqDyjqV
tNyrjAGACPhbH3J9ydGQX3SqdrGNwFiBRwTvxPqukFu3+JIoRpYMzwXfbRnig1fW
R+creYRgloizGYu/M4gqV8LBwE/k7plrwTwsA8BhijhcR5asC+htRSB2SaS+teN0
ski5EJ4ajcv78vkN9y+BFKMcq3Cb5jCjoCleUoMm/BVoQNs0ZAcLJmQ3VHVeeY/S
drI9OzODQ8dSyCZKm5KDMBS2inOljLyPinbtk7JZWLQDetbW1BsjaC98BNeriYQ1
0aavGr0TPFrC9NWAB5ze3342LsTPZwiIEeGhB4AGsVTO3oYRi3yc52r22W40mdcH
f+fRgCrqddoKdPgGGp8+p9+IsWConFjzEQpN5iJRJlzorCw8nvvuKkG2I+mj9hMK
WA3/r5m0C+/ZUw+rYLpLstbCoHDBymAFFOb6P66tMl3JSvb5/Rjnhrhmtccq7o5k
Qfmr2tGrNs+8cdrVN9efn4es30fljVGP98YNtVHz7rC/nicXU5eLxHSHna4iOwlU
wpuX5DOrnF35Y0YP6eJsJS/p7y+k3L3N/iavLOj0LVIILSlTJzOZgDSbK4tG6PNk
-----END CERTIFICATE-----
subject=C = CH, ST = Bern, L = Bern, O = Bundesamt fuer Informatik und Telekommunikation
(BIT), OU = Swiss Government PKI, CN = www.pt1.bfs.admin.ch

issuer=C = BM, O = QuoVadis Limited, CN = QuoVadis Global SSL ICA G3

---
No client certificate CA names sent
Peer signing digest: SHA256
Peer signature type: RSA
Server Temp Key: ECDH, P-256, 256 bits
---
SSL handshake has read ... bytes and written ... bytes
Verification error: self signed certificate in certificate chain
---
New, TLSv1.0, Cipher is ECDHE-RSA-AES128-SHA
Server public key is 2048 bit
Secure Renegotiation IS supported
Compression: NONE
Expansion: NONE
No ALPN negotiated
SSL-Session:
    Protocol  : TLSv1.2
    Cipher    : ECDHE-RSA-AES128-SHA
    Session-ID: ...
    Session-ID-ctx:
    Master-Key: ...
    PSK identity: None
    PSK identity hint: None
    SRP username: None
    Start Time: ...
    Timeout   : 7200 (sec)
    Verify return code: 19 (self signed certificate in certificate chain)
    Extended master secret: no
---
```