SmartLock Picking

Valerie Lemuth, Johanna Sacher, Benjamin Burse and Pia Fichtl
firstname.lastname@uni-weimar.de

***Abstract*— Smart technologies have grown in popularity over the last years and are used in multiple aspects of everyday life. Since the smartness and convenience of these new technologies is in the foreground, the question of security is often overlooked by the user.**
In the project *SmartLock Picking*, we examined different types of smart locks in order to find security vulnerabilities. We mainly concentrated on smart padlocks, which are padlocks that do not (or not only) require a traditional key to be unlocked, but instead (or in addition) use some type of technology like Bluetooth or NFC to identify a key. After defining our security goals, we separated the locks by technology and ended up with four types: Bluetooth, Client-Server-Architecture, NFC and Fingerprint Readers. Depending on the technology of the lock, we tried specific attacks, ranging from soft- to hardware attacks. Since some locks use multiple technologies and therefore fall into multiple categories, they reappear throughout the paper.
We also propose ways for continuing our work and building on our findings. (Valerie)

I. INTRODUCTION

Smart technology is a trend that has reached the masses. It is present in all areas of life today: phones, heating, cars - and even locks. Smart locks are getting increasingly common in different use contexts and are often used in conjunction with an app.
These new technologies are tempting because they promise to be handy and comfortable, they make everyday life easier and are time-saving. Instead of finding one's keys deep down in a bag, a user only has to reach for their phone or a small token.
However, few users think about potential security issues when buying and using the locks, trusting in the developers. For locks in particular, we have to ask questions such as the following: Who can gain access? What happens if the lock's battery runs low? Who is the person or company server-side and thus, the one granting us access to our own home? What happens if the company, for some reason, stops supporting the door lock - can the owners still access their possessions or home?
Over the course of the project *SmartLock Picking*, we examined different types of smart locks in order to find potential security vulnerabilities. We mainly looked at smart padlocks: In addition to or instead of a traditional key, those locks can be opened with some digital technology that users mainly already know (and trust) from their everyday lives. Bluetooth, for example, is also used to connect smartphones to a speaker, NFC can make connecting to the WiFi much easier and nearly every smartphone today can be unlocked using a fingerprint.
In this paper, we first determine important security goals a lock has to fulfill to deserve to be called a lock. It is important to do this before examining the locks, as security vulnerabilities can best be found when searching for a specific one, rather than just trying to break something.
After determining desirable security goals, we examine each of the locks separated by their respective technology. Four types of technologies are presented and assessed regarding security vulnerabilities: Bluetooth, Client-Server-Architecture, Near Field Communication (NFC) and Fingerprint Readers.
In order to determine security problems with the locks, different attacks, some of them specific to each technology, are presented. These range from software to hardware attacks and we present everything an adversary could try to pick a lock without just cutting it open. Breaking a lock is of course effective, but with the right saw or bolt cutter every lock can be opened. This is not a weakness of the lock's technology and therefore we will not pay attention to it and rather focus on vulnerabilities the different technologies entail.
In each technology chapter we present the specific locks, how they work, which attacks we conducted and whether any weaknesses to the attacks were found. Some locks appear at multiple locations in this paper, since they can be accessed using multiple technologies.
We conclude that many of the locks are vulnerable to common attacks that can easily be conducted by any adversary. In each case, users need to consider the security risks associated with smart locks very carefully and could mostly be better off using a traditional padlock.
Lastly, we propose ways of further continuing the work done in this project. We only had a limited time and budget range for a research area that promises a lot more interesting findings.
(Pia and Hanna)

II. RELATED WORK

We analysed related works according to the different technologies.

A. On Near Field Communication (NFC) (Hanna)

In his Master-thesis *"Access Granted: On the Security of Near-Field Enabled Keycards"* Stephen Jones examines the Security of NFC cards that are used as a key for NFC enabled locks, especially the UEA campus card. He concludes the card has vulnerabilities and recommends ways of improving

the security issues. Jones takes a closer look at phishing and skimming attacks on actual users [1] while our research is focusing more on the locks and tags themselves.
The TU Darmstadt developed NFCGate, which "is an Android application meant to capture, analyse, or modify NFC traffic. It can be used as a researching tool to reverse engineer protocols or assess the security of protocols against traffic modifications" [2]. A note on their gitHub page says "This application was developed for security research purposes by students of the Secure Mobile Networking Lab at TU Darmstadt. Please do not use this application for malicious purposes." [2], which is due to the fact that this app can create copies of NFC-tags and use the copy as the original tag [3].

B. Fingerprint Reader (Valerie)

In the Paper *"Impact of Artificial "Gummy" Fingers on Fingerprint Systems"* [4], Tsutomu Matsumoto explains how to create different types of artificial fingerprints. He shows how the so-called gummy fingers can fool specific fingerprint devices with optical and capacitive sensors.
In this project, we specifically concentrated on fingerprint systems within smart padlocks, whereas he examined a wider range of devices.

C. Client Server (Ben)

The field of client server architectures offers the attacker to choose from a wide variety of possible attacks. The most simple attack is sniffing, where the content of unencrypted protocols, like HTTP, FTP or telnet, could directly be analysed [5]. A very popular - but depending on whether encryption is used, quite complex - attack is the Man-In-The-Middle (MITM) attack. Richard Ford and Michael Howard propose a MITM attack on the HTTPS protocol by faking server certificates [6]. With their attack they show that by exploiting bad user habits, it is possible to decrypt the - commonly assumed to be safe - HTTPS protocol.

D. Bluetooth Low Energy (Pia)

In the paper *"Gattacking Bluetooth Smart Devices"* [7], Slawomir Jasek first gives an overview of Bluetooth Low Energy technology, communication and possible security. He then analyses common security issues, attacks on Bluetooth Low Energy devices and countermeasures, based on real-life experiments to find vulnerabilities. He finds that often, security measures are not properly implemented by the manufacturers [7]. With *GATTack*, he introduces a new open-source tool that exploits common security issues to run a man-in-the-middle attack [7].
Tal Melamed examines the main security issues concerning Bluetooth Low Energy devices in his paper *"An active man-in-the-middle attack on bluetooth smart devices"* [8]. He names several tools available for attacking Bluetooth Low Energy devices and then discusses how to use them together in a possible attack scenario. He successfully uses the tools to eavesdrop on the communication between the central and peripheral device and show how the tools can even enable the attacker to even gain control of functions on a mobile device [8].
The paper *"Smart Locks: Lessons for Securing Commodity Internet of Things Devices"* [9], the authors analyze the security of several smart locks for homes. First, they examine the mechanics of smart lock systems, then they examine threats, security goals and different attack models on smart devices. They survey different locks according to the attack models in order to show that they are vulnerable to the attack models. They propose ways of defending smart locks against the attacks, concluding that the attack models and defenses against them should also hold true for other IoT devices [9].

III. Security Goals

A lock is always supposed to be secure, otherwise the usage of a lock would be pointless. In order to find the weaknesses of a lock, it is important to define what security means, so each security goal can be tested for each lock.
We formulated the following security-rule that applies to locks: *Exclusively the person(s) in possession of an authorized key should be able to open the lock.*
This, however, contains several sub-challenges.

1) A second key or a legal copy of the key has to be authorized by the owner or master-user, otherwise the lock can not be unlocked with it.
2) Copying the key illegally (i.e. without the owner's permission) is impossible.
3) Faking a key is impossible.
4) Opening the lock without a key (i.e. with a switch on the lock or with typical lock-picking tools) is impossible.
5) The owner can revoke access to the lock/unauthorize a second key.
6) Access log integrity needs to be maintained - an attacker should not be able to interfere with the access log or prevent their interaction with the lock from being recorded.

(Hanna and Pia)

IV. Technologies

A. Near Field Communication

Near Field Communication (NFC) is an international communication standard which enables the exchange of data between two devices in close proximity to each other [3, 10]. It is based on *Radio-Frequency Identification-Technology* (RFID) [10]. The idea here is to read Data via an electromagnetic field. RFID consists of two components: a transponder, also called RFID-tag, and a reading device [11]. The reading device generates an electromagnetic field and sends a signal to the coiled antenna of the tag [12]. The antenna forwards the signal to the tag which then sends the data back via the same field. This works over great distances from up to 100 metres and through different materials, which makes RFID a useful technology [12].

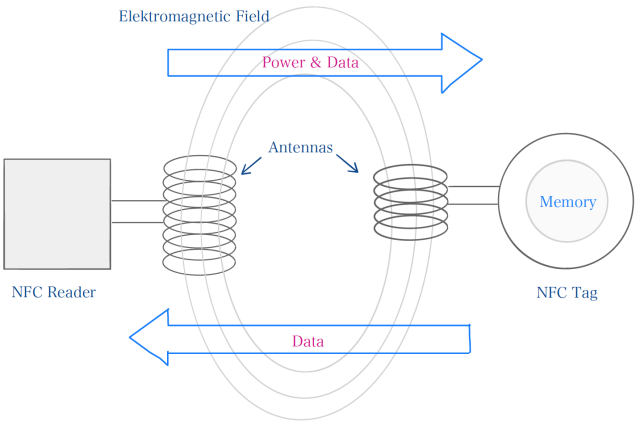


Fig. 1: Data transmission via electromagnetic field

For example, it is frequently used in electronic article surveillance, chips for pets, livestock and cars, in cashless payment or in lock technologies [12].

NFC basically works the same as RFID but has some more constraints specified in international standards (ISO 14443-2, 14443-3, 18092, 21481) [13]. In short, the NFC Tag is a memory chip with a coiled antenna around it (see figure 2), the data transfer works via electromagnetic induction (see figure 1), it has a special way of connecting and the connection only works over a very short range [14].

Fig. 2: NFC Tag with coiled antenna around memory chip [15]

For that reason, NFC typically has different fields of application than Bluetooth or WiFi. It is a good alternative to bar or QR codes, as its memory is larger [10].

There are two transmission modes: passive (active reading device and passive NFC-tag which gets powered by the electromagnetic field) and active (connection between two active devices) [10]. To connect, the components only have to be close enough, there is no other authentication necessary. A consequence of this fairly easy connection process is that security is relatively hard to realise with NFC. It mostly depends on how the data is stored on the tag. The tag itself is always accessible by any reading device close enough and while some tags can be encrypted, a lot of them can not and are therefore cheaper and more frequently used. Additionally it is important to understand that each NFC-tag has an unique ID (UID) [3] that can typically not be changed. A smart lock that uses NFC is a lock with a NFC reading device inside. The key is a NFC-tag and when held in front of the lock, the reading device checks whether the tag has the correct ID and/or the correct data in it's memory. If it has, the lock will open, if not, it will stay closed.

1) Attack Model: There are several different attacks that can be tried to pick NFC locks. The goal is to emulate the key to get the lock to accept the adversary as valid user rather than stealing the key, which would not be an attack on the lock but on the user. These attacks will be explained briefly in the following:

a) PHISHING: an attacker tries to manipulate the user by false promises or wrongful information in order to gain sensitive information [16]. In this case, that could mean tricking the user into giving or lending the key to the adversary, so they can scan and save the tag data to replay it later without the user knowing.

b) SKIMMING: the unauthorised reading of data [17]. This could, for example, happen when a tag is exposed even for a short period of time. An attacker just has to pass by that tag and scan it with a smartphone. Thinking of all the key-cards being worn clipped to the outside of working clothes, this is a very likely scenario. (The scanning could of course also work through a thin material like cloth). In a split second, the attacker can create an illegal key-copy without the owner even noticing it. The reading and storing of NFC data for the purpose of replaying it later to use it as a key is possible with several apps that are available for free, as we will see in subsection "Phishing/Skimming" of subsection 2, "Smart Travel Padlock".

c) RELAY ATTACK: for a relay attack, there are two additional reading devices necessary, for example two smart phones. These devices have to be connected via a server. Then one of them is used to read the NFC-tag. It sends the data via the server to the other device, which emulates the tag by replaying the received data to the reader of the lock. Figure 3 shows the connections and the flow of the data.

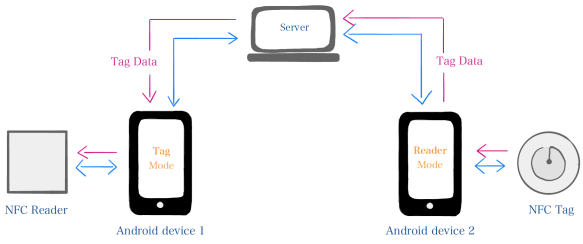


Fig. 3: Relay Attack

The lock then will accept the smartphone as the key and open up. This attack reveals an important vulnerability of NFC: the distance of maximal 10 centimetres for data transmission can easily be widened up to hundreds of kilometres - the only thing necessary is a working server and a lock in Hamburg can be opened even if it's key is currently located in Munich [2, 3].

d) COPING THE KEY TO AN EMPTY TAG: With the right tools, an unencrypted NFC-tag can be read and copied to an empty NFC-tag. This is comparable to the reproducing of a traditional key.

e) SHIMMING: Inserting a thin but stable piece of sheet to press down the pin keeping the shackle closed. For a more thorough explanation see Attack Model "Shimming" in Section B "Fingerprint Reader".

As for security issues, it is important to note that any picked up data, even encrypted data, can be useful because it can potentially be decrypted offline [1]. However, there are some type of tags that can be encrypted and the standard encryption mostly used today is not yet broken [18]. A patient attacker could save the data and wait or try to break it anyway, but this is more of a lurking threat that also applies to most of the used passwords today. In all of the above cases, the user does not know they were attacked or stolen from until they see the actual damage [1]. In comparison, if a physical key is lost, the owner switches all locks and is safe again, but when someone just briefly holds a smartphone over a key card to scan it and safe the data, no one has to even notice.

2) Smart Travel Padlock: The eGeeTouch Smart Travel Padlock GT1000-96 [19] is a padlock (see figure 4) that can be unlocked via Bluetooth using an app and via NFC using the NFC-tag that comes with it. It has no additional physical key but can be opened with a TSA key for "by-pass access by TSA authorised personnel in US airports" [19]. On their website the manufacturer promises a "Superior access security authentication with hundreds of millions of user-defined IDs" [19] and claims the lock were "Fast, easy and secure for safeguarding personal belongings in the luggage or travel bag" [19].

Fig. 4: eGeeTouch Smart Travel Padlock GT1000-96 [19]

Regarding the NFC function, the lock works as intended. When pressing the red power button on the left, it beeps and a green light flashes from the "TAP HERE"-Area which indicates that the reader is searching for a NFC-tag. Tapping the provided NFC-tag on the lock, it beeps again and now the power button can be dragged down to open up the lock. Pressing the red button again results in two beeps and the mechanism locking. When trying to open the lock with any other NFC-tag like the Thoska or an empty tag the tag gets rejected with three beeps and a red light and the lock stays locked. So far, this is all the NFC feature promised to be able to do: unlocking on identifying the correct tag and staying locked otherwise. When attacked with any of the above mentioned attacks though, it fails completely due to the ill-conceived implementation of the NFC-tag that functions as the key. As we will show, this key can easily be copied and the data then be replayed, fooling the reading device into accepting the copy as the original key.

We tried several different attacks on this lock and found weaknesses in nearly all possible areas of security. The NFC-tag which serves as a key can easily be read and the data can be stored and replayed in front of a reading device. The data is neither encrypted nor protected in any other way. The reader of the lock can as easily be tricked with the replayed data, it accepts any tried smartphone replaying the data as valid key and opens up.

PHISHING/SKIMMING: As it is the easiest attack, we first tried Phishing/Skimming (we don't need to distinguish between those two models here as we own the key and we always know when we are scanning it). In order to perform this attack, we needed additional tools in form of software that was able to read, store and replay the tag data. There are several apps that promise to be able to do that, two of them are NFCProxy [20] and NFCGate [2].

NFCProxy "is an Android app that lets you proxy transactions between an RFID credit card and a reader. The saved transactions can be replayed to skim credit cards or the RFID credit card can be replayed at a POS terminal." [20] For the app to be able to replay data it requires a special version of CyanogenMod installed on the Android device [20]. At first, a lot of research about making a backup of an Android phone and how to return to Android when a custom ROM has been installed was done to be sure the phone could be used again, even after a possibly buggy installation of a custom ROM. CyanogenMod is no longer supported and hard to find on the internet, so after enabling the developer options on the phone, installing twrp [21] and downloading the Nexus 5 hammerhead.img, LinageOS 16.0 was installed on the Nexus 5 as it is the successor of CyanogenMod and therefore the alternative that was most likely to work. However, after installing the NFCProxy App and searching for an option to enable NFC, we found it not working. Rumours on the internet had it that the developers switched off NFC due to it not working and there was nothing to be done about not being able to switch it on, so different older versions of LineageOS were installed and tested. LineageOS 14 did indeed support NFC, but on scanning any NFC-tag the NFCProxy App always sent the error: "pcd support not available unpredicted behaviour ahead." Even after researching for quite some time there

was nothing to be found on ”pcd support” or how to fix it. As a last resort we found an old CyanogenMod build which was then successfully installed and which supported NFC. However, the app kept sending the same error message without any indication as to why and was therefore no longer used.

NFCGate was developed by the TU Darmstadt for researching purposes [2, 3]. This app can be cloned from gitHub [2] or downloaded as a compiled apk [22]. It has several different modes of operation: one for cloning, one for relay and one for replay attacks. In order to work, the NFCGate app needs the phone to be rooted and have the xposed framework installed [2], so after learning how to flash a custom ROM on an Android phone, we learned how to root a phone. At first, the backup on the Nexus 5 was restored to get back the original Android Version. Then the phone was rooted and flashed with the xposed framework, the NFCGate app was installed and the ”NFCGate” box under ”Modules” in the xposed app was checked. After rebooting the phone once again, NFCGate scanned all NFC-tags perfectly fine. To clone the tag, NFC has to be enabled and the app opened in clone mode. Then the phone is held over the tag and thus the tag is scanned. The scanned data can then be saved or just left open on the screen. The phone will constantly send out the currently open data so it now just needs to be held in front of the scanning reader. The travel padlock will accept the data from the phone as the valid key and open up. The saved data can hereafter be replayed whenever needed without having to scan the tag ever again; the smartphone now contains a working copy of the key.

For future work it is really important to note that a second NFC App, like *NFC TagInfo* for example, can be the reason why an attack like described is not working properly. We installed such an app after conducting the experiment and it did not work anymore after that. On uninstalling any other NFC apps, it worked again perfectly fine.

RELAY ATTACK: After the successful conduct of the skimming/phishing attack we wanted to go one step further and perform a relay attack on the lock and it’s tag (the relayed devices). Necessary to that end are two Android devices (the relay devices), both rooted and with the NFCGate app installed, as well as a working server which both devices can connect to. Such a server written in Python can conveniently be cloned from the developers of NFCGate [23] and simply be started via the terminal. When in the same WiFi, on both devices the Hostname, the Port (as indicated by the running server) and the Session need to be specified in the settings [24]. Then in relay mode one of the devices gets assigned to be the ”Reader”, the other one to be the ”Tag”. Now the Phone in ”Reader” mode is held over the tag while the phone in ”Tag” mode is placed in front of the scanning reader. The padlock will again accept the transmitted data as the original key and open up. The transmission is saved as ”Session” by the app and when switching to ”Replay” mode, each session can be replayed whenever needed. So again, a working copy of the key was created and stored on the Android device.

COPY KEY TO EMPTY TAG: This attack was conducted as part of trying to pick the second NFC lock and we tried to copy the data from the tag to an empty NFC-tag using nfcpy [25] and the SCL3711 Contactless USB Smart Card Reader [26]. The resulting tag could not open the lock but this is probably due to the limited possibilities of formats for writing on the tag provided by nfcpy. Using a different method or library copying the tag data to an empty tag would probably work just as well as the previous attacks and we would propose this for future work.

SHIMMING: Shimming did not work on this lock as the gap between the sheathing and the shackle is too tight to insert a shim.

The *Smart Travel Padlock* is therefore vulnerable to nearly all of the above mentioned attacks as it’s tag is an NTAG216; a type 2 tag which cannot be encrypted [27] and is an easy target to any scanning attempt because that is what it was designed for. This type 2 tag is not designed for security use cases [27] and is therefore a completely inappropriate choice for a key.

3) Rothult NFC-lock: The ROTHULT NFC-lock from IKEA [28] is a drawer lock (see figure 5) that can only be unlocked using an NFC key-card. Enclosed are two master key-cards that always open or close the lock (depending on it’s current state) when being tapped on the front of it.

Fig. 5: ROTHULT NFC lock [28]

The lock works with three AAA Batteries and as soon as these are placed inside and the lid is screwed down, the lock beeps and closes (i.e., the metal stud comes out). Unlocking it works just as well by placing the tag on the lock yet again. When trying to open the lock with an unknown key such as a Thoska it beeps three times and stays locked. There is no other way to open it (except for unscrewing all screws, taking everything out and spin the gears with one’s finger, which can of course not be performed when the lock is installed correctly on the inside of the drawer and closed). The user manual describes one useful feature: when the lock

is unlocked, any NFC card or NFC enabled smartphone can be placed on it and the lock will close and henceforth accept this card as an additional key until it gets reset by taking the batteries out. This works with most of the NFC cards and tags we tried but with only one smartphone, the Sony xperia xCompact, which proved as a difficulty when trying to attack the lock with the above mentioned attack models. The downside is, that the manual promises this extra key function to only work with one extra key at a time while we were able to add as many keys as we wanted. So in case anyone ever came across this lock unlocked, they could very easily make an illegal key-copy with any NFC card and gain access until the lock were reset. This is a weakness that can easily be abused and it is described incorrectly in the user manual, so this lock, while doing some other things right, is still not as secure at it claims to be.

PHISHING/SKIMMING: The master key-card was scanned using the NFCGate app, as done before. However, the lock rejects the Nexus 5 anytime the phone gets near the lock and we cannot decide whether this is due to the fact that the attack simply does not work on it or because the phone is not compatible to the lock. We were also not able to add the Nexus 5 as an additional key, which should be possible according to the user manual. As we did not have any other smartphone available that could be rooted, we had to find out what exactly was the problem. To figure out whether the problem lay with the key-card or with the phone, the key from the *Smart Travel Padlock* was added as an extra key to the lock. Then we tried to open up the *ROTHULT* with the scanned data from the *Travel Padlock* key, which could trick the *Smart Travel Padlock* into unlocking, but the *ROTHULT* still rejected all attempts. So it is standing to reason that the Nexus 5 definitely is one problem, but we still could not be sure whether another phone would work or if the lock could just not be tricked. So we followed two paths: First, we had a closer look at the code of the NFCGate app [2] but while this was very interesting, it did not lead us to the intended results, as nothing in the code gave us any clue as to how the rejection of the phone could be a problem with the app. Then, using the *NFC TagInfo* app, scan reports of the *ROTHULT* key-card as well as of the *Smart Travel Padlock* key-chip were generated and compared in order to find out whether the problem lay with the used NFC-tag. It was found that the tag used for the *Smart Travel Padlock* is a NTAG216; a type 2 tag using Mifare Ultralight [29] which cannot be encrypted, whereas the key-card for the *ROTHULT* lock is a type 4 tag. Type 4 tags can be encrypted and while the encryption of the Mifare DESfire was broken in 2011 [18], the Mifare DESfire EV1 is unbroken yet and the most used encryption for access control as of today [18]. This means it is very likely the *ROTHULT* key-cards are encrypted with Mifare DESfire EV1 and thus the data cannot be cloned using an app. However, one could try to copy the key from the *Smart Travel Padlock* to an empty tag, thus copying a key that can be copied and using the feature of adding an additional key against the lock. If this attack would work, the effort of providing a safe master key would have been useless because an unsafe key can be added and multiplied several times.

SHIMMING: This lock does not provide any target for a shimming attack as it is installed inside a drawer and cannot be reached when locked.

COPYING THE UNSAFE KEY: In order to copy the data from the unsafe type 2 tag to an empty tag, we first tried several apps available in the Google PlayStore and claiming to be able to copy data from one NFC-tag to an other. All of these apps failed; they could either not read the original tag or an error occured while writing the data to the empty tag. After those attempts we used nfcpy [25], the SCL3711 Contact-less USB Smart Card Reader [26] and empty NTAG216 Chip Tags Sticker [30] which are the same type as the key to be copied. The SCL3711 was installed under Debian following the instruction of its drivers. In addition the pcscdlite [31] was needed and installed. Now a NFC-tag can be placed on the reader to read it or write on it. We used the Python library nfcpy to do so. This library works with the NFC Data Exchange Format (NDEF) which is a special data format for nfc data and ”consists of NDEF Messages and NDEF Records” [32]. It is ”maintained by the NFC Forum” [32] and can be used to store links, pictures, text and other data. As we found out, the master key-card does not contain any data in the NDEF format so nfcpy cannot work with it and we propose it for future work to try other tools and libraries to read this key-card. The tag from the *Smart Travel Padlock* could however be read and copied to an empty tag, but as the content of the tag consists exclusively of the tag ID followed by zeros in the memory block, this did not have the wished upon effect. The data was successfully copied trying out various of the offered formats, but the ID of the empty tag could of course not be changed. So neither the *Smart Travel Padlock* nor the *ROTHULT* accepted the copy as valid key. Again, we propose it for future work to try other tools and libraries to copy the key to an empty tag. As the NFCGate app succeeded in cloning the data and using it as a valid key, it has to be possible.

B. Fingerprint Reader

Fingerprint recognition systems are built up of three basics parts, the scanner that records the fingerprint, the algorithm that then identifies specific features and the database that stores the found information [33]. As can be seen in figure 6, these systems can be in two different states, the registration or the verification state.

During the registration, the system is accepting new fingerprints and saving them as keys [33]. To do this, it uses the uniqueness of the patterns that the ridges and valleys form on the tip of the finger. They create discernible patterns like loops and arches but for fingerprint identification especially

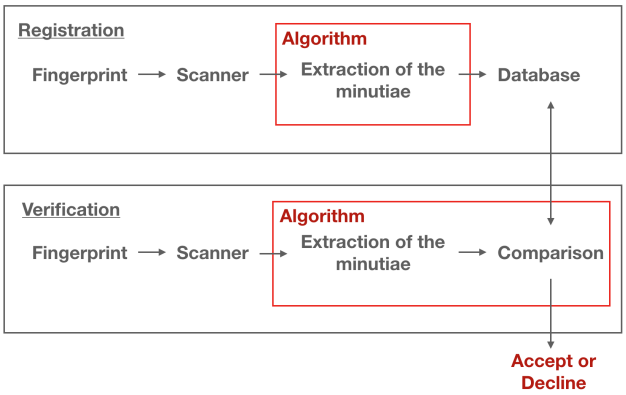


Fig. 6: Fingerprint recognition system

the points where the ridges suddenly end, or fork are important [34, 35]. These points are called minutiae [35]. So, when a new finger is scanned, the relative position of the minutiae is extracted and then stored in a database [36, 37].

During the verification, the system is deciding whether to accept or decline the presented fingerprint [33]. It does this based on the keys that were stored in the database during the registration. So, after the presented finger is scanned, the system compares the extracted data with all of the keys in the database and depending on whether multiple minutiae match, the finger is either accepted or declined [36]. For the comparison the minutiae dont need to be in the same position but rather their relative position is important [37]. By only comparing the minutiae and not the whole fingerprint, the number of errors is reduced, and a fingerprint can also be identified when the finger is only partially on the scanner [36].

Different fingerprint recognition systems use different types of scanners, the most common ones are optical, ultrasonic and capacitive scanners.

Optical Scanners are an older method and relatively bulky, the basic idea is to take a high contrast picture of the fingerprint, therefore, the scanner consists of LED's that light up during the use [36]. Ultrasonic scanners are a newer technology and not as widely used, the idea is to create a 3D model of the fingerprint using ultrasound, most of the time they are placed underneath glass [36]. The smart locks with fingerprint readers that we examined during the project all contained capacitive scanners, that is why we concentrated on this technology. Capacitive fingerprint scanners use capacitor circuits which are connected to conductive plates, this way the electrical charge within the capacitor is changed when something conductive touches the plates [36, 38]. Depending on the conductivity of the material the charge is changed differently, this gives the scanner a possible way for differentiating between real and fake fingers [36]. The scanner then tracks the differences of where the ridges of the finger touch the scanner and where the valleys leave the charge unchanged [36]. This information can be used to reconstruct a picture of the fingerprint. From there the algorithm extracts the minutiae and depending on the state the system is currently in it continues with its next steps.

The smart locks examined by us that use a fingerprint reader as a key, start with the system in the registration state so that the owner can save their fingerprints. From then on, the system will stay in the verification state most of the time, the lock will only open when a fingerprint is accepted, if its declined it will stay closed. To save more fingerprints, the owner can decide to manually go back into the registration state.

1) Attack Model: With fingerprint locks there are two basic types of attacks, software and hardware attacks. An adversary could try to trick the fingerprint scanner and the algorithm behind it or the locking system and therefore the hardware. In the following possible attacks will be explained:

a) ARTIFICIAL FINGERPRINT: Other than just using the finger of the sleeping/drugged owner, an adversary could try to create an artificial fingerprint. There are several different ways for doing this, depending on the amount of time and resources the attacker has.
An artificial finger can be created by using the real finger of the owner and pressing it into a type of modeling clay [39]. The created mould then gets filled with a gelatin liquid or liquid silicon. After hardening, the so called gummy finger can be used as an artificial fingerprint [39].
Another possible way would be to use the fingerprint of the owner and a photosensitive Printed Circuit Board, a PCB. Here, a lot more steps are involved, the attacker would need to dust the fingerprint and take a picture [40]. They would then need to mirror the image and invert the colors so that it could be printed onto a sheet [39, 40]. This sheet could be used as a mask to transfer the fingerprint onto a PCB using UV light [40]. After developing and etching the mould would be ready and could be used with white glue or a gelatin liquid [39, 40].
This method requires a lot of special equipment and therefore we decided to use the method with the real finger. Since that one can be done quickly and also seemed to have the biggest success rate in comparison to the needed time and materials.

b) MANIPULATING THE FIRMWARE: If the lock has a USB port and can, therefore, be connected to a computer, an adversary would have the possibility to manipulate the firmware. This could also be possible with a lock that does not have a USB port but only if the adversary had access to the circuit board. Depending on the microchip inside the lock, a hex file could be written to override the firmware or the firmware itself could be attained and manipulated. This could be done in a way so that the lock always opens

when a finger is rejected by the scanner. For this, the adversary would need to know the specific pins of the microchip that connect to the fingerprint scanner and target them individually. With our locks, there was no easy way of seeing which pin was connected to the scanner and researching this further would have gone beyond the scope of this project.

c) SHIMMING: Depending on the design of the lock, a process called shimming can be used to manually move the catch, thats holding the lock closed. For this a shim is used, a thin piece of metal in a kind of T-shape. The shim is inserted between the shackle and the body of the lock and pressed downwards [41]. While holding the shim down and therefore pressing the catch to the side, the shackle of the lock can be pulled up [41]. This method only works for padlocks with spring-loaded catches [42].

d) LOCK SPECIFIC ATTACKS: If an adversary knows the specific lock well enough, they could also try using weaknesses of that specific design. For example, if they know that a lock has an emergency unlock that is triggered by a low battery, they could try manually discharging it. Some locks have an actual emergency switch that is hidden but still easily accessible in case the owner needs it. There could also be some software bugs that can easily be triggered and exploited.
These types of attacks really depend on the lock design and the knowledge of the adversary and therefore are not mentioned as individual models.

2) Fingerprint padlock number 1 [43]: This padlock consists of a metallic shackle, a silver body with a black front panel and a fingerprint scanner, which is embedded into the panel. Next to the scanner, there is a LED light that gives feedback to the user and indicates the status of the system. (see figure 7)

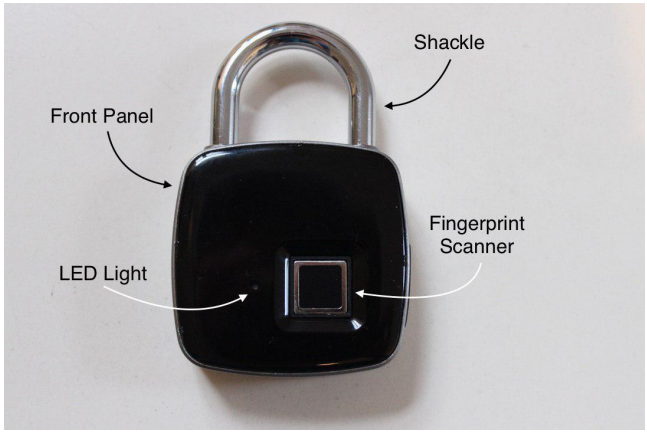


Fig. 7: Fingerprint padlock number 1 with annotations

The darkly coloured plate of the scanner suggests a conductive plate and is therefore neither an optical nor an ultrasonic scanner but rather a capacitive scanner. At the side, the lock has a USB port where a charging cable can be plugged into. According to the manual, the manufacturer promises a battery life of 2 years.

The lock cant be controlled through an external app and there are no extra functionalities like Bluetooth or NFC. The user can control every action by interacting with the fingerprint scanner. Overall 10 fingerprints can be registered and the first two will automatically be the administrators. The first finger can be registered instantly and without any authentication, every fingerprint afterwards needs to be authorized by an administrator's finger. Only the administrators can delete fingerprints, although the instructions dont specify how to select which fingerprint to delete or which fingerprint will be deleted. The instructions are relatively minimal: they only show the process of saving and deleting fingerprints. There is also no official company named anywhere and therefore there is no possibility of finding out any further information. When trying to delete a fingerprint, we found that there is no way to enter which fingerprint should be deleted. The lock simply deletes all fingerprints and returns to its original state. This means that from then on, every finger can open the lock, and everyone can register their finger as the administrator. If the owner does not realize this, it could lead to a security gap. An adversary could trick the owner into deleting a fingerprint, for example through showing interest in the registration process. They could ask to register their finger since the owner can delete it afterwards. If the owner then simply trusts that the deletion worked, without making sure, the attacker and anybody else could get access to the lock.

Other than this easily exploitable inaccuracy in the instruction manual, the lock also has a few design flaws. The shimming attack can be done exactly as described in the attack model. There is the possibility of buying professional shims, but in our case, a thin piece of metal, cut out of an empty soda can, worked very well. (see figure 8)

Fig. 8: Shim made out of a soda can

There is an easy way to avoid this security gap, most modern padlocks use ball bearings, often on both sides,

instead of a spring-loaded catch but for some reason, this step was not taken here [42].

While experimenting we found out that the front panel of the lock can be lifted with the use of a pocketknife. The panel is simply glued on and can easily be removed after applying a bit of force. Underneath the front panel, there is a switch which instantly opens the lock. (see figure 9)

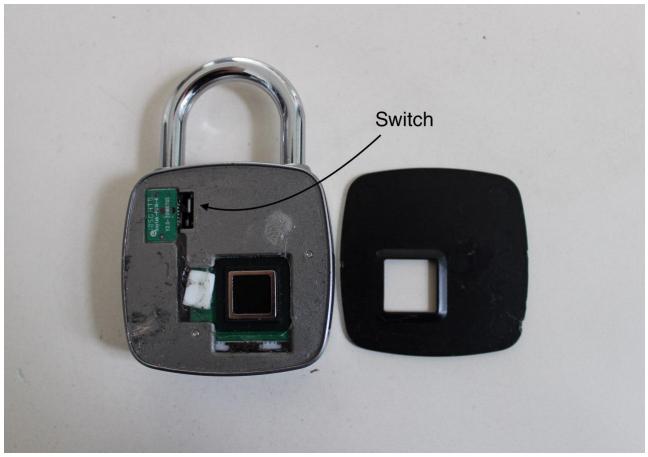


Fig. 9: The switch underneath the front panel

For an adversary this attack would be especially easy if the lock was used outside and therefore had been exposed to the sun for a few hours. The glue underneath the panel would soften and make it even easier to remove it. After using the switch, the adversary could put the panel back in its place, without leaving a trace. This switch is never mentioned in the instructions or anywhere else, the user would have no knowledge of this security gap.

We also tried to create an artificial fingerprint, using the method with the real finger and modelling clay. We tried this attack with a lot of different materials, in a lot of different combinations. We started by using putty, in this case, Play-Doh, as the mould and gelatin as the liquid [44]. To create the gelatin liquid, we used 27g of gelatin and 150ml of hot water to dissolve it. After filling the mould with the gelatin liquid, we quickly realized that Play-Doh is not the best type of modelling clay to use. The liquid is very hot and even though we cooled the Play-Doh after creating the mould, it started to melt as soon as we poured in the gelatin. This resulted in the mixing of both, the gummy finger and the putty, therefore the resulting artificial fingerprint was not as clear as we wanted it to be, the ridges and valleys could not be distinguished very well. Another problem with this first try was the gelatin to water ratio, there was to much water which resulted in a very runny and unstable gummy finger. Although this finger did not turn out as we wanted, we still tried it on the scanner and although we were not able to fool the scanner, at least it recognized the gummy finger as a real finger and rejected the fingerprint. The scanner only reacts if it thinks a finger has been placed on the scanner, for example when pressing fabric onto the scanner, it does not react at all. Then we started experimenting with different types of materials for the mould. In the end, Fimo gave us the best result [45]. Fimo is a firmer type of putty that can be moulded easily but at the same time shows the imprints of the finger very well. After baking, it hardens completely, this way, the mould can be used again and again. The next problem was trying to get the gelatin to water ratio right. We needed enough water to still be able to melt the gelatin but at the same time, the gummy finger needed to be firm and hold it's shape. We found that, although it takes quite a long time to melt, the best ratio was a 1:1 ratio, mixing 30g of gelatin with 30ml of hot water. Since the water cools down relatively quickly, the mixture had to further be heated on the stove to fully melt. In the end, we created a relatively good replica of the original finger, the ridges and valleys could be seen clearly, both in the mould and in the artificial fingerprint. To be sure of this we used the Cherry FingerTIP ID Mouse [46], a mouse with a fingerprint reader that is able to transform the scanned fingerprint into a picture.

(a) Original fingerprint **(b)** Gummy fingerprint

Fig. 10: Original fingerprint and gummy fingerprint compared

As can be seen in figure 10 the overall pattern is discernible although the gummy finger is not as flexible as a real finger and therefore can not be pressed onto the scanner in the same way. Unfortunately, we were not able to fool the fingerprint scanner with our replica. An idea was that maybe the scanner could be temperature sensitive and since the gelatin needed to be in the fridge to fully harden, there might have been a problem with the temperature of the finger. To bypass this problem, we tried to cut the gelatin finger as thin as possible and put it on top of a real finger so that the warmth of the finger would get through to the scanner. But in the end, it did not make a difference. A different thought was, that although gelatin consists out of animal products, it might still not be as conductive as human skin.

The Chaos Computer Club (CCC) used conductive varnish while trying to fool the fingerprint scanner of the iPhone 5s. They used it to improve the conductivity of their artificial fingerprint by simply spraying it onto white glue that had an imprint of the fingerprint [40]. We tried the same with our gummy fingers but it again did not make any difference. All of the attacks on the fingerprint scanner had the same effect on fingerprint padlock number 2.

3) Fingerprint padlock number 2 [47]: This padlock has a metallic casing all around, the shackle being out of the same material. Same as padlock number 1, it has a capacitive scanner and a LED light next to it that indicates the status of the system. (see figure 11)

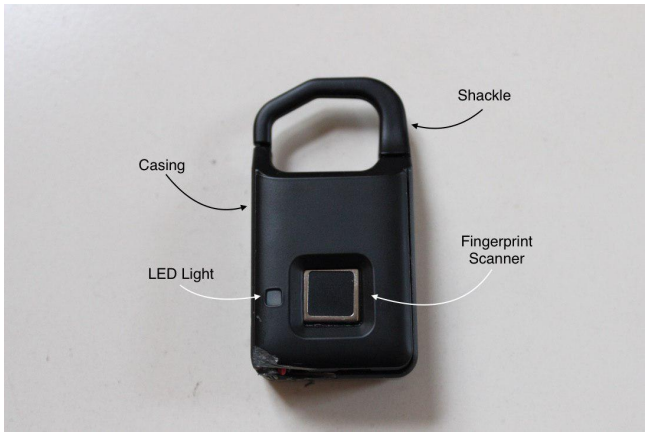


Fig. 11: Fingerprint padlock number 2 with annotations

This lock arrived later during the course of the project and turned out to have the same packaging as the first lock, with no company name anywhere. Since the functionalities of both locks, especially the ones of the scanners, are identical, we believe that they come from the same manufacturer. The only differences are the exteriors of the locks and the locking mechanisms. The second lock does not have a spring-loaded catch but rather an audible motor that closes the shackle. (see figure 12) That is why the shimming attack doesn't work on this lock, but since the scanners of both locks are identical, the problem with the deletion of a fingerprint remains the same.

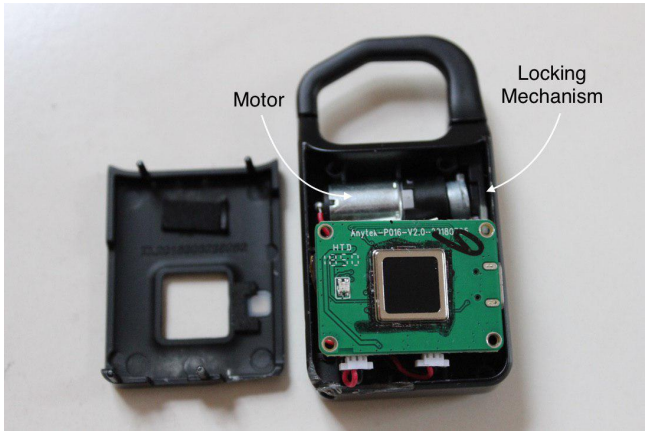


Fig. 12: The motor and the locking mechanism

During the course of the project, we decided to open up the exterior of the second padlock since the scanners are identical and we had hoped that more information about the scanner might help us with possible attacks. As can be seen in figure 12, this lock doesn't have a front plate that can simply be lifted, the design of the casing is much sturdier. To remove the front, we had to use a lot of force and a drill. Therefore, this is nothing an adversary could quickly do on the street and it would not be helpful to them even if they managed it. There is no emergency switch like with padlock number 1 and in the end, it would be quicker for them to simply break the shackle. But this is not the purpose of this project and therefore we do not consider this an attack.

Another reason why we decided to open this lock is that we found out that when we connected one of the locks to a computer, via the charging cable, it appeared as an external hard drive. It only appeared for a few seconds and then quickly disappeared again. We thought about trying a USB-attack but when listening to the communication between lock and computer we did not have a lot of success and therefore decided to not go further into the topic. As mentioned in the attack model, we also looked into manipulating the firmware and therefore we needed more information about the microchip inside the lock. To use a hex file, we would need to know which specific pins of the chip are connected to the fingerprint scanner since we only wanted to manipulate those functionalities. For this we would have needed to further work with the circuit board. Since this would have gone beyond the scope of the project, this is explained in more detail in section VI "Future Work".

C. Server Client Architecture

The client-server model is a basic computer science programming paradigm. Typical problems which get solved by this model are the central storage of data, the distribution of data over multiple devices or the execution of computationally heavy tasks. The essential element of the whole concept is the exchange of messages between client and server. Therefore it should be obvious that this is a vulnerable point of attack. In the early years of the Internet, nearly no service offered any kind of protection, so it was quite easy to passively listen or actively manipulate the data transferred between the communication partners. This only changed when connections where secured with SSH and SSL (now TLS) [48].

1) Attack Models:

a) MAN IN THE MIDDLE (MITM): For a MITM attack, the attacker tries to set up an instance under his control in-between the actual communicating parties, which pretends to both sides to be the other partner. We used a tool called MITMProxy to perform such an attack in a real world scenario. By forcing network traffic trough the proxy one can have a look into (encrypted) communication. MITMProxy makes the client believe that he is talking with the host requested and takes the role of a client in terms of the server. This approach works out of the box for unencrypted (HTTP) but not for encrypted (HTTPS) traffic. To protect users from a MITM attack, the servers authenticate themselves with certificates. MITMProxy solves this problem by creating new certificates for each requested host on the fly. To make the client believe that the certificates

generated by MITMProxy are valid, one has to trust its Root CA by importing the MITMProxy Root CA certificate [49].

b) BRUTE FORCE: Within this model, an attacker tries to break something with exhaustive search. In a client-server architecture, this could for example involve online attacks like breaking into an account by trying all possible passwords or offline attacks like recovering a hash's value (e.g. a hashed password) or finding an encryption key of a previously recorded communication.

c) EXPLOITS: Within this model, an attacker searches for or uses known weaknesses in the program to be attacked (in our case the app or the web API). For the www-service of the internet, SQL-Injection [50] and XSS [51] are very popular. These attacks are very powerful because they often grant an attacker access to the underlying database.

2) Slok: The Slok is a decent looking, Red Dot Design Award winning smart lock. Its system consists of three components, namely the smart lock itself, the app installed on a mobile device and the server API. To allow a long service life of the lock, without recharging or changing the battery, it is only equipped with a Bluetooth Low Energy communication module. Therefore it relies on a mobile device providing the internet connection.
After downloading the app and creating an account, the lock can be bound to an account. To open the lock, the mobile device must be connected to the internet and the app must be opened. Then one presses the shackle of the lock to activate the transmission mode. If then the lock is detected, the app enables the open button. Pressing the button then opens the lock with a little "click"-sound.
Unfortunately the whole system contains some serious design flaws which will be discussed in detail in the next sections.

a) HARDWARE VULNERABILITY - SCREWS: While we opened the box, even before taking the lock out of place, a little screwdriver has fell out of the box. The intended use of this tool is to open the casing of the lock to replace the battery. So we directly opened the lock to see what the internals look like - a small compartment for the battery, a red and black wire and some red glued things which looked a slightly like screws (see figure 13). Later investigation by removing the Loctite (screw glue) confirmed our guess. To find out more about the production of the look and see which electronics were inside casing, we decided to take the lock apart as much as possible. Removing the inner screws enabled us to open the opposite case cover too. Right at the top of the second compartment, we found the main PCB of the lock. Carefully moving the PCB out of the way, opened the view to the internal locking mechanism. The system is quite simple the shackle, a little engine and a shackle-locking-peace (see figure 14). By removing the locking piece, one can separate the whole lower section of the lock from its shackle.
An attacker would be able to disassemble and later reassemble the lock while only leaving the two inner screws without Loctite.

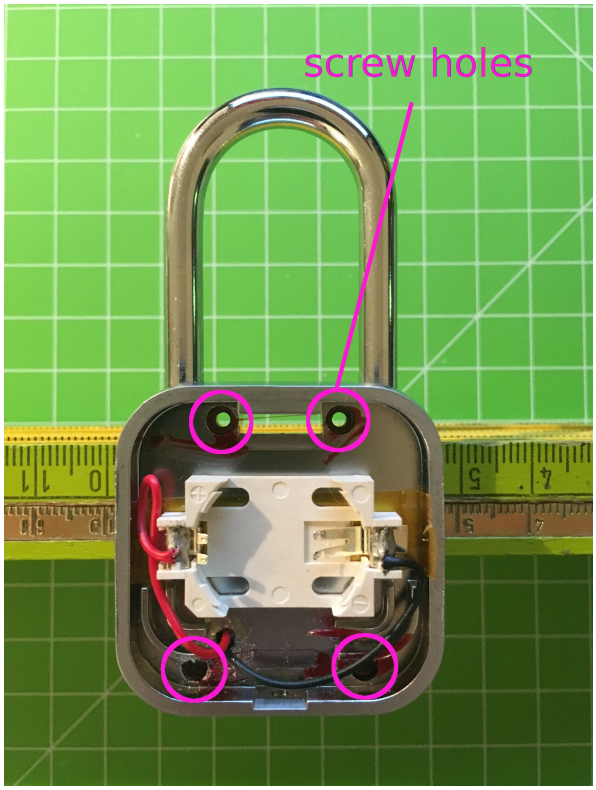


Fig. 13: The slok smartlock

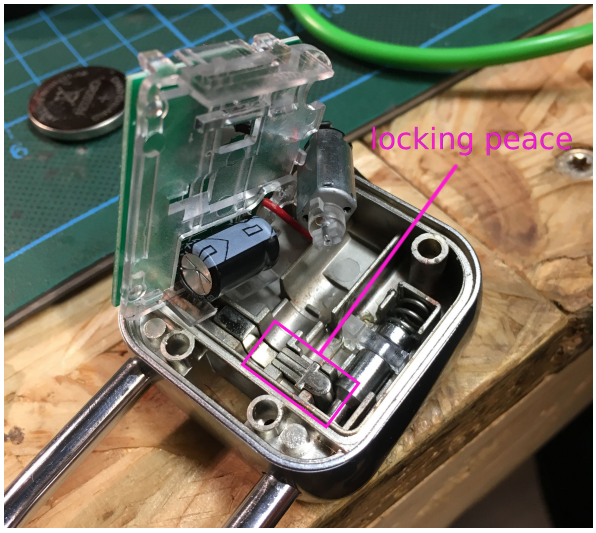


Fig. 14: The slok smartlock

b) SOFTWARE VULNERABILITY: Independent from the previously explained hardware vulnerability, we also analysed the software side of the lock.

As the lock requires an internet connection via mobile device, it was the first logical step to use MITMProxy to see what is actually sent over to the server. With MITMProxy running we registered the lock to our account and could see that five requests were sent to the URL you-lock.com/index.php. Here right away we discovered the first design mistake, as the requests were only using unencrypted HTTP instead of HTTPS. Apart from that, we were lucky, because the request bodies were in the human readable JSON format. This enabled us to compare them and the first thing we noticed here, was that they all had a similar structure. The number and type of the data fields varied but always contained the fields ”code” and ”vkey”. At this point we were able to say, that the request with code 219 must have been related to the registration, because the given email address was only part of this type of request. We continued using different functions of the app like adding a lock, opening a lock and deleting a lock, to follow the requests in MITMProxy which all looked as similar as before.

To get an idea what these requests are good for, we used JADX, a tool which converts Android dex-files back to Java files, to have a look into the source code of the app. Luckily again, the app was not obfuscated and searching for the code numbers delivered a file containing all possible codes and their meaning. Knowing the possible request codes, we tried sending a request with some random data which directly failed with an error message. Something must have been wrong with the data of our request, so we decided to go one step back and check if a previously recorded request would be successful. It was, which also showed, that some kind of validation mechanism must be included within the requests. Comparing different requests, in respect to the number of parameters sent to the API, it showed that the ”vkey” field was the only place were validation data could have been included. Again we searched the source code to find out how the ”vkey” is created. It showed that the mechanism is quite simple - first encode all data fields as URL string sorted by the field name, then append the md5-hash of the accounts password and finally hash everything again with md5. With that knowledge we built a small python API client which helped to do automatic testing and data collection.

After we finished the analysis of the client-server messaging, we thought about possible attacks for our knowledge. We came up with two types of attacks, namely a online / offline brute force attack and a lock hijacking attack.

The online brute forcing attack is based on the fact, that all requests after the registration request contain a field ”user_id”. If one has seen one of these requests containing the ”user_id” one can pick its value. Then one only needs to create all possible passwords and request the API with an arbitrary request for each password. If no error is returned, the password is valid. This approach is quite simple but takes a long time to finish, as the request takes a moment to complete. It should also be mentioned, that there is the chance, that the API locks the client doing so many requests after each other.

The offline brute forcing attack is a little smarter, as one only needs one valid request to the API to recover the password. As explained above, the ”vkey” field contained in all requests contains the hashed password, so the password recovery problem is reduced to calculating the md5 hash of a fixed string concatenated with the md5 hash of a password. This could be done quite efficiently by using hashcat [52], a high performance hash calculation tool. In our experiments with an older consumer graphics card, we achieved a recovery speed of around 17 days for an eight character long password containing the character a to z in lower- and uppercase and the numbers 0 to 9. This could be improved by using more and more modern graphic cards in parallel.

The hijacking attack is based on the idea of adding the lock to an unauthorised account by using an own implementation of the Slok API. Creating an account and adding the lock to that account is quite simple, as the API mostly answers a simple ”one” for successful requests. The problem is that the lock key is needed to open the lock. It is transmitted when a lock is registered and when asking for the lock being registered to the account. So one can not simply ask for the locks of another user, as this would require having or breaking the users password, which is then the attack explained above.

So we did some investigation on the key and found out that it is constant for each lock. Adding the lock to different accounts didn’t change the key, neither changing the lock’s name, so the only influencing parameter is the lock’s MAC address. The key also could not be a randomly generated value, because the lock would not be able to validate it based on the data available. As all tests did not work with so little data, we used our python client to retrieve the key for 1000 imaginary locks. The interesting fact was that all started with the sequence ”01”. This strengthened our belief that there must be an algorithm to calculate the key from the MAC address. We tried different ideas but no one of them worked. So far this is the last open question regarding the lock’s API.

We also tried a short test on SQL injection, but the API seemed to be protected against these kind of attacks. As this kind of attack is also close to being illegal, we did not continue this kind of attack approach.

3) Chamberlain 830 REV: In contrast to the previously discussed ”Slok”, the Chamberlain 830 REV is not a smart lock itself, it is a bridge device to add smart capabilities to your existing Chamberlain garage-door opener.

The system consists of two components, namely the

hardware device and an app to be installed on an Android or iOS Device. within the shipping package you will also find a light barrier, which is connected to the garage door opener to prevent it from closing when someone or something is underneath.
The setup process is quite simple, connect the hardware device via cable to your LAN, then plug in the power. After registering your account and adding the bridge device with its serial number, the garage door opener could be added [53].
We never went so far, as we only had the bridge device without an actual garage door opener, but we gave out best to investigate on both components.

a) INVESTIGATION OF THE HARDWARE: As the bridge's hardware device is designed to be plug-and-play without any user configuration, investigation gets a little bit more tricky. For the Slok device, we could exploit the fact that the lock relies on the mobile device being its access point to the internet. By forcing the whole device traffic through MITMProxy, we where able to see the whole communication in real time. Unfortunately, this approach does not work with the bridge's hardware device, as there is no configuration intended.
To take a look at the communication anyway, we used a modified router running OpenWRT. This gave us the chance to run tcpdump, a powerful network package filter tool, to look for TCP packages from and to the hardware device. Unfortunately no really interesting packages where transmitted.
b) INVESTIGATION OF THE SOFTWARE: After the investigation of the hardware wasn't successful, we continues with the analysis of the app. We took the same approach as for the Slok smart lock - run MITMProxy, import the root CA and set the smart phone's gateway to the PC running MITMProxy. The result of this procedure was that we were unable to lock in to our account. First we thought, this could be related to a problem with the somewhat older version of the Android device, so we tested the same setup on a current iOS device, but got the same error seen before. To double check that it is not an error of the app, we removed the MITMProxy setup from the mobile devices and were then able to login in to our account.
This showed that there must be some kind of security mechanism built into the app, which prevented us from intercepting the connection between app and server. A research on possible reasons for this behavior showed that most likely a security technique called "certificate pinning" is responsible for creating the problem. Here the expected certificate of a host, e.g. the API to contact, is stored locally and then compared with the received certificate. As MITMProxy generates new certificates for each requested host, the expected certificate does not match the received one and therefore the connection is closed. The developers of MITMProxy state, that there is no chance to get around this problem without modifying the app [54].
As we were already unable to see which data is transmitted between the server and the app, we decided to not continue investigation here.

D. Bluetooth Low Energy

General Information Bluetooth Low Energy is a wireless personal area network technology that is already native on many platforms, Android, iOS and Linux among them. It is intended for use in smart home, healthcare, fitness and security applications.
Bluetooth Low Energy (BLE) has the same communication range of about 10 to 20 meters as Bluetooth Classic. However, it has much faster pairing and connection times (Classic Bluetooth 30 seconds, BLE 3 seconds). Both use the 2.4 GHz radio frequencies. The chips are low cost and very small.
Due to its low power consumption, devices using Bluetooth Low energy can run for months or years on just a button cell. This enables the utilization of the technology for communication with proximity sensors, heart rate monitors, and fitness devices, which all only have little power available.
Generic Attribute ... Communication is based on the Attribute Protocol (ATT). It specifies how two BLE devices send and receive messages. Bluetooth Low Energy application profiles are all based on the the Generic Attribute Profile (GATT), which is built on top of the ATT. It specifies the sending and receiving short pieces of data, so called attributes, over a Bluetooth Low Energy link [55].
The profiles [56] define standard ways in which services, characteristics (discovering, reading, writing, notifying and indicating) and their descriptors can be found and then used to transfer data. Thus, the profiles specify how a device works in a particular application. One device can offer multiple profiles. Some standard examples would be a blood pressure profile, an environmental sensing profile or a location and navigation profile, but many others are available. Each profile is comprised of one or more services necessary to fulfill a certain use case. The appropriate implementation is up to the manufacturer [57] [56].
A profile offers one or more services to other devices, with a service being a collection of characteristics. They consist of some properties, a value and descriptors that describe a characteristic value. The descriptor could, for example, contain a human-readable description of the value or an acceptable range for the value. A service could be a Heart Rate Monitor, it's characteristic could be the heart rate measurement [57][56].
Connection Unlike in Bluetooth Classic, the connection between two devices is not exclusive. No permanent connection between the devices is needed. Due to the hierarchic organization of the profiles, the data transferred between devices is kept as small as possible. These properties also save on battery power [58].

Communication happens between a *central* device, in our case that would be a phone; and one or more *peripheral* devices, this would be the smart locks.
Devices scan the advertisement channel at specific intervals. This means that each phase, they send data packages to and listen to the advertisement channel. When two devices happen to be sending and listening in the same phase, they can connect and change into a data channel.
Bluetooth applications should use link layer security for the connection. To ensure the security, two devices need to *pair*, that is to authenticate the identities of the devices, encrypt the link and find some kind of key for further communication. For pairing (”handshake exchange” between two BLE devices at first connection), the BLE standard offers JustWorks (a 6-digit PIN that is just zeros), the 6-digit PIN and Out of band (additional data is transferred on a channel that is not BLE, for example by NFC). For bonding (”for connecting again later), the devices should establish and store a long term key [8]. Lastly, encryption should be used while exchanging data [59]. In many devices, AES and a static secret are used. Many devices (including smart locks) do not implement the link layer security measures. An AES-128 block cipher is recommended for authenticated communication. Often, the developers write their own communication protocol with an unencrypted BLE link, which makes interception possible [60][59][61].

1) Attack Model: Man in the middle: In the *Man in the middle* (MITM) attack, the adversary tries to emulate the peripheral device, in our case the smart lock, in order to trick the central device, into a connection. Once the connection is established, the adversary can monitor data exchanged between the devices [60].
The *GATTack* application implements a MITM attack. Devices that do not implement pairing and link-layer encryption are vulnerable to an attack [60]. The adversary only needs a Bluetooth adapter and a Raspberry Pi or similar. GATTack emulates the peripheral devices, tricking the app into a connection, then gets the data sent between the two original devices. The data is still forwarded between the two original devices, but the adversary can use GATTack not only to intercept, but also to modify requests and responses [62][8]. The setup for the attack can be seen in figure 15.
It can be used for different attack scenarios: disrupting functionality, so the user can not get service from BLE device (Denial of Service), spoofing (giving wrong information to the user), intercepting data (to gain personal information,), controlling the device and more [62][8]. *nRF Connect* is an app which can be used for Bluetooth scanning, reading and writing functionalities, to monitor Bluetooth connections on a device. This enables the user to get information, e.g. IP or device type, read advertisements and services and even clone a device. The application also offers ”Macros”, which enable the user to run and replay Bluetooth exchanges by using a simple XML file [63]. By converting data overheard with GATTack to an XML file, we could use nRF Connect

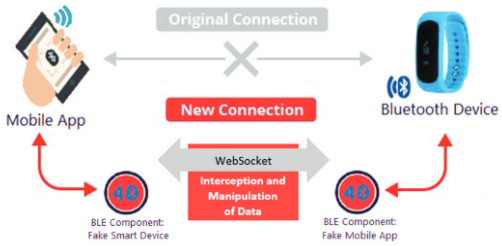


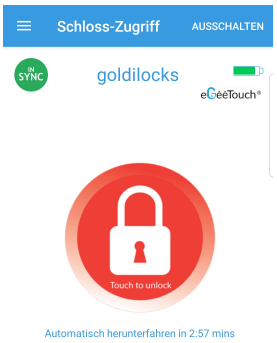
Fig. 15: The architecture for the MITM attack on a BLE device. Graphic by [8]

to replay it on any device [60]. *BLECryptracer* [64] can be used to examine Android applications files and search for cryptographically processed BLE communications [64][65]. Since often, applications that use BLE do not implement security measures, this can be used to find vulnerable applications.

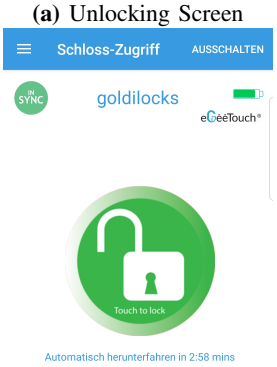
2) Smart Travel Padlock: As described before, the eGeeTouch Smart Travel Padlock GT1000-96 [19] seen in figure 4 is a padlock that can be unlocked via Bluetooth Low Energy using an app and via NFC using the NFC-tag. Even though there is no physical key (except the NFC-tag) provided to the buyer, the lock does have a TSA lock (as seen in 16 for access by (authorized) TSA personnel. It should be noted, however, that the key is available online cheaply and so are blueprints for 3D-printing it, and has been for some time. This already violates one of our security principles, as non-authorized people can obtain a duplicate of the key, and thus open this lock. To use the BLE function,

Fig. 16: The Travel Padlock’s side, with the TSA keyhole

it is neccessary to download the ”eGeeTouch Manager” app [66]. Account registration via E-Mail is required before the app can be used. Then, app and lock can be paired by pressing the power button and using the app at the same time. To distinguish between different locks, a name can be given to the lock. Once the devices are paired, the lock

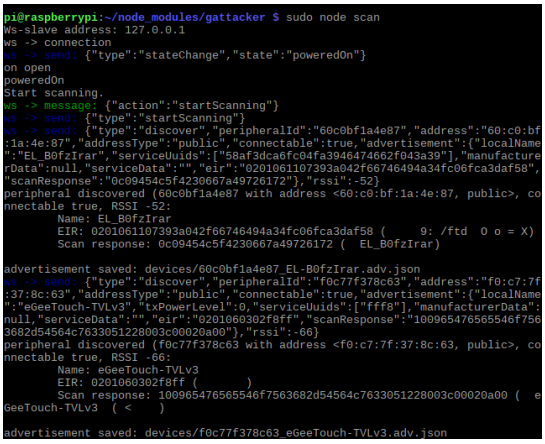


(a) Unlocking Screen

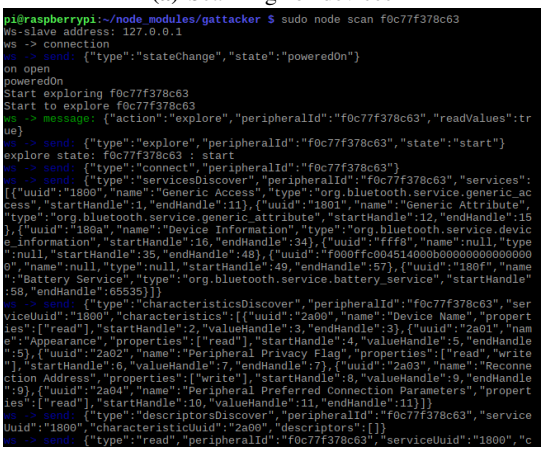


(b) Locking Screen

Fig. 17: The eGeeTouch Manager app [66]



(a) Scanning for devices



(b) Exploring the eGeeTouch Lock

Fig. 18: Getting the information from the lock using GATTacker

can be locked and unlocked by pressing the power button and the app button (seen in 17), the battery status of the lock is displayed and a function to determine distance from the lock is available, however this did not work with our phone. The app also has a screen for getting the last times the lock was unlocked from the lock, but getting this data also proved not to be possible. It is possible to share a guest key with other users registered with the eGeeTouch app.

GATTacker A MITM attack on the lock is possible by using GATTacker [67]. First, the tool can be used to scan for available devices, gaining information such as the MAC address, which is saved in a .json file. If an interesting device is found, the user can scan this device to read characteristics and services [67]. All information can be obtained from the eGeeTouch lock. A sample of the process can be seen in figure 18.

After this, we have all the information necessary to impersonate this device. Now two modules can be run either on two different devices, communicating for example via a web socket, or on two virtual machines on the same computer [8]. First, the peripheral device of which we collected information (in our case the smart lock) is cloned, in order to trick the central device into a connection. This is possible because all the information on bonding is obtained. To the phone, the cloned device appears just like the real one, and pairing again is possible without a new key exchange. The second module then impersonates the central device. Now the attacker is connected to both devices and can monitor communication.

Hook functions [67] can be used to manipulate communications. This enables us to unlock the padlock when the power button press is registered, no matter what the real app replies. A sample hook function can be seen in 19. Alternatively, the monitored information can be converted to an XML file [60] and replayed by nRF Connect [63][60]. The BLECryptracer tool [64] can be run on the .apk file of the app, which can be downloaded online. There are two tools for analysing the .apk. First, the *getvalue* script analyzes BLE reads. On the other hand, the

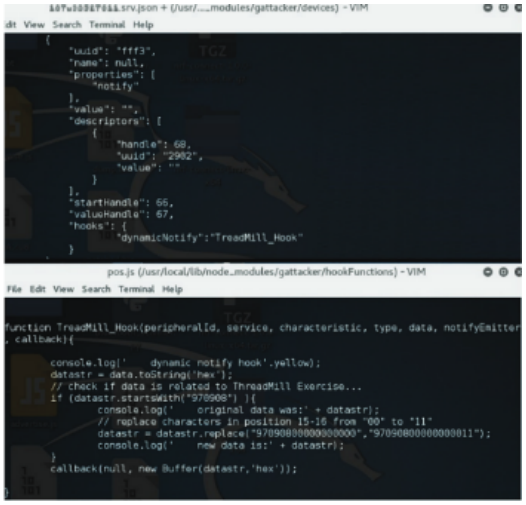


Fig. 19: A sample hook function. Image by [8].

Fig. 20: The Yeelock and and its accessories. Image from https://www.nextsmarthome.de/yeelock-das-smartes-schloss-im-test/

Fig. 21: The screen for unlocking the Yeelock.

setvalue script analyzes BLE writes [64][65]. Since we can use the GATTack tool on this lock, we already know that the security measures are not implemented correctly.

3) Yeelock: The Yeelock lock for drawers can be seen in figure 20. While some sellers, such as this one on eBay [68], cite Xiaomi as the manufacturer, the actual lock and its packaging do not. It is a drawer lock made up of two components. It has to be screwed to the inside of the drawer the user wants to protect.

This lock can only be opened via BLE by using the "Yeelock" app [69]. The app is written in a combination of English and Chinese. First, the user needs to register an account using a valid phone number. Then, the lock can be registered to this account by scanning a QR code provided in the manual. Unfortunately, the manual was completely in Chinese, no English instructions were provided with the lock.

To unlock the lock, only a button press is needed (see 21. After a few seconds, it automatically locks again. Closing the drawer is still possible later due to the physical locking mechanism. This app keeps a log of all the times the lock was unlocked by a user. Keys can be shared to any person using a link or a phone number. The lock can also be transferred to another account. The functionalities on this app seem more thought out than those of the eGeeTouch padlock IV-D.2.

However, using the GATTack tool [67] also yields the results named with the eGeeTouch padlock IV-D.2. The communication is not protected by encryption, enabling an attacker to read the necessary information, before taking over the communication.

V. Conclusion

In conclusion, smart locks, in general, seem to have many vulnerabilities. It is important to note that we only had a look at locks from the lower price range, nevertheless, they oftentimes did not even fulfill the lowest security standards. Using an unsafe NFC-tag that is not even built for security systems as a key or allowing to add additional keys that are possibly unsafe are loopholes that can and will easily be abused by any adversary.

Often the simplest things were overlooked, for example in

the case of the fingerprint locks, we were not able to fool the scanner, but the most basic hardware principles were ignored. Giving an adversary entrance, not only by using a very vulnerable type of catch with known weaknesses but also through an actual switch.

The Bluetooth Low Energy devices were vulnerable to our attacks because the security standards were not implemented correctly or implemented at all. Since the locks are intended to protect important objects, security should be a priority, not an afterthought. The Bluetooth SIG provides standards and suggestions on how to implement them, so developers should make use of them to protect consumers and maintain a quality standard that consumers can actually trust. As our research showed, some very simple principles of data protection have not been considered - so it should become a standard to use encryption technologies to protect the user's private data against attacks of any kind.

There is a lot left to be done for smart locks until they can call themselves truly "smart" and our findings are probably just the tip of the iceberg. (Hanna, Valerie, Ben, Pia)

VI. Future Work

To continue the work on NFC locks, we would propose concentrating on finding a way to copy the data from the key tag to an empty tag. NFCGate is able to clone and use the UID of NFC-tags [3] but we were not able to do the same when writing on an empty tag using nfcpy. Achieving this would be an immense vulnerability because this way multiple key copies could be generated and used by multiple attackers. It would most likely also solve the problem of attacking the *ROTHULT* NFC lock as a working copy of an unsafe additional key should open the lock without any problem. In addition we propose rooting a smartphone that works as an additional key for this lock (as the Sony xperia xCompact does) and using the clone mode as well as the relay mode of the NFCGate app to try and pick the lock. (Hanna)

Concerning the fingerprint locks, the suggested next step would be to look into the manipulation of the firmware by getting a better look at the microchip and finding the specific pins connected to the scanner.

In the case of our locks, to find those pins one would need to carefully braze off each element of the circuit board, then file the board down and take photos of each layer to follow the paths of the different pins.

With this information, one could specifically address the pins connected to the scanner and create the hex file accordingly. This hex file could then be written in a way so that the lock always opens when the scanner rejects a fingerprint. (Valerie)

For the Slok smart lock the only unanswered question remaining is, how the lock's key is generated. As we were unable to find an algorithm which creates the key based on the MAC address and we do not have any chance of getting access to the server side source code, the only possible option left is the analysis of the lock's firmware. As our research showed, the lock does not open if a wrong key is given, so there must be some kind of validation mechanism hidden inside the firmware. Therefore it should be possible to create an image of the lock's firmware through the SWD port on the PCB's front for deeper investigation. If it's possible to figure out how this mechanism works, every lock could be hijacked by using a custom API. (Ben)

As we have seen, there exist several useful tools for MITM attacks on Bluetooth Low Energy devices, which are also applicable to our locks. Another line of inquiry would be during the pairing process. A significant weakness of the pairing process is that an attacker can force the devices to choose a very short key, which can then by cracked by a brute-force attack [8]. This way, the attacker can also eavesdrop on the communication. In the future, it would be interesting to compare this method to the MITM attacks done here, to see which performs better. (Pia)

All in all, we have seen some interesting results in the limited time we had available. However some lines of inquiries should still be explored, and customer's attention for the security risks needs to be raised.

References

[1] Stephen J. Jones, *Access Granted: On the Security of Near-Field Enabled Keycards*. PhD thesis, 2014.

[2] David Wegemer and Max Maass, "NFCGate Project on GitHub." https://github.com/nfcgate/nfcgate. (accessed: 19.04.2019).

[3] David Wegemer and Max Maass, "NFCGate - NFC security analysis with smartphones." https://www.youtube.com/watch?v=PSbfRvzmUII. (accessed: 19.04.2019).

[4] Matsumoto, Tsutomu and Matsumoto, Hiroyuki and Yamada, Koji and Hoshino, Satoshi, "Impact of Artificial Gummy Fingers on Fingerprint Systems," *Datenschutz und Datensicherheit*, vol. 26, 04 2002. (accessed: 05.10.2019).

[5] A. Ornaghi and M. Valleri, "Man in the middle attacks," in *Blackhat Conference Europe*, vol. 1045, 2003.

[6] F. Callegati, W. Cerroni, and M. Ramilli, "Man-in-the-middle attack to the https protocol," *IEEE Security & Privacy*, vol. 7, no. 1, pp. 78–81, 2009.

[7] Jasek, Slawomir, "Gattacking Bluetooth Smart Devices." https://github.com/securing/docs/raw/master/whitepaper.pdf. (accessed: 02.10.2019).

[8] Melamed, Tal, "An active man-in-the-middle attack on bluetooth smart devices," 02 2018.

[9] Ho, Grant and Leung, Derek and Mishra, Pratyush and Hosseini, Ashkan and Song, Dawn and Wagner, David, "Smart Locks: Lessons for Securing Commodity Internet of Things Devices," in *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security*, pp. 461–472, 05 2016.

[10] Elektronik Kompendium, "NFC - Near Field Communication." https://www.elektronik-kompendium.de/sites/kom/1107181.html. (accessed: 04.10.2019).

[11] smart-TEC GmbH & Co. KG, "RFID-Technologie." https://www.smart-tec.com/de/auto-id-welt/rfid-technologie. (accessed: 06.10.2019).

[12] RFID Grundlagen, "RFID." https://www.rfid-grundlagen.de/. (accessed: 04.10.2019).

[13] Fakir, "Unterschied RFID und NFC Eine kurze Erklrung." https://www.fakir.it/unterschied-rfid-und-nfc-eine-kurze-erklaerung/. (accessed: 19.04.2019).

[14] NFC Forum, "NFC - What it does." https://nfc-forum.org/what-is-nfc/what-it-does/. (accessed: 20.04.2019).

[15] NFC21 GmbH, "NFC Sticker, 30 mm, NTAG 213, 180 Byte, transparent." https://www.nfc-tag-shop.de/nfc-aufkleber/nfc-sticker-transparent/nfc-sticker-30-mm-ntag-213-180-byte-transparent-456. (accessed: 30.09.2019).

[16] R. Dhamija and J. D. Tygar and M. Hearst, *Why phishing works. In Proceedings of the SIGCHI conference on Human Factors in computing systems*. 2006.

[17] L. Francis and G. Hancke and K. Mayes and K. Markantonakis, *Potential misuse of NFC enabled mobile phones with embedded security elements as contactless attack platforms. In Internet Technology and Secured Transactions, 2009*. 2009.

[18] nfc-tag.de, Inh. Urs Hettich, "NFC Chiptypen." https://www.nfc-tag.de/nfc-chiptypen. (accessed: 30.09.2019).

[19] eGeeTouch, "TSA Travel Lock." https://www.egeetouch.com/de/products/electronic-comb-lock/travel-padlock. (accessed: 26.09.2019).

[20] nfcproxy, "nfcproxy." https://sourceforge.net/projects/nfcproxy/. (accessed: 27.09.2019).

[21] Team Win LLC, "twrp app." https://twrp.me/. (accessed: 27.09.2019).

[22] David Wegemer and Max Maass, "nfcgate releases." https://github.com/nfcgate/nfcgate/releases. (accessed: 27.09.2019).

[23] David Wegemer and Max Maass, "nfcgate server." https://github.com/nfcgate/server. (accessed: 27.09.2019).

[24] David Wegemer and Max Maass, "nfcgate Relay Mode." https://github.com/nfcgate/nfcgate/blob/v2/doc/mode/Relay.md. (accessed: 27.09.2019).

[25] S. Tiedemann, "Python module for near field communication." https://nfcpy.readthedocs.io/en/latest/. (accessed: 30.09.2019).

[26] IDENTIVE, "SCL3711 Contactless USB Smart Card Reader." https://www.identiv.com/products/smart-card-readers/mobile/scl3711/. (accessed: 30.09.2019).

[27] NXP Semiconductors, "NTAG 213/215/216: NFC Forum Type 2 Tag compliant IC with 144/504/888 bytes user memory." https://www.nxp.com/products/rfid-nfc/nfc-hf/ntag/ntag-for-tags-labels/ntag-213-215-216-nfc-forum-type-2-tag-compliant/-ic-with-144-504-888-bytes-user-memory:NTAG213_215_216. (accessed: 04.06.2019).

[28] IKEA, "ROTHULT NFC-Schloss." "https://www.ikea.com/de/de/p/rothult-nfc-schloss-weiss-00359739/. (accessed: 26.09.2019).

[29] NFC21 GmbH, "bersicht ber NFC-Chiptypen." https://www.nfc-tag-shop.de/info/nfc-tag-typen-chipsaetze.html. (accessed: 30.09.2019).

[30] BLOCKARD, "NFC Tag Sticker (10 Stck) selbstklebend — NTAG216 Chip Tags Aufkleber — 888 Byte Speicher — read & write — kompatibel mit allen NFC bzw. RFID Lesegerten & Android Smartphone Apps." https://www.amazon.de/gp/product/B06XH2R5ZP/ref=ppx_yo_dt_b_asin_title_o01_s00?ie=UTF8&psc=1. (accessed: 30.09.2019).

[31] Ludovic Rousseau, "PCSC lite project." https://pcsclite.apdu.fr/. (accessed: 30.09.2019).

[32] lady ada, "About the NDEF Format." https://learn.adafruit.com/adafruit-pn532-rfid-nfc/ndef. (accessed: 30.09.2019).

[33] Davide Maltoni, "Fingerprint Recognition." http://icb12.iiitd.ac.in/Fingerprint-ICB2012.pdf. (accessed: 26.09.2019).

[34] Perpetual Enigma, "Fingerprint Recognition." https://prateekvjoshi.com/2012/07/22/fingerprint-recognition/. (accessed: 26.09.2019).

[35] Bayometric, "Minutiae Based Extraction in Fingerprint Recognition." https://www.bayometric.com/minutiae-based-extraction-fingerprint-recognition/. (accessed: 26.09.2019).

[36] Robert Triggs, "How fingerprint scanners work: optical, capacitive, and ultrasonic variants explained." https://www.androidauthority.com/how-fingerprint-scanners-work-670934/. (accessed: 25.09.2019).

[37] Tom Harris, "How Fingerprint Scanners Work, Analysis." https://computer.howstuffworks.com/fingerprint-scanner4.htm. (accessed: 25.09.2019).

[38] Tom Harris, "How Fingerprint Scanners Work, Capacitance Scanner." https://computer.howstuffworks.com/fingerprint-scanner3.htm. (accessed: 27.09.2019).

[39] Tsutomu Matsumoto, "A Case Study for User Identification." http://web.mit.edu/6.857/OldStuff/Fall03/ref/gummy-slides.pdf. (accessed: 28.09.2019).

[40] Chaos Computer Club - frank, "Chaos Computer Club breaks Apple TouchID." https://www.ccc.de/en/updates/2013/ccc-breaks-apple-touchid. (accessed: 28.09.2019).

[41] Von Malegowski, "How to Open Locks with Padlock Shims." https://www.youtube.com/watch?v=PC-1S4cHUPs". (accessed: 28.09.2019).

[42] Wikipedia, "Shim (lock pick)." https://en.wikipedia.org/wiki/Shim_(lock_pick). (accessed: 28.09.2019).

[43] Amazon, "XciteRC Bgelschloss mit Fingerabdruck-Scanner Vorhngeschloss Schloss." https://www.amazon.de/XciteRC-Bgelschloss-Fingerabdruck-Scanner-/Vorhngeschloss-Schloss/dp/B07MZ439PF/ref=sr_1_10?__mk_de_DE=M&crid=UHHCCB4KQNA1&keywords=fingerabdruck+schloss&qid=1570264015&sprefix=fingerabdr. (accessed: 04.10.2019).

[44] Hasbro", "Play-Doh." https://playdoh.hasbro.com/de-de. (accessed: 04.10.2019).

[45] STAEDTLER, "Was ist FIMO eigentlich? Alles Wissenswerte zur beliebten, bunten Modelliermasse von STAEDTLER." https://www.staedtler.com/de/de/entdecken/was-ist-fimo-eigentlich-alles-wissenswerte-zur/-beliebten-bunten-modelliermasse-von-staedtler/. (accessed: 04.10.2019).

[46] Bromba Biometrics, "Cherry FingerTIP ID Mouse (Siemens ID Mouse Professional)." https://www.bromba.com/tdidme.htm. (accessed: 29.09.2019).

[47] Amazon, "Festnight OWSOO LED Light Intelligentes Fingerabdruck Vorhngeschlo Geeignet fr Haustr, Koffer, Rucksack, Fitnessstudio, Fahrrad, Bro, USB-Aufladung untersttzen." https://www.amazon.de/Intelligentes-Fingerabdruck-Vorhngeschlo/-Fitnessstudio-USB-Aufladung/dp/B07MT5B22T/ref=sr_1_24?__mk_de_DE=M&crid=3P5IYWXABW2XR&keywords=fingerabdruck+schloss&qid=1570263611&sprefix=fingerabdruck. (accessed: 04.10.2019).

[48] Tanenbaum, Andrew S and Van Steen, Maarten, *Distributed systems: principles and paradigms*. Prentice-Hall, 2007.

[49] MITMProxy Developer Team, "How MITMProxy works." https://docs.mitmproxy.org/stable/concepts-howmitmproxyworks/. (accessed: 28.09.2019).

[50] C. Anley, "Advanced sql injection in sql server applications," 2002.

[51] P. Bisht and V. Venkatakrishnan, "Xss-guard: precise dynamic prevention of cross-site scripting attacks," in *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, pp. 23–43, Springer, 2008.

[52] Hashcat Developer Team, "Hashcat website." https://hashcat.net/hashcat/. (accessed: 28.09.2019).

[53] M1Molter - Der Heimwerker, "Garagentor mit Handy ffnen - MyQ Chamberlain installieren und einbauen von M1Molter." https://www.youtube.com/watch?v=x6NhiQhdgKQ. (accessed: 20.09.2019).

[54] MITMProxy Developer Team, "MITMProxy About Certificates." https://docs.mitmproxy.org/stable/concepts-certificates/. (accessed: 28.09.2019).

[55] "Bluetooth low energy overview." https://developer.android.com/guide/topics/connectivity/bluetooth-le. (accessed: 02.09.2019).

[56] Bluetooth SIG, "GATT Specifications." https://www.bluetooth.com/specifications/gatt/. (accessed: 02.09.2019).

[57] "Bluetooth Low Energy." https://de.wikipedia.org/wiki/Bluetooth_Low_Energy, 2019. (accessed: 02.09.2019).

[58] Merz, Alexander, "Golem.de programmiert: BluetoothLE im Eigenbau." https://www.golem.de/news/golem-de-programmiert-bluetoothle-im-eigenbau/-1404-105896-2.html, 4 2014. (accessed: 02.09.2019).

[59] Jasek, Slawomir, "black hat USA 2016 Presentation." https://github.com/securing/docs/blob/master/slides.pdf, 8 2016. (accessed: 02.09.2019).

[60] Jasek, Slawomir, "How to pick a BLE smart lock and cause 'cancer' using just a mobile phone." https://smartlockpicking.com/tutorial/how-to-pick-a-ble-smart-lock-and-cause-cancer/, 10 2017. (accessed: 02.09.2019).
[61] Till at returnfalse.de, "Sicherheitsmechanismen von Bluetooth Low Energy." https://return-false.de/archive/1098. (accessed: 27.09.2019).
[62] "GATTack.io - FAQ." https://gattack.io/. (accessed: 23.09.2019).
[63] "nRF Connect for Mobile - Apps bei Google Play." https://play.google.com/store/apps/details?id=no.nordicsemi.android.mcp&hl=de. (accessed: 23.09.2019).
[64] projectbtle@github, "Github - projectbtle/BLECryptracer." https://github.com/projectbtle/BLECryptracer. (accessed: 05.10.2019).
[65] Pallavi Sivakumaran and Jorge Blasco, "Attacks Against BLE Devices by Co-located Mobile Applications," *CoRR*, vol. abs/1808.03778, 2018.
[66] Digipas Technologies Inc., "eGeeTouch Manager." https://play.google.com/store/apps/details?id=com.egeetouch.egeetouch_manager&hl=en_US. (accessed: 05.10.2019).
[67] Jasek, Slawomir, "Github - securing/gattacker." https://github.com/securing/gattacker, 2016. (accessed: 02.09.2019).
[68] "." https://www.ebay.de/i/163877183719?chn=ps&norover=1&mkevt=1&mkrid=707-134425-41852-0&mkcid=2&itemid=163877183719. (accessed: 05.10.2019).
[69] Xian Yisuobao Electronic Technology Co., Ltd., "Yeelock." https://play.google.com/store/apps/details?id=com.yeeloc.yisuobao&hl=en_US. (accessed: 05.10.2019).