DP-3T: Ensuring Cryptographic Security and Privacy In COVID-19 Contact Tracing

Isha Agarwal and Minseo Kim
Mentor: Talia Blum

June 2021

1 Introduction

COVID-19 has devastated the lives of millions of people around the globe. As of June 2021, approximately 176 million people have contracted the virus and about 3.8 million people have died from the virus [9]. While no one of a specific stature or walk of life is immune to the virus, demographical statistics show healthcare disparities and uneven COVID-19 case distributions across different racial communities [5].

To combat the virus, virologists have recommended several measures one can take, including wearing a mask and social distancing. However, once someone has tested positive for COVID-19, *contact tracing* can help curtail the spread of the virus.

Contact tracing is defined as the process of identifying persons who may have come into contact with an infected person, and the subsequent collection of further information about these contacts. According to the CDC, "Contact tracing is a key strategy to prevent the further spread of COVID-19" [8].

Countries have taken different routes to facilitate contact-tracing: informational check-in apps, location monitors, and modified health-related security and publicity policies. But as countries continue facilitating testing, contact tracing, isolation, and quarantine, there have been public controversies over the methods used to facilitate contact tracing. We can take the instance of South Korea in the following case study.

Example 1.1. In 2019, after the first waves of COVID-19 cases, South Korea responded with healthcare policies prepared since the 2015 MERS outbreak. These policies included governmental transparency and revamping medical center supply chains and materials. As per transparency, they also used contact tracing software: individual software developers like Bae Won-Seok and Lee Jun-Young created applications like Corona 100m and Corona Map respectively, which ingeniously provide real-time updates with visual mapping [24]. The government, hospital, and district-based broadcasting systems have information on

close-contact information store based on individuals' specific device locations and exchanged signals.

However, by sacrificing privacy for the health of the nation, open location disclosure and descriptions led to an up-rise in online bullying, harassment, and slander [20]. Furthermore, the new application models which move through a second-party filter to analyze government information and software structures are susceptible to hacking and timed attacks, meaning the applications threatened people's security as well.

As highlighted by Example 1.1, maintaining privacy as well as security is essential, and a challenge of digital contact tracing. This paper will deconstruct the current contact tracing methods by pinpointing weaknesses and proposing solutions to reinforce their security.

In Section 2, we explain how the *Decentralized Privacy-Preserving Proximity Tracing* or DP-3T algorithm for digital contact tracing works to retain the anonymity of its users. Section 3 offers a security analysis of DP-3T by determining weaknesses and strengths of both the overall algorithm and its specific parts. Finally, in Section 4 we propose solutions that could mitigate the effect of the vulnerabilities discussed in Section 3.

2 DP-3T Algorithm

The DP-3T algorithm allows for digital contact tracing while protecting users' privacy. The model of the algorithm presented in this paper is drawn from [22]. Apps that utilize this algorithm are currently available on smartphones in Austria, Belgium, Croatia, Germany, Ireland, Italy, the Netherlands, Portugal, and Switzerland [14]. We begin with an overview of the algorithm before delving into its specific parts.

2.1 Overview

First, the algorithm generates random keys or codes and takes note of them. Next, when two smartphones come in close contact with each other for an extended period of time, they exchange these keys using Bluetooth. During this exchange, the phones keep track of all codes that they *say* or send to the other smartphone and *hear* or receive from the other smartphone.

If someone tests positive for COVID-19, then all the codes which their phone said are uploaded to a *back-end server* or *hospital database*. All smartphones regularly check this database to see if any codes it heard pop up. If a code the phone heard shows up in the database, then the app alerts the user that they are a close contact and should quarantine [22].

Let's look at an example to better understand how this algorithm functions.

Example 2.1. Let Avia and Bai be two people who have downloaded the app. Everyday, their phone will generate random keys. Avia and Bai go to a cafe where they sit six feet apart from each other for fifteen minutes. Every ten

minutes, their phones communicate, via Bluetooth, exchanging their respective random keys. Both phones will keep track of which keys they heard and said.

The next day, Bai does not feel very well, and she tests positive for COVID-19. She alerts the app, and her phone immediately posts her secret keys on the hospital database. Avia's phone will check this database periodically and compare it against keys it heard. It will see Bai's code and realize Avia came in contact with someone with COVID-19. As a result, it will notify Avia that they are a close contact, and they will quarantine.

Note that no personal information about Avia or Bai is required throughout the entire process. Furthermore, the use of secret keys allows Bai to keep her COVID-19 diagnosis private from others since she remains completely anonymous to Avia.

There are three key parts to the algorithm: generating the random keys, exchanging and storing information between the devices, and reporting infections. In the following subsections, we break down how each of these three parts function in more detail.

2.2 Generating Random Keys

There are three steps in the process of generating random keys [22]. The algorithm for generating these keys inputs the current day and outputs the random keys for that day.

Step 1: Calculating the Secret Key. Let t be the current day. The phone will first produce a *random initial daily seed* or *secret key*, SK_t, which will be used to produce the *ephemeral identifiers* (EphIDs). EphIDs are the random messages the algorithm will say to communicate with other smartphones. The value SK_t is calculated by the following:

$$SK_t = H(SK_{t-1}),$$

where H is a *cryptographic hash function* that maps SK_{t-1} to the value SK_t in a way that is difficult to guess what SK_{t-1} is [15]. The initial value, SK_0 is generated using a secret key algorithm [22]. These steps ensure that SK_t is created securely to prevent the user from being traced.

Step 2: Producing String of EphIDs. Next, the algorithm uses SK_t to produce the EphIDs. If the EphID changes every L minutes, a total of $n = (24 \cdot 60)/L$ new EphIDs must be produced every day. In Example 2.1, since the devices communicated every 10 minutes, we have $L = 10$, and each phone would have to create $n = (24 \cdot 60)/10$, or 144 EphIDs every day. In order to create all the necessary EphIDs, the algorithm takes SK_t as an input and outputs a string with all the EphIDs. The device computes:

$$E = PRG(PRF(SK_t, P)), \tag{2.2}$$

where PRF is a pseudo-random function that produces random numbers, PRG is a pseudo-random generator that turns a random seed into a longer pseudo-

random string, and P is a fixed public string. Note that this process is deterministic, so inputting the same value of SK_t will produce the same value of E.

Step 3: Turning the String into Individual EphIDs. The computation in Equation (2.2) prepares E, which is a string of all the EphIDs. E has length $16n$, and it is split into n 16-bit sections that are the n EphIDs. The device chooses a random order to broadcast the EphIDs for L minutes each.

This process allows for the creation of randomized EphIDs, which do not convey any information about the user and thus protect their privacy. Furthermore, EphIDs are changed regularly to prevent the user from being traced by them, protecting their security.

2.3 Device Interaction and Data Storage

Figure 1: Avia and Bai's phones interact and exchange EphIDs over Bluetooth. Figure created using [12] and [25].

Next, smartphones communicate via Bluetooth Low Energy (Bluetooth LE)[1] to exchange their respective EphIDs, as shown by Figure 1 [22]. The random letters said by each phone in Figure 1 represent EphIDs.

Phones send out *beacons* conveying its EphID to other phones, represented by the speech bubbles in Figure 1. The device receiving the beacon stores:

1. The EphID,
2. A measurement of the exposure, and
3. The date.

In order to measure the exposure risk, the phone stores the strength of the Bluetooth signal as an indicator of how far away the other person was. In

[1] See Section 3.5 for a more thorough explanation of Bluetooth LE.

Example 2.1, this measurement would include that Avia and Bai sat 6 feet apart for 15 minutes. Data is stored by EphID to maximize efficiency (since there could be multiple received beacons for each EphID). An estimate of the storage space required is 6.1 MB [22].

Furthermore, each phone stores its SK_t value for the past 14 days, the same length as the incubation period for the COVID-19 virus. This ensures that a user can upload all the SK_t values from the period of time they were infectious should they test positive for COVID-19.

2.4 Infection Reporting

Finally, if someone contracts COVID-19, the algorithm has a secure method of reporting infections to preserve privacy [22].

Step 1: Notify Application of a Positive Result. First, the user must report to the app that they have tested positive for COVID-19. The positive test result is self-reported, but see Section 4.1 for a method of verifying the user actually has COVID.

Step 2: Upload Secret Key Values. Then, the SK_t values for all the days when the user was contagious, which could be determined by a medical health expert or the user themselves, are sent to the back-end server.

Step 3: Delete Secret Keys from User Device. After reporting the values of SK_t, the user's phone deletes those seeds and restarts the process of generating random keys. Since the next value of SK_t is dependant on the previous one, this step prevents an attacker from tracking users by figuring out their current SK_t value and EphIDs.

Step 4: Devices Check Back-end Server. The server then reports the values (SK_t, t), which all phones with the app download. From this ordered pair, the device can derive the user with COVID-19's EphIDs during the time when they were infectious. The device is able to compute the EphIDs from the secret key value because the process of creating these EphIDs is deterministic. These EphIDs are then cross-checked to the local database of EphIDs the device heard to determine if the user was a close contact. Note the value t is necessary to report because the app must make sure it came in contact with an EphID before SK_t was made public to avoid being tricked by an attacker broadcasting those EphIDs after they are published.

Step 5: Alert Close Contacts. Finally, the algorithm alerts the user if they were a close contact and what their exposure measurement is. With this information, the user can then quarantine to mitigate the spread of the virus.

3 Security Analysis

In this section we take a closer look at DP-3T and its vulnerabilities and strengths with respect to security and privacy. We begin by looking at different impact scenarios in which the algorithm's security is breached in Section 3.1.

Next, we look at different attacker capabilities in Section 3.2. Then, we do a general analysis of the algorithm in Section 3.3.

In the next few subsections, we deconstruct the different parts of the algorithm mentioned in Section 2. In Section 3.4 we analyze the process of uploading secret keys to the back-end server. We then look at security of the security of Bluetooth Low Energy, the method the application uses to transmit information in Section 3.5. Next, we look at vulnerabilities pertaining to individual devices in Section 3.6. Finally, in Section 3.7 we look at other miscellaneous issues with the algorithm.

3.1 Problem Cases and Impacts

While analyzing the DP-3T program's security, we must also measure the impact calculus of the situation and what problems arise in the following scenarios.

1. **Close Contacts Not Notified.** In this scenario, the user is a close contact but they are not notified. Given particular close contacts are not notified, the virus continues to spread and the contact tracing objective of the algorithm is not achieved.

2. **False Positives.** The user is not a close contact but they are notified. This means the user will have to go through testing and possibly quarantine procedures which takes time and energy. For some users, this could be detrimental to their working status and financial situation.

3. **Denial of Service.** In this case, as the hackers upload mass entries of random strings and EphIDs, it may overload the server and block out the "true" EphIDs. Users are ultimately unable to access and use the application.

4. **Tracing Users.** If an attacker is able to figure out a user's secret key and generate all their EphIDs from that secret key, they can predict which EphIDs the user will say throughout the day. This allows the attacker to track the user whenever one of these EphIDs appear.

3.2 Attack Models

An *attack model* describes the attacker's capabilities. For example, an attacker with access to the app poses a different security threat than an attacker who can only see people's EphIDs. For each of the following attack models, we explain the attacker's abilities as they relate to the algorithm [22]:

Definition 3.1. A *passive attacker* is an attacker who can only view information but cannot change it in any way.

Definition 3.2. An *active attacker* can manipulate information in some way.

Definition 3.3. A *regular user* only has access to the application as it appears to all ordinary users.

Definition 3.4. An *eavesdropper* is a passive attacker that can "eavesdrop" or intercept communication but cannot disrupt it.

Definition 3.5. A *whitehat hacker* is a passive attacker that can view the application's code, information stored on a user's phone, and communication with the back-end server.

Definition 3.6. A *malicious hacker* is an active attacker able to share EphIDs, make adjustments to the protocols, and overload the system so that it is unavailable to users.

Depending on how active or passive an attacker is, different security standards must be upheld. In the subsections that follow, we will see how DP-3T fares against these attack models.

3.3 General Analysis

Even though DP-3T preserves location privacy, there are still vulnerabilities present. The DP-3T algorithm is *informationally insecure* because an attacker with infinite computing power could crack the algorithm. Such an attacker could implement a brute force method, testing all the possible keys and comparing them against the received EphIDs. While there are multiple keys that produce the same EphIDs, the attacker can test all of these keys to track the person.

This brute force algorithm has a time complexity of 2^n, where n is the length of SK_t. For large values of n, this is impractical to calculate with current technology. Moreover, with the added restraint that an attacker only has a 24-hour time slot to discover the secret key before it changes, it is impossible for them to figure out a user's secret key.

However, with the advent of quantum computing, DP-3T could be put at risk. Quantum computers have *qubits* that can store much more information than a normal computer bit. This allows for significantly faster computing times, and could make a brute force algorithm feasible. Fortunately or unfortunately, quantum computers are still far away from becoming a reality [2]. Nonetheless, in Section 4.7 we describe some of the ways DP-3T can be fortified in a post-quantum world.

On the other hand, DP-3T is an *indistinguishable* protocol, which means the information being sent out (EphIDs) is indistinguishable from a random string. Indistinguishability increases the security of the protocol because if an attacker was given values of SK_t and EphIDs, they would not be able to tell which EphID matches with which value of SK_t. Thus, additional information would be required to extract values of SK_t from the EphIDs, making it significantly harder for an attacker to do so.

3.4 Back-end Server Vulnerabilities

DP-3T's decentralized system means that positive results for COVID-19 are self-reported. Thus, an attacker could lie about having COVID to have their

key uploaded to the hospital database. This would notify people who are not close contacts to quarantine.

The case study CVE-2020–15957 revealed an oversight in the code for the back-end server which allowed people to upload their secret keys to the hospital database without authorization [7]. DP-3T uses a *JSON Web Token* (JWT) to exchange information. JWT is secure because it requires a digital signature. However, it is not entirely impervious to attacks. There is a "none" algorithm in JWT that assumes the information being transmitted was already verified. When an attacker uploads a secret key with the label "none," they are able to upload secret keys to the back-end server without the proper authorization. This means that even with methods to verify if someone has tested positive COVID-19 (such as the one explained in Section 3.5) there are loopholes attackers can use to continue uploading faux secret keys.

There are also more invasive attacks that an attacker can carry out. An advanced attacker could hack into the hospital database and change or delete some of the codes, thus exploiting patients' medical data and secret keys. This reflects one of the drawbacks to randomness: because the codes are random, if someone were to change or add a code there would be little to no way of telling. The impact of this would be even more grave, as someone who is actually a close contact would not be alerted.

3.5 Connection Encryption Security Analysis

The usage of Bluetooth LE in DP-3T, which uses AES-CCM encryption, ensures further security. The counterpart E0 Bluetooth communication algorithm for data transmission is a 128-bit *symmetric stream cipher*. Before moving further, let's go over some important terms from [2].

Definition 3.7. A *plaintext* is the original message prior to any encryption.

Definition 3.8. A *ciphertext* is the result of a plaintext being encrypted by an encryption algorithm or cipher.

Definition 3.9. In *symmetric encryption*, a private key is shared among both the sender and recipient and is used to both encrypt and decrypt a message.

Definition 3.10. In *asymmetric encryption*, or public-key encryption, a user can distribute their public key widely and keep their own private key. Other parties can use the public key to encrypt messages and the user can use the private key to decrypt these messages.

Definition 3.11. A *pseudorandom cipher digit stream*, or a key stream, is generated from a random seed value using digital shift registers and is statistically defined as random but produced through exact mathematical procedures.

Definition 3.12. A *stream cipher* is a symmetric key cipher where plaintext digits are combined with a pseudorandom cipher digit stream in order to produce a ciphertext.

Each encryption is based off of the cipher's current state and the combination operation for digits use the exclusive-or (XOR) operation. In essence, the 128-bit data file encryption technique is a popular method in encryption algorithms, but cryptoanalysts have determined that the Bluetooth protocol could be breached by time complexity of 2^{64} operations. E0 is vulnerable to certain plaintext attacks.

On the other hand, Bluetooth LE uses a combination of Advanced Encryption Standard (AES) and Counter With Cipher Block Chaining Message Authentication Code (CBC-MAC or CCM mode) which provides authentication methods while remaining secure and keeping data confidential.

3.6 Individual Vulnerabilities

Further downsides lie in the person-to-person device handling and personal data security, where hackers may intercept the Bluetooth broadcast and alter the "heard" and "said" random strings. As a result, one who may have been a close contact is not listed as one.

A third party or hacker can hijack the Bluetooth connection between two parties and operate a Man-in-the-Middle (MitM) attack. MitM is a tactical method used to secretly intercept communications and eavesdrop or modify traffic between two targets, re-encrypting values along the way to remain unnoticed [21].

Example 3.13. In this scenario, Avia and Bai are exchanging data over Bluetooth at the cafe. What they are unaware of is that eavesdropper Eve is listening in to the connection from a few tables away and recording the exchanged data.

Eve can eavesdrop on the conversation in a few ways.

- **Networking**. Eve may have interfered with legitimate networks or created fake networks to control. Compromised traffic can be stolen, altered, or rerouted.
- **SSL Stripping**. Eve can establish an HTTPS connection between herself and the server but set up an unsecured HTTP connection with the user, which means information doesn't become encrypted.
- **Evil Twin Attack**. Eve can emulate a legitimate WiFi networking site and control data flow so that she can collect large amounts of data between two parties.

Currently, most devices use the AES-CMM Bluetooth protocol. Although it is one of the most reliable encryption techniques amongst modern encryption methods, it is still susceptible to cyber-attacks like some versions of the MitM attack. Simply listening in on a connection and harvesting data may not sound dangerous as it does not actively augment the values or cut off data flow, but a third party can utilize these data records to backtrack and pinpoint the EphID re-seeding algorithm or encryption security mechanism.

The input, which we'll call PT, is XOR'ed with some unknown number of bytes into the system, and thus difficult to control. We will use the $\oplus$ symbol to denote the XOR operation. As one continues to attack the same block by resetting the device to reset the AES state and run through the first block multiple times, the input generation pattern is solvable as the number of unknown bytes are constant. The third party can mathematically determine the encryption algorithm for an upper layer. Although one cannot determine the value of encryption key k, $k \oplus CBC_{m-1}$ is still determinable, where CBC_{m-1} is the output of the previous-block *ciphertext*. All constant inputs are run into a modified key, then used to determine the second-round "true" key [19].

Before a more detailed explanation of the attack sequence, we must go over some terminology used in the attack sequence to crack an AES-CCM [10].

Definition 3.14. An *inverse cipher* is the function which reverses the transformation of a given cipher with the same cryptographical key.

Definition 3.15. A *block cipher* applies a deterministic algorithm with a symmetric key to encrypt a text block rather than encrypting one nit at a time like stream ciphers (explained further in 3.5).

Definition 3.16. A *round key* is one of the ten outputted permutations when the AES algorithm expands the main key to multiple parts by a key schedule round. Adding these 128-bit keys is what makes the algorithm a block cipher.

Definition 3.17. The *AddRoundKey* function is what transforms the cipher and inverse cipher by adding a round key using an XOR operation. It is the only stage in AES encryption which directly interacts with the round key.

The AES-CCM attack works through the following steps.

1. One recovers the value $k' = k \oplus CBC_{m-1}$.
2. AddRoundKey is applied to the AES algorithm to get $\text{AddRoundKey}(a, b) = \text{AddRoundKey}(k, CBC_{m-1} \oplus PT) = a \oplus b$, or simply $\text{AddRoundKey}(k', PT)$ for the modified key.
3. Finally, roll back the key schedule to determine the first-round key in relation to the second-round key.

In summary, one can attack the AES algorithm's second round and determine the inputs based on our modified key to then figure out the known plaintext values and uncover the algorithmic encryption pattern for the first round.

3.7 Additional Issues

We can find further potential inefficiencies in DP-3T regarding (1) its vulnerability to targeted identification attacks, (2) potential problems of Pseudorandom Number Generators (PRNG) which Bluetooth LE uses, and (3) possible "false positive" collisions.

Regarding (1), researchers have carried out theoretical attacks on DP-3T. Hackers can bypass the earlier DP-3T models by tracking users who voluntarily upload their identifiers. Once a third party amasses Bluetooth LE devices around the target, as long as the target is part of a centralized system for contact-tracing, devices exchange user identifiers and the hacker can collect the ones transmitted by the target to disperse elsewhere. Although such methods don't work with newer models where the identifiers are relayed through encased states of keys or seeds, the method highlights problems regarding hackers exploiting Bluetooth linkability to disrupt normal exchange of identifiers.

To expound on the PRNGs from (2), any lacking in the following traits would mean the PRNGs are deterministic to a point they would fail a statistical pattern-detection test: relatively short periods for certain "weak" seed states (DP-3T changes the Eph-ID's 144 times per day); lack of distribution uniformity and equal probability per each possibly value, also known as low *entropy*; correlation between consecutive values; poor dimensional distribution of output sequence.

While Bluetooth LE uses a *cryptographically secure* PRNG (CSPRNG), short periods and improving computational algorithms risk the efficacy of pseudorandom generators.

Finally, (3) expounds on RPI collision cases over Bluetooth. The Apple-Google protocol [13], a similar program to DP-3T, starts by generating a Temporary Exposure Key (TEK), one per day, and passes its TEK and current time block organized by 10's to a central server. Other devices can download the TEKs and times from the central server and check their RPIs against ones they listened to throughout the day.

The problem appears once a TEK-timestamp pair gets hashed to the same RPI as another user's, both ending with the same RPI. 16-byte identifiers yield low probability for collisions while keeping device data storage requirements low, but does not prevent collisions from occurring at all. Let's bring Avia and Bai back into the picture. Avia's TEK-timestamp pair and Bai's derive the same RPI. If Avia tests positive for COVID-19, their TEK-timestamp information will enter the Diagnosis Keys. As Bai's data codes to the same RPI, she will be given a false positive.

4 Proposed Solutions

In this section, we analyze solutions as a well as offer some of our own solutions to some of the security concerns for DP-3T mentioned in the previous section.

4.1 Positive Test Verification

Additional protocols could be put in place to verify a user has actually tested positive. This prevents users from arbitrarily uploading secret key values to the back-end server, resulting in false positives. One possible protocol that is described in [23] utilizes an *authorization code*.

Step 1: Getting Tested. As part of the testing process, users are given an authorization code. This could either be generated by their device or given to them by a health official. If it is the former, the user must give this code to the health official when getting tested. This authorization code remains inactive for now.

Step 2: Positive Result Secret Key Upload. If the user tests positive for COVID-19, the health official notifies them and activates their authorization code. Then, the user will give the device the authorization code. Once it is accepted by the back-end server, the device uploads the user's secret keys for the time when they were contagious (which they can determine with the help of the health official). The user's phone deletes the authorization code after completing the upload.

Disrupting this method requires attackers or users who have considerable skills. A long enough authorization code makes a brute-force attack method impossible. However, an attacker could steal a user's authorization key as it is on their phone for a long window of time while they are waiting for their test results. This could allow them to upload fake secret keys should the user test positive for COVID-19.

The authorization code can be hidden from the user throughout the entire process, which prevents the average user from changing the code or uploading different secret keys from another device. However, the system is not entirely infallible; a user with strong technical skills could find and edit the code.

Nevertheless, this verification algorithm prevents any malicious actions on the part of the health official. It also preserves the user's privacy since the authorization code is used as soon as it is sent to the back-end server. This prohibits an attacker from tracking a user through their authorization code.

4.2 Fortifying the Back-end Server

An algorithmic defense mechanism can be put in place for a hospital database in order to avoid attackers accessing centralized bounties of data. Five general steps can be taken in order to increase the cybersecurity levels of hospital databases [6].

- **Endpoint Security Layer**. All staff devices and back-end servers must use cybersecurity tools, including those from TrendMicro, Symantec, Microsoft and Cisco to provide protection.
- **Network Security Layer**. Hospitals should monitor closely and determine whether an attack has happened, either through an inspired version of the BB-84 [2] or through scanners like Nessus, Microsoft Advanced Threat Analytics and Cisco Stealthwatch.
- **Perimeter Security Layer**. Firewalls and basic public-facing security layers must be in place to avoid hackers accessing the database section entirely.

[2] See Section 4.7.

- **Application Security Layer**. There should be security policies and procedures in place and facilitated by the hospital IT department.
- **Physical Security Layer**. Physical access will risk important data and network access. To prevent a physical breach, hospitals must follow HIPAA guidelines for physical security safeguards and control access to server and network appliances with protected health information like contact cases and secret keys.

4.3 Dummy Traffic

Sending *dummy traffic* involves sending fake messages to mask the true information being exchanged. It could be an effective strategy to prevent an eavesdropper from compromising the security and privacy of DP-3T. This strategy could be used at two parts of the algorithm: sharing EphIDs and uploading secret keys to the back-end server.

When sharing EphIDs, sending dummy traffic would mean sharing real as well as fake EphIDs. The device and back-end server would have a method of differentiating between random and fake EphIDs when recording the data. However, to an eavesdropper all codes shared would appear to be random 16-bit strings, and thus they would not be able to glean anything from intercepting these EphIDs.

Another key interaction is when COVID-positive users upload their secret keys to the back-end server. Eavesdroppers could easily figure out which users are COVID-positive as well as their secret keys when these users upload to the hospital database. This interaction is even more crucial than the prior one; if an attacker can determine which users are COVID-positive, this destroys the algorithm's privacy protection. Furthermore, a malicious hacker could tamper with that user's information and prevent close contacts from being notified. To avoid these scenarios, all users can upload keys to the hospital database, but most of them would be fake [22].

The fake data that is created must appear randomized. This way, a simple pattern analysis would not reveal which keys are fake and which ones are real. A secure encryption model like the one we present in Section 4.5 would help randomize the appearance of the fake keys. However, a skilled eavesdropper might be able to determine the mechanism and figure out the true information being sent. Nonetheless, this would take considerably longer and require an attacker with more skills and resources, meaning that sending dummy traffic could still be an effective strategy against a range of attackers.

4.4 Derivation of Secret Key Values

In the DP-3T algorithm, each secret key is derived from hashing the previous one. Therefore, just discovering one of these secret keys could allow an attacker to figure out the ones from that day on wards. In April of 2020, Apple and

Google proposed an Exposure Notification (EN) algorithm similar to DP-3T that remedies this problem [13].

The EN algorithm randomly generates a key every single day instead of the different keys being dependant on one another. This way, if an attacker got one of the keys, it would only be active for a 24-hour period or if the user tests positive for COVID and that key is uploaded to the back-end server.

While the EN algorithm does provide a user a little more security than DP-3T, it does have its cons as well. A user must upload all values of SK_t to the back-end server instead of simply the SK_t value for the first day they were contagious. Aside from taking up more space and thus requiring more resources, a larger data upload is more likely to be intercepted or disrupted by an attacker because it takes a longer amount of time. Therefore, while EN might be more effective than DP-3T at ensuring a user's privacy at the individual level, its benefits might not necessarily outweigh the costs.

4.5 AES-RSA Encryption

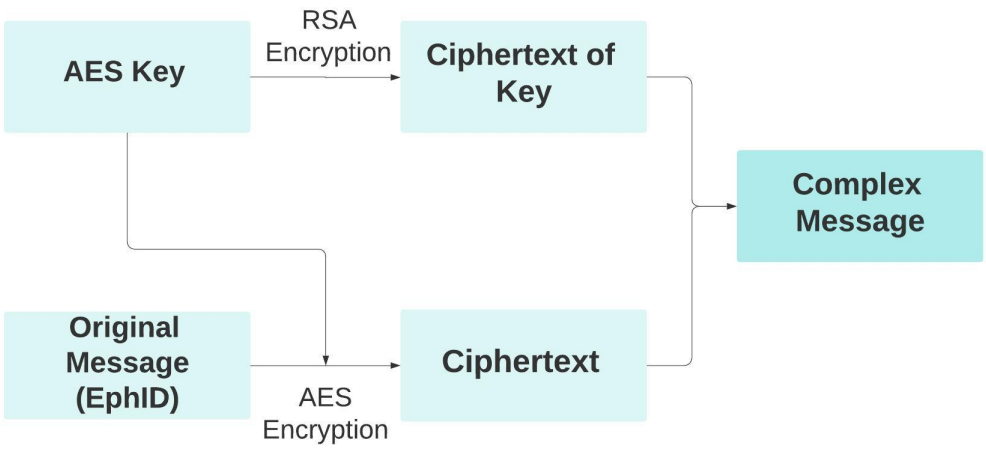

Figure 2: The AES-RSA encryption scheme uses AES to encrypt the original message and RSA to encrypt the AES key, forming a complex message. Figure adapted from [18].

As explained earlier in Section 3.5 and Section 3.6, the Bluetooth LE AES-CMM encryption method is effective but still faces attack vulnerabilities. We can strengthen the security of AES-CMM by implementing RSA.

The *Rivest-Shamir-Adleman* (RSA) algorithm is one of the most widely-used encryption algorithms for much of our important information, such as bank accounts, mail, and online transactions. Cracking RSA is at least as hard as factoring very large numbers, a computation that takes hundreds of thousands of

years with modern technology. Hence, RSA is practically impossible to decrypt and thus a secure encryption scheme [2].

Part of RSA's security comes from the fact that is an asymmetric method, so the key used to encrypt messages can be shared publicly, whereas the key used to decrypt is only known to the person receiving messages. Since the private key is never exchanged, RSA's security is much less vulnerable to attack compared to a symmetric encryption scheme in which a secret key has to be exchanged [2]. Hence, combining RSA's asymmetric method with AES would be highly efficient.

The process described below is from [18].

As shown in Figure 2, AES is used to encrypt the EphID since AES encryption requires less time than RSA. This creates the ciphertext of the EphID. However, since AES is a symmetric encryption scheme, the recipient needs to be sent an AES key in order to decrypt the EphID. We use RSA to encrypt the AES key, forming the ciphertext of the key. The recepient is then sent a *complex message* which includes the ciphertext of the AES key and the ciphertext of the EphID.

Thus, the AES-RSA encryption model is a secure way to transmit information, and it would improve the security of DP-3T.

4.6 Improving PRNGs With Machine Learning

One of the downsides of DP-3T mentioned in Section 3.7 is the insufficient randomness of PRNGs. It is near impossible to achieve true randomness in machines, as True Random Number Generators (TRNGs) rely on natural phenomena like thermal noise, cosmic background radiation, or radioactive decay. Such measurements are highly expensive. On the other hand, a PRNG attempts to emulate TRNGs mathematically. A PRNG generates a sequence of bits and manipulates a starting state, or seed, with a certain algorithm. We call the state S_t, as shown in Figure 3, and the starting state will be called S_0. The algorithm outputs following states and continues the process with the new states until a full cycle, or *period*, is achieved when the output value returns to S_0.

A good PRNG's goal is to maximize the periods so that it would be time-costly and near-impossible for hackers to determine the reseeding algorithm which creates the "random" values. One of the greatest challenges is to create long-enough periods. A commonly-used but not cryptographically secure PRNG in computer science is called the Mersenne Twister. This PRNG is named after the length of its period, which equates to a Mersenne prime. A Mersenne prime is a prime number one less than a power of 2. This particular algorithm's advantage is that it has a relatively large period, with a value of $2^{19937} - 1$.

In order to further randomize results with longer periods, researchers have used various Machine Learning (ML) methods. ML itself is often based off of the idea of randomness, as algorithms achieve better mapping when they use randomness to learn from a data sample. Currently, ML technologies use Neural Networks (NNs), in particular Deep Neural Networks (DNNs) where the layers are hidden, to train ML models for image recognition, natural language

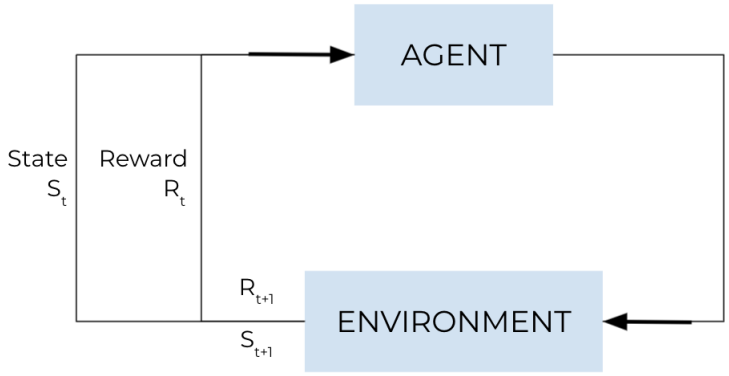

Figure 3: The reinforcement learning mechanism has an autonomous agent take actions to maximize rewards and traverse recurrent neural networks to model sequential data. Figure adapted from [17].

processing, protein modelling, and crafting effective PRNGs. Some research groups used Recurrent Neural Networks (RNNs) to create PRNGs [17]. RNNs are useful for mapping out sequential data and maintaining an internal state, or memory, while training itself. The RNN mends relationships between inputs and thus become advantageous for programs like reseeding algorithms, where one state derives the next. Another group used a Generative Adversarial Network (GAN), an unsupervised-learning framework designed in 2014, which contains a generator that generates additional training data based on the same statistics of the training set [16].

Recently, a group introduced the idea of taking a Reinforcement Learning (RL) approach in order to create a PRNG from scratch. See their published paper [17] for more details. In brief, RL is where a machine optimizes an action by accumulating as much reward from the environment as possible through a *Markov Decision Process* (MDP). An MDP is a "discrete-time stochastic control process," where outcomes are partially random and partially influenced systematically by some decision maker.

Outcome-wise, the RL approach opened up potential in multiple areas:

- No input data is required so it is guaranteed that the generated PRNG is a novel algorithm. Based on the characteristics of RL, the generated PRNGs are likely different from each other after each training process.
- RL policies are stochastic, which means that they are randomly determined. Thus, a single seed can create a non-deterministic PRNG.
- NNs, especially the DNN hidden layers, are black-box operations. Therefore, RL policies are a black-box, and since they give rise to the PRNG

algorithm, the PRNG algorithm itself becomes a black-box. There is no human insight in the PRNGs algorithmic functions which makes it difficult for hackers to determine.

- Experiments such as those from [17] prove the approach is feasible and RL techniques are valid.

Based off of this novel approach using RL, we propose using an RNN-series Machine Learning model which trains through RL and generates bit-by-bit sequences with periods long enough to sufficiently be called "random." The model will be a combination of RL and Long-Short Term Memory (LSTM) architecture. LSTM specializes in observing temporal consecutive data values and masterfully extracts data characteristics [3]. This way, we can create PRNGs with longer periods that rely on less energy to process over Bluetooth LE.

ML is a promising solution to improve not only PRNGs but overall encryption security. It can create non-deterministic algorithms, employ *side-channel analysis* to uncover errors in security proofs, simulate new attack techniques to then create follow-up security proofs and defenses, and do an in-depth cryptoanalysis. By creating better PRNGs, we ensure security in the Bluetooth LE and the encrypted packages and EphIDs being passed along over the Bluetooth connection.

4.7 BB84

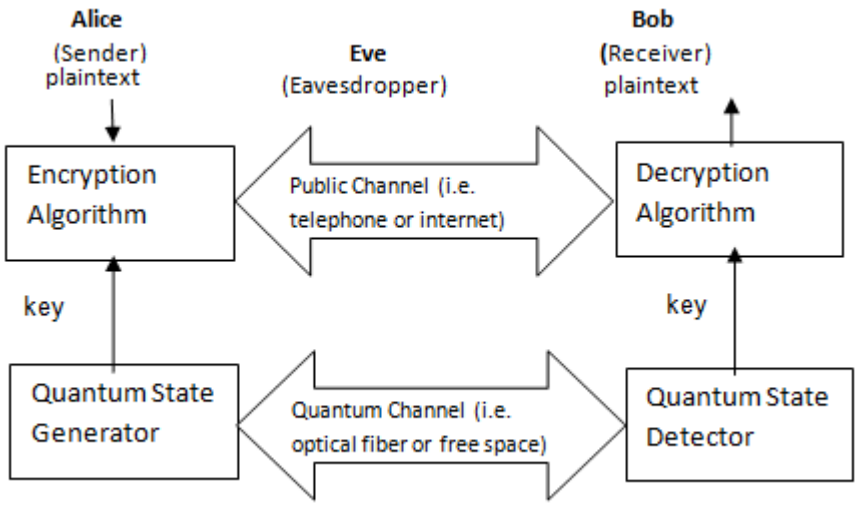

Figure 4: Avia and Bai's twins, Alice and Bob, are also exchanging data; their devices mend both a classic and quantum channel to use BB84. Figure taken from [1].

Quantum computing is a powerful tool that makes a variety of algorithms vulnerable—including the "unhackable" RSA encryption scheme—due to its enhanced computing speed and power [2]. Therefore, in a post-quantum world, or a world in which quantum computers exist, new protocols must be used to retain the security of the DP-3T algorithm. One of these protocols is BB84, the first quantum cryptography protocol developed by Charles Bennett and Gilles Brassard in 1984.

For more details on the mathematics behind the algorithm, check Appendix A. The process described below is from [4].

Returning to example Example 3.13, Avia wishes to send an EphID to Bai's device without Eve listening in on their conversation. Avia will randomly produce a string a and a base b, both of which are n bits long. They use a and b to produce quantum bits or *qubits*.

Qubits mainly differ from a classical computer bit because of the quantum mechanics law of *superposition*. According to this law, a qubit has no definite value, and instead takes on different states simultaneously. Whereas a classical computer bit is either 0 or 1, a qubit only takes on a definite value when it is measured using a base [11].

Avia will transmit these qubits to Bai over a quantum channel as shown in Figure 4. In order for Bai to turn the qubits into definite values, she randomly produces a base b'. Bai will "run" the qubits through this base, and it will produce definite values. If for some index i we have $b_i = b'_i$, Avia and Bai will produce the same definite value. However, if $b_i \neq b'_i$, then Bai has a $1/2$ chance of producing the same definite value as Avia.

In this manner, Bai constructs a', a string of definite values. Avia and Bai then publicly share b and b'. If for some index i, if we have $b_i = b'_i$, that means $a_i = a'_i$. Otherwise, if $b_i \neq b'_i$, there is a chance that $a_i \neq a'_i$, so Bai and Avia reject these values. In the end, Avia and Bai are left with two strings, which we will denote as k_a for Avia and k_b for Bai. Since this string only includes values where their bases matched, we will have $k_a = k_b$.

If a third party, eavesdropper Eve, is listening in, the situation becomes more complex. The act of Eve measuring Avia's qubits will change their values because of superposition. Furthermore, according to the *no-cloning theory* there is no way for Eve to make a copy of Avia's qubits because of their complex and erratic nature [26]. Thus, if Eve is intercepting the messages, Bai does not receive the message Avia sends her, but instead receives a different set of qubits.

As a result, even if for some index i we have $b_i = b'_i$, we will not necessarily have $a_i = a'_i$ because the qubits might be different. Thus, $k_a \neq k_b$. In order for Avia and Bai to determine if a third party is listening in, Avia will publicly share a random selection of bits from k_a. If these bits match with Bai's, they can conclude they are on a secure channel and use the unshared bits of k_a as a secret key to encrypt their EphIDs and exchange data. If the bits do not match, Avia and Bai form a new quantum channel and start the process over again.

Hence, once the BB84 mechanism is placed as an automatic precursor to EphID exchange over Bluetooth, third-party interception can be spotted at an early stage and security measures can be taken without having revealed any of

the close contact data, much less leave the data open for alteration.

5 Conclusion

While DP-3T is a highly-effective solution for carrying out contact tracing while protecting location privacy, we underscore multiple weaknesses in the algorithm. These weaknesses could cause contact-tracing exchanges or data leakage, which then would result in economic, social, and political troubles at both an individual and national scale.

We also explain a diverse combination of both pre-existing and novel solutions. These include an algorithm to verify if a user contracted the virus, technical methods for increasing the security of the back-end server, sending dummy traffic to confuse malicious attackers, deriving secret keys, and using an AES-RSA encryption scheme to exchange information. We also introduce a machine learning model for improving PRNGs inspired by pre-existing solutions, and we introduce a "futuristic" solution: a quantum computing algorithm, BB84, for transmitting data securely.

Continual experimentation with these encryption mechanisms and solutions may greatly improve contact tracing systems for COVID-19, as well as for future pandemics or epidemics, in terms of both privacy and security.

Technology has allowed us to stay connected despite the 6-feet social distancing barriers that have become a norm during the pandemic era. Our investigations into DP-3T serve as testament to the electronic inter-connectivity and digital health systems that exist in the modern world. Furthermore, DP-3T takes such technologies and attempts to find a middle ground in the spectrum between security and privacy.

A Appendix: The Mathematics Behind BB-84

The BB84 protocol logic is based on the quantum property that information gain, or a hacker's observation of secret keys, is only possible if the signal of the two states is disturbed in one way or another. These observational signal disturbances are often the case for symmetric encryption methods. This is when one party attempts to securely communicate a private key to another through a *one-time pad encryption* (OTP).

OTP is an encryption technique which requires a pre-shared key between parties which meet the following fundamental rules of strong encryptions: (1) the key is truly random, (2) the key is at least as long as the plaintext, (3) the key is never reused in whole or in part, and (4) the key is kept completely secret. Condition (4) becomes difficult once hackers are able to intercept and "open" the message sent between the parties. This is where BB84 comes into play. In this appendix section, we will further explain the mathematics behind the BB84 protocol already covered in Section 4.7.

In quantum computing, qubits are characterized with amplitudes, which are

complex numbers in the form $a+bi$. Here, a and b are real numbers and i is the imaginary equal to $\sqrt{-1}$. Each qubit contains two amplitudes A and B, and the "$|\rangle$" notation denotes vectors in a quantum state. It also tells us that the qubit will appear as 0 with probability A^2 and 1 with probability B^2 [2].

As in Example 3.13, Avia wishes to send a private key to Bai. Avia starts with two strings a and b, each n bits long. They encode the two strings as a tensor product of n qubits,

$$|\psi\rangle = \bigotimes_{i=1}^{n} |\psi_{a_i b_i}\rangle \,,$$

where a_i and b_i are the i-th bits of a and b respectively. With two states total, a_i b_i provide four qubit states total after evaluating the equation above:

$$\begin{aligned}
|\psi_{00}\rangle &= |0\rangle \,, \\
|\psi_{10}\rangle &= |1\rangle \,, \\
|\psi_{01}\rangle &= |+\rangle = \frac{1}{\sqrt{2}} |0\rangle + \frac{1}{\sqrt{2}} |1\rangle \,, \\
|\psi_{11}\rangle &= |-\rangle = \frac{1}{\sqrt{2}} |0\rangle - \frac{1}{\sqrt{2}} |1\rangle \,.
\end{aligned}$$

For these states, we see that the probabilities for 0 and 1 are both $1/2$ since $(\frac{1}{\sqrt{2}})^2$ is equal to $1/2$. Additionally, because the qubit states are not mutually orthogonal, or their vector dot products do not equal 0, it becomes impossible to determine all values without knowing the value for b.

As explained in Section 4.7, Avia sends $|\psi\rangle$ over a public and authenticated quantum channel E to Bai, who receives a state $E(p) = E(|\psi\rangle \langle\psi|)$. Here, E represents the numerical interpretation of noise in the channel and effects of eavesdropping by eavesdropper Eve.

Avia, Bai, and Eve each have their own qubit states' values. Currently, only Avia knows the value b, while Bai and Eve find it virtually impossible to distinguish the qubit states. Eve can attempt to measure Avia's qubit states, but in the process risks disturbing a qubit with probability $1/2$ if she guesses the wrong basis.

After, Bai generates a random bits string b' of the same length as b and measures the string she received from Avia, a'. Bai sends a public channel to Avia to determine which b_i an b'_i aren't equal. Both Avia and Bai discard their qubits in a and a' where the b and b' don't match.

For the remaining k bits, Avia randomly chooses $k/2$ bits to disclose over the public channel and both check whether more than half the numbers agree. Once the check is confirmed, Avia and Bai continue to use *information reconciliation and privacy amplification techniques* to create shared secret keys an exchange data. Otherwise, the process is repeated again with new random values until there is a confirmed secure connection.

References

[1] The AES-256 Cryptosystem Resists Quantum Attacks - Scientific Figure on ResearchGate. Available from: https://www.researchgate.net/figure/Distributing-key-over-quantum-channel_fig3_316284124 [accessed 13 Jun, 2021]

[2] Aumasson, Jean-Philippe. *Serious Cryptography: a Practical Introduction to Modern Encryption*. No Starch Press, 2018.

[3] Beaufays, Francoise, et al. *Long Short-Term Memory Recurrent Neural Network Architectures for Large Scale Acoustic Modeling*. https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/43905.pdf

[4] Bennett, Charles H. and Brassard, Gilles, *Quantum cryptography: Public key distribution and coin tossing*, Theoretical Computer Science, Volume 560, Part 1, 2014, Pages 7-11, ISSN 0304-3975, https://doi.org/10.1016/j.tcs.2014.05.025. (https://www.sciencedirect.com/science/article/pii/S0304397514004241)

[5] Choi, Yunseo and Unwin, James. *Racial Impact on Infections and Deaths due to COVID-19 in New York City*. 9 Jul 2020, https://arxiv.org/pdf/2007.04743.pdf.

[6] Chouffani, Reda. "Fortify These 6 Security Layers to Protect Patient Data." *SearchHealthIT*, TechTarget, 28 June 2019, searchhealthit.techtarget.com/tip/Fortify-these-6-security-layers-to-protect-patient-data.

[7] Conikee, Chetan. "Case Study: CVE-2020–15957 Vulnerability Discovery in DP-3T COVID-19 Contact Tracing Backend..." Medium, Medium, 6 Aug. 2020, https://bit.ly/3gkRFFz.

[8] "Contact Tracing – CDC's Role and Approach." *CDC*, CDC, 15 Jan. 2021, www.cdc.gov/coronavirus/2019-ncov/downloads/php/contact-tracing-CDC-role-and-approach.pdf.

[9] "COVID-19 Coronavirus Pandemic." *Worldometer*, Worldometer, www.worldometers.info/coronavirus/.

[10] Craven, Connor. "What is the Advanced Encryption Standard (AES)?" *SDxCentral*, SDxCentral, 13 May. 2020, https://www.sdxcentral.com/security/definitions/what-is-advanced-encryption-standard-aes-definition/.

[11] Davidson, John. "Quantum Computing 101: What's Superposition, Entanglement and a Qubit?" Australian Financial Review, 26 Dec. 2019, https://bit.ly/3iE5q3C.

[12] "Download Free Phone Icons Transparent PNGs." Stick PNG, Stick PNG, www.stickpng.com/cat/electronics/phone-icons?page=1.

[13] *Exposure Notification — Cryptography Specification*. Apple and Google. Apr. 2020, covid19-static.cdn-apple.com/applications/covid19/current/static/contact-tracing/pdf/ExposureNotification-CryptographySpecificationv1.2.pdf?1.

[14] "FAQ." Coronalert, 2 June 2021, coronalert.be/en/faq/#faq-17.

[15] Frankenfield, Jake. "Cryptographic Hash Functions Definition." Investopedia, Investopedia, 19 May 2021, www.investopedia.com/news/cryptographic-hash-functions/.

[16] Goodfellow, Ian J., et al. *Generative Adversarial Networks*. 10 Jun 2014, https://arxiv.org/abs/1406.2661

[17] Pasqualini, Luca and Parton, Maurizio. *Pseudo Random Number Generation through Reinforcement Learning and Recurrent Neural Networks*. 19 Nov 2020, https://arxiv.org/abs/2011.02909

[18] Rege, Komal, et al. Bluetooth Communication Using Hybrid Encryption Algorithm Based on AES and RSA. International Journal of Computer Applications, citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.402.8867&rep=rep1&type=pdf.

[19] Ronen, Eyal et al. *IoT Goes Nuclear: Creating a ZigBee Chain Reaction*. 2016, *Cryptology ePrint Archive, Report 2016/1047*. https://eprint.iacr.org/2016/1047

[20] Sang-hun, Choe. "In South Korea, Covid-19 Comes With Another Risk: Online Bullies." *The New York Times*, The New York Times, 19 Sept. 2020, www.nytimes.com/2020/09/19/world/asia/south-korea-covid-19-online-bullying.html.

[21] Swinhoe, Dan. "What Is a Man-in-the-Middle Attack? How MitM Attacks Work and How to Prevent Them." CSO Online, CSO, 13 Feb. 2019, https://bit.ly/3zm6VJL.

[22] Troncoso, Carmela, et al. *Decentralized Privacy-Preserving Proximity Tracing*. 25 May 2020, github.com/DP-3T/documents/blob/master/DP3T%20White%20Paper.pdf.

[23] Troncoso, Carmela, et al. *Secure Upload Authorisation for Digital Proximity Tracing*. 30 April 2020, https://github.com/DP-3T/documents/blob/master/DP3T%20-%20Upload%20Authorisation%20Analysis%20and%20Guidelines.pdf.

[24] Watson, Ivan, and Jeong, Sophie. "Coronavirus Mobile Apps Are Surging in Popularity in South Korea." *CNN*, Cable News Network, 28 Feb. 2020, edition.cnn.com/2020/02/28/tech/korea-coronavirus-tracking-apps/index.html.

[25] "What Is A Blue Tooth?" *Cromer Pier & Pavilion Theatre*, 9 June 2020, www.cromer-pier.com/what-is-a-blue-tooth/.

[26] Wootters, W.K. and Zurek, W.H. *A single quantum cannot be cloned*. 28 October 1982, *Nature*. https://www.nature.com/articles/299802a0.