Cryptographic Contact Tracing
Lecture Notes

draft of June 29, 2020

1 Introduction

Contact tracing is the process through which a person who is diagnosed with an infectious disease can notify people who he interacted with in a specific time window (i.e his contacts). The recent spread of COVID-19 has drawn attention to contact tracing as an effective mitigation of the spread, with strong interest in a way to automate the process.

An automated process would require some sort of logging to take place, as notifications need to be made to people with whom the infected interacted with in the past. As such, it is only natural that safeguards are built into such a system, so as not to leak any information other than what is strictly necessary.

We will begin with describing the protocols of such a system and their general operation, before stating security requirements and definitions, or suggesting a candidate design.

High-Level Description:

A Cryptographic Contact Tracing (CCT) system consists of a central server and three protocols:

- Contact: a protocol which is run between users (or their phones), with the purpose of (localy) recording each meeting by users of the system.
- Scan: a protocol run between a user and the server, with a purpose of notifying the user if one of his past meetings was with a newly diagnosed person.
- Notify: a protocol that allows users to notify the server that they have recently been diagnosed.

As a starting assumption, we think that the server is honest but curious, and that the notify procotol is augmented with some out of band system that verifies infection (e.g. via a registered medical professional).

2 Syntax of a CCT protocol

2.1 Contact

The contact protocol is run between users to keep track of contacts between each other.

- Initialization: $\mathsf{Contact}(1^\lambda, \epsilon) \to \mathsf{st}_0$: Initialises a new state st_0 with keysize λ
- Update: $\mathsf{Contact}(\mathsf{st}, \epsilon) \to (\mathsf{st}', \mathsf{msg}')$ When run with an empty second argument, Contact updates its state and produces a new message for broadcasting.
- Store: $\mathsf{Contact}(\mathsf{st}, \mathsf{msg}) \to \mathsf{st}'$ Upon receiving a message msg from another user, it is used to update the state.

Intended use: Alice first initialises her state, and runs update in regular intervals. At the same time she is broadcasting the last msg she has produced. When receiving a message from other users, she uses the store interface.

2.2 Scan

Scan is a protocol that is run after contacting the server to obtain a reply τ. The result depends on the servers *reported state* τ.

$$\mathsf{Scan}(vk, \mathsf{st}, \tau): \text{ output 0 or 1}$$

Intended use: When Alice wants to check if she is at risk, she asks the server for a scan file, and the sever replies with τ. Alice locally runs $\mathsf{Scan}(vk, \mathsf{st}, \tau) \to 0/1$ with 1 meaning one of her recent contacts has been diagnosed positive. The vk is assumed to be available to Alice ahead of time.

2.3 Notify

The Notify protocol used to inform the server that a user has been diagnosed as infected. It has three interfaces:

- Server Initialization $\mathsf{Notify.Server}(1^\lambda) \to (vk, sk, \tau_0)$. Server creates a verification key vk and corresponding secret key sk, and an initial stored state τ_0. The key vk is distributed to the users.
- User notifying server $\mathsf{Notify.User}(\mathsf{st}) -> \mathsf{msg}$. Produces a message that will be sent by the user to the server.
- Server message handling $\mathsf{Notify.Server}(sk, \mathsf{msg}, \tau) \to \tau'$. Server updates its state based on the user message.

Intended use: Server initialises the state. Alice, upon being diagnosed positive, runs Notify.Useron her state and produces a message to the server. Upon receiving Alices message the server updates its state.

3 Security Definitions

Informally, we can group our security requirements in three groups: correctness, integrity and privacy. We provide a brief description of the three areas before attempting a formal treatment.

- Correctness: when operated by honest users, the protocol should produce the desired outcome.
- Integrity: a malicious user is unable to produce false positives or false negatives.
- Privacy: the participants (i.e users and the server) should not receive any information apart from what is required.

3.1 Correctness

We formalize the notion as follows: when Alice runs Scan, the result is one if and only if, one of her contacts has previously run Notify.

Definition 1. *A CCT scheme* $(\mathsf{Contact}, \mathsf{Notify}, \mathsf{Scan})$ *is* ***Correct*** *if:*

$$\begin{aligned}&\Pr[(vk, sk, \tau) \leftarrow \mathsf{Notify.Server}(1^\lambda); \mathsf{st_{bob}} \leftarrow \mathsf{Contact}(1^\lambda, \epsilon); \mathsf{st_{alice}} \leftarrow \mathsf{Contact}(1^\lambda, \epsilon);\\ &\quad (\mathsf{msg}, \mathsf{st'_{bob}}) \leftarrow \mathsf{Contact}(\mathsf{st_{bob}}, \epsilon); \mathsf{st_{alice}}' \leftarrow \mathsf{Contact}(\mathsf{st_{alice}}, \mathsf{msg});\\ &\quad \mathsf{msg}' \leftarrow \mathsf{Notify.User}(\mathsf{st'_{bob}}); \tau \leftarrow \mathsf{Notify.Server}(sk, \mathsf{msg}', \tau) :\\ &\quad \mathsf{Scan}(vk, st_{\mathsf{alice}}', \tau) = 1] \approx 1\end{aligned}$$

It is also possible to allow for malicious scheduling by allowing an adversary to insert additional calls between the ones required by the experiment above (e.g that would cover a protocol that misbehaves if the contact protocol between Bob and Alice is run more than once).

3.2 Integrity

We can partition the integrity requirement in two: false positives (forcing a user's scan to output 1 when they have not contacted a user who has run Notify) and false negatives (forcing a user to output 0 when they have contacted persons running notify).

Under our envisioned setup, there exists trivial attacks that produce false negatives. Malicious users may turn off, damage, or shield their devices preventing correct data exchange. Malicious servers might discard or delay processing on Notify messages.

Given that our primary application is linked with voluntary use of an application and a government-run server, we opt to focus on false positives. In the definition, we allow the adversary to communicate freely with two honest users over a period of days. We also provide a set of dummy users named "Ian", such that ian^d will be diagnosed at the end of day d. In the base game, we do not allow the adversary to interact with the Ians. We assume that users sun Scan every night. In class, we covered "one-day" security

Game $\mathsf{CCT}_{\mathsf{INT}}^{\mathcal{U},d_{max}}(\mathcal{A},1^\lambda)$

$(vk,sk,\tau) \leftarrow \mathsf{Notify.Server}(1^\lambda); \mathsf{st_{bob}} \leftarrow \mathsf{Contact}(1^\lambda,\epsilon); \mathsf{st_{alice}} \leftarrow \mathsf{Contact}(1^\lambda,\epsilon); d = 0;$
#Morning
$d = d + 1$
$\mathsf{st}_{\mathsf{ian}^d} \leftarrow \mathsf{Contact}(1^\lambda,\epsilon)$
$\perp \leftarrow \mathcal{A}^{\mathsf{OUser}}()$
$\mathsf{msg}_d \leftarrow \mathsf{Notify.User}(\mathsf{st}_{\mathsf{ian}^d}); \tau \leftarrow \mathsf{Notify.Server}(sk, \mathsf{msg}_d, \tau)$
$(\tau', next) \leftarrow \mathcal{A}(\tau, \mathsf{msg}_d)$
If $(next = 1)$ & $(d < d_{max})$ Goto #Morning
If $(\mathsf{Scan}(vk, st_{\mathsf{alice}}', \tau') = 1)$ Return 1
Return 0

In the above game the adversary has access to the following oracle: OUser.

- The user Oracle, when called with a single argument argument $\mathsf{OUser}(u)$ runs $(\mathsf{st}_u, \mathsf{msg}) \leftarrow \mathsf{Contact}(\mathsf{st}_u, \epsilon)$ and returns msg. When called with two arguments $\mathsf{O}_{\mathsf{user}}(u,x)$ it runs $\mathsf{st}_u \leftarrow \mathsf{Contact}(\mathsf{st}_u, x)$. In both cases, if $u \notin \mathcal{U}$ it returns $\perp$ instead, without affecting the user's state.

In the lecture, we defined one-day security for a non-predicting adversary (i.e one who cannot predict that a user will be infected). We thus define one-day non-predicting CCT integrity (INT-NP1) as follows.

Definition 2. *A CCT scheme* $(\mathsf{Contact}, \mathsf{Notify}, \mathsf{Scan})$ *has 1-day non-predicting integrity if for every stateful PPT adversary* $\mathcal{A}$*:*

$$\Pr[\mathsf{CCT}_{\mathsf{INT}}^{\{\mathsf{alice},\mathsf{bob}\},1}(\mathcal{A},1^\lambda)] \approx 0$$

If we allow more than one day to pass, we can model adversaries who use the server's message to craft messages to other users. If we allow $\mathcal{U}$ to include ian, we can model adversaries using relay or replay attacks based on high-risk individuals.

3.3 Privacy

Privacy in terms of such a system can be very complex: we cover different types of users (state, healthy, infected, suspect) who might or might not be contacting each other. One way to express privacy is to state it as a negative: use the system should not reveal any information other than what is required (at a base level, informing a user that a non-zero number of their recent contacts was diagnosed).

A simpler way is to explicitly describe the kind of inferences that should not be possible in the system: i.e the state should not be able to determine if Alice has met with Bob. Towards that, we will design a game where the adversary is able to contact both users and in addition, view all communication from Alice and Bob to the server.

Game $\mathsf{CCT}^b_{\mathsf{PRI}}(\mathcal{A}, 1^\lambda)$

$(vk, sk, \tau) \leftarrow \mathsf{Notify.Server}(1^\lambda); \mathsf{st}_{\mathsf{bob}} \leftarrow \mathsf{Contact}(1^\lambda, \epsilon); \mathsf{st}_{\mathsf{alice}} \leftarrow \mathsf{Contact}(1^\lambda, \epsilon); d = 0;$
$\perp \leftarrow \mathcal{A}^{\mathsf{OUser},\mathsf{ONotify}}();$
$\mathsf{msg}_{\mathsf{alice}} \leftarrow \mathsf{Contact}(\mathsf{st}_{\mathsf{alice}}, \epsilon)$
$\mathsf{If}\ (b = 1) : \mathsf{st}_{\mathsf{bob}} \leftarrow \mathsf{Contact}(\mathsf{st}_{\mathsf{bob}}, \mathsf{msg}_{\mathsf{alice}})$
$\mathsf{msg}_{\mathsf{bob}} \leftarrow \mathsf{Contact}(\mathsf{st}_{\mathsf{bob}}, \epsilon)$
$\mathsf{If}\ (b = 1) : \mathsf{st}_{\mathsf{alice}} \leftarrow \mathsf{Contact}(\mathsf{st}_{\mathsf{alice}}, \mathsf{msg}_{\mathsf{bob}})$
$b^* \leftarrow \mathcal{A}^{\mathsf{OUser},\mathsf{ONotify}}()$
$\mathsf{Return} b^*$

In the above game the adversary has access to the following oracles: $\mathsf{OUser}, \mathsf{ONotify}$.

- When called with a single argument, $\mathsf{ONotify}(u)$ runs $\mathsf{msg} \leftarrow \mathsf{Notify.User}(\mathsf{st}_u); \tau \leftarrow \mathsf{Notify.Server}(sk, \mathsf{msg}\tau)$ and returns (msg, τ).
- The OUser Oracle, operates as in section 3.2, without ever checking inclusion in $\mathcal{U}$.

Definition 3. *A CCT scheme* $(\mathsf{Contact}, \mathsf{Notify}, \mathsf{Scan})$ *is private if for every stateful adversary* $\mathcal{A}$*:*

$$\left|\Pr[\mathsf{CCT}^0_{\mathsf{INT}}(\mathcal{A}, 1^\lambda)] - \Pr[\mathsf{CCT}^1_{\mathsf{INT}}(\mathcal{A}, 1^\lambda)]\right| \approx 0$$

4 A Simple CCT protocol

We will describe a simple CCT protocol offering a baseline level of security based on two well-known primitives:

- A digital signature scheme $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ which accepts messages in $\{0,1\}^*$.
- A Pseudorandom function $(\mathsf{KeyGen}, \mathsf{Eval})$.

Definition 4. *A Pseudorandom function (PRF) is a tuple of algorithms* $(\mathsf{KeyGen}, \mathsf{Eval})$ *with the following syntax:*

- *A key generator* $\mathsf{KeyGen} : \{1\}^* \to \mathcal{K}$.
- *An evaluation function* $\mathsf{Eval} : \mathcal{K} \times \mathcal{M}_\rho \to \mathcal{P}$.

Definition 5. *A PRF is secure if no PPT adversary* $\mathcal{A}$ *can distinguish* Eval_k *(i.e.* Eval *with the first argument fixed to* k*) from a random oracle. Concretely, for any PPT* $\mathcal{A}$*:*

$$\Pr\left[k \leftarrow \mathsf{KeyGen}(1^\lambda) : \mathcal{A}^{\mathsf{Eval}_k}(1^\lambda) = 1\right] - \Pr\left[\mathcal{A}^{RO}(1^\lambda) = 1\right] \approx 0$$

For simplicity we will assume $\mathcal{K}, \mathcal{P}$ to be $\{0,1\}^\lambda$ where λ is the security parameter and $\mathcal{M}_\rho$ to be $\{0,1\}^*$.

4.1 Contact

First, we describe the layout of a users state st. The state is a tuple $\mathsf{st} = (k, i, M)$, where k is a PRF key, i is a counter (i.e a positive integer), and M is a list of user messages m. User messages m, are tuples $m = (i, \rho)$ of a counter i and a bitstring ρ.

We now describe the opereration of the three interfaces:

- Initialization: $\mathsf{Contact}(1^\lambda, \epsilon) \to \mathsf{st}_0$: Initialises a new state st_0 by calling $k \leftarrow \mathsf{KeyGen}1^\lambda$ and setting $\mathsf{st}_0 := (k, 1, [])$.
- Update: $\mathsf{Contact}(\mathsf{st}, \epsilon) \to (\mathsf{st}', \mathsf{msg}')$ When run with $\mathsf{st} = (k, i, M)$ and an empty second argument, $\mathsf{Contact}$ runs $\mathsf{msg}' \leftarrow (\mathsf{Eval}(k, i), i)$, calculates its new state $\mathsf{st}' := (k, i+1, M)$ and returns $(\mathsf{st}', \mathsf{msg}')$.
- Store: $\mathsf{Contact}(\mathsf{st}, \mathsf{msg}) \to \mathsf{st}'$ Upon receiving a message msg from another user, it is used to update the state $\mathsf{st} = (k, i, M)$ by appending[1] msg to M i.e $M' = M + \mathsf{msg}$ and returning the new state $\mathsf{st}' := (k, i, M')$.

4.2 Notify

Notify is the protocol used to inform the server that a user has been diagnosed as infected. The servers *reported state* τ is a tuple (D, σ) where D is list of PRF keys, and σ is a digital signature.

We describe the three interfaces:

- Server Initialization $\mathsf{Notify.Server}(1^\lambda) \to (vk, sk, \tau_0)$. The Server runs $(vk, sk) \leftarrow \mathsf{Gen}(1^\lambda)$, obtaining a public key vk and corresponding secret key sk. The initial server state is $\tau_0 := ([], \epsilon)$. The key vk is distributed to the users.
- User notifying server $\mathsf{Notify.User}(\mathsf{st}) -> \mathsf{msg}$. Given a user state $\mathsf{st} = (k, i, M)$, returns the message $\mathsf{msg} = k$.
- Server message handling $\mathsf{Notify.Server}(sk, \mathsf{msg}, \tau) \to \tau'$. Server updates $\tau = (D, \sigma)$ by appending the message containing the user key, i.e. $D' = D + msg$, producing a new signature $\sigma' = \mathsf{Sign}(sk, D')$ and setting $\tau' = (D', \sigma')$.

4.3 Scan

Scan is a protocol that is run after contacting the server to obtain a reply τ.

- $\mathsf{Scan}(vk, \mathsf{st}, \tau)$: Parse τ as D, σ. If $\mathsf{Verify}(vk, D, \sigma) \neq 1$, abort. Otherwise, parse D as $k_1, k_2, \ldots, k_n$ and M as $(\rho_1, i_1), (\rho_2, i_2) \ldots (\rho_m, i_m)$. Then calculate all possible $\rho_{j,l} = \mathsf{Eval}(k_j, i_l)$ for $j = 1 \ldots n$ and $l = 1 \ldots m$. If, for any j, $\rho_{j,l} = \rho_l$ then return 1, otherwise return 0.

[1] We slightly abuse notation by using + for appending.

5 Security

5.1 Correctness

Theorem 1. *The CCT scheme of section 4 is correct given any PRF* (KeyGen, Eval)*, and any correct signature scheme* (Gen, Sign, Verify)*.*

Proof. We note that when Bob contacts Alice, his state is $\mathsf{st}_{\mathsf{bob}} = (k_{\mathsf{bob}}, i_{\mathsf{bob}}, M_{\mathsf{bob}})$. This results in $\mathsf{msg}_{\mathsf{bob}} = (\mathsf{Eval}(k_{\mathsf{bob}}, i_{\mathsf{bob}}), i_{\mathsf{bob}})$ being appended in Alice's list M_{alice}. We also note that no operation[2] can remove items from M_{alice} or alter the value of k_{bob}. This implies that: at the time of Bob running Notify.User, k_{bob} is appended to D, from which no operation will remove it.

Thus, when Alice runs Scan, she obtains a signed list D that contains k_bob. By the correctness of the signature scheme, the signature σ will verify. Because k_bob is in D, there exists l such that $k_l = k_{\mathsf{bob}}$. Because $\mathsf{msg}_{\mathsf{bob}}$ is in M, there exists j such that $i_j = i_bob$. Thus, there exists an l, j pair such that $\rho_{l,j}$ is contained in M. □

We will prove 1-day security for a non-predicting adversary. A non-predicting adversary is one without access to soon-to-be-diagnosed users.

5.2 Integrity

We will show that a PPT adversary with oracle access to the PRF (with no access to Notify.User) cannot force Alice's Scan to return 1. We will describe a reduction to the unforgeability of our signature scheme. The main intuition is as follows: if the adversary passes the server's scan message unaltered the probability of Alice outputting 1 is negligible . If the adversary alters the message, he contradicts existential unforgeability. We begin with the first claim.

Theorem 2. *If,* (Gen, Sign, Verify) *is existentially unforgerable under chosen message attack, the CCT scheme of section 4 has 1-day integrity against non-predicting adversaries with oracle access to the PRF.*

Proof. We start by assuming the adversary chooses τ' such that $D' \neq D$.

We build a reduction $\mathcal{B}$ that plays the role of the adversary in the unforgeability game and the challenger in the integrity game. The reduction starts the unforgeability game by obtaining a verification key vk that will be passed to $\mathcal{A}$ as the server verification key. The reduction is able to play the parts of all users in the system, with the exception of creating τ by signing D. Fortunately, it can use its signing oracle from the unforgreability game. When $\mathcal{A}$ returns $\tau' \neq \tau$, the reduction checks if the signature is valid and if it is, offers it as a forgery in the unforgeability game. If not, it aborts.

The reduction clearly runs in polynomial time. We now evaluate the probability of it succeeding in the unforgeability game. We claim that the probability of $\mathcal{B}$ wining the unforgeability game is at least equal to that of A wining the CCT integrity game. If $\mathcal{A}$ wins, that implies that the signature on τ' verified, i.e $\mathcal{B}$ has produced a new signature that verifies under the challenge key vk.

[2]The proof also holds if we allow an adversary with an OUser oracle in the experiment.

To complete the proof we also need to handle the case where the adversary's choice of D' does not allow us to complete the reduction, i.e $D = D'$.

Claim 2.1. *If the adversary sets τ such that $D' = D$, then the probability that $\mathsf{Scan}(vk, st_{\mathsf{alice}}', \tau') = 1$ is neglibible.*

Proof. We first note that the list D inside τ contains exactly k_{ian}, which is unknown to the Adversary[3]. Thus, the goal of the adversary is to send Alice a message ρ, i such that $\rho = \mathsf{Eval}(k_{\mathsf{ian}}, i)$. We note that the adversary's view is independent of k_{ian}. We assume that the Adversary made q_e queries to Eval, and (pessimistically) assume he wins if he queries on k_{ian}. This event only happens with probability $q \cdot 2^{-\lambda}$, and as q_e is polynomial in λ is negligible. What remains is to check the probability that Alice's evaluations will match an adversarial message, on the condition that the adversary never queried k_{ian}. We assume the adversary sent q_m messages. Alice will perform at most q_m evaluations, obtaining at most q_m random values, each to be checked against the adversary's q_m messages. The probability of a match is bounded by $q_m^2 \cdot 2^{-}\lambda$. Letting $q = q_m + q_e + 1$ we have that the adversary's probability of success is $q \cdot 2^{-\lambda}$ which is negligible. □

The claim completes the proof. □

Stronger Adversaries At this point, we note that without any modifications we cannot prove security for multiple days, or against predicting adversaries. Multiple days allow the adversary to obtain a known-infected key from the server and use that to craft malicious messages. In practice this can be mitigated by making the current time available to the protocol. Predicting adversaries are able to contact Ian (who will end up being diagnosed) and replay his messages towards Alice. This can be (partly) mitigated by challenge-response protocols (with appropriate changes to the syntax). Relaying attacks are harder to mitigate, but may be mitigated by introducing a location parameter to the protocol.

5.3 Privacy

Theorem 3. *The CCT scheme of section 4 is private against any[4] adversary $\mathcal{A}$ playing the role of the server.*

Proof. We point out that the view of the adversary is not dependent on the value of b. The only thing that depends on the value of b in the experiment are the contents of the M lists inside $\mathsf{st}_{\mathsf{bob}}$ and $\mathsf{st}_{\mathsf{alice}}$. However, that part of the state is never read within the protocol, but only appended to. Since the view of the adversary is independent of b, its output will be independent as well. This implies that $\Pr[\mathsf{CCT}^0_{\mathsf{INT}}(\mathcal{A}, 1^\lambda)] = \Pr[\mathsf{CCT}^1_{\mathsf{INT}}(\mathcal{A}, 1^\lambda)$, thus the scheme is private. □

[3] Technically the Adversary learns it, but can no longer message users.

[4] The definition requires the adversary to be semihonest i.e honest but curious.