Fingerprints On Mobile Devices: Abusing and Leaking

Yulong Zhang, Zhaofeng Chen, Hui Xue, and Tao Wei
FireEye Labs

I. Introduction

Fingerprint scanners are becoming more and more popular on the modern mobile devices like HTC One Max, Huawei Mate 7, iPhone 5s/6/6+, and Samsung Galaxy S5. Based on a recent projection [1], 50% of smartphone shipments will have a fingerprint sensor by 2019. Those fingerprint scanners are more than just a gimmick to unlock your phone -- they can also let you conveniently authorize money transactions with a swipe of your finger. With the introduction of Apple Pay, mobile payment is going to be a primary driver for fingerprint sensors. Most of Apple's competitors have reacted quickly to match Apple's mobile payment lead in the market.

However, how secure those fingerprint frameworks are designed remains the customers' biggest concern. In the traditional password-based auth systems, victims can easily replace the stolen passwords with a new one. But fingerprints last for a life -- once leaked, they are leaked for the rest of your life. Moreover, fingerprints are usually associated with every citizen's identity, immigration record, etc. It would be a hazard if the attacker can remotely harvest fingerprints in a large scale.

Previously there have been some works focusing on fingerprint spoofing attack [2][3]. These works demonstrate that fingerprints can be stolen from polished surfaces (e.g. smartphone screens) or from a waving hand photo, and can be spoofed using electrically conductive materials. We categorize this type of attack as optical attacks. In this paper, we will rather focus on the system attacks of mobile fingerprint auth framework. To our knowledge, we are the first to discuss this type of threats.

We will analyze the mobile fingerprint authentication and authorization frameworks, and discuss several security pitfalls of the current designs, including:

- Confused Authorization Attack
- Unsecure fingerprint data storage
- Trusted fingerprint sensors exposed to the untrusted world
- Backdoor of pre-embedding fingerprints

This paper is structured as follows: we will introduce the background of fingerprint auth framework in Section II, and describe detailed vulnerabilities in Section III. Further discussions will be provided in Section IV. We conclude the paper in Section V.

II. Background

1. Original Mobile Fingerprint Auth Framework

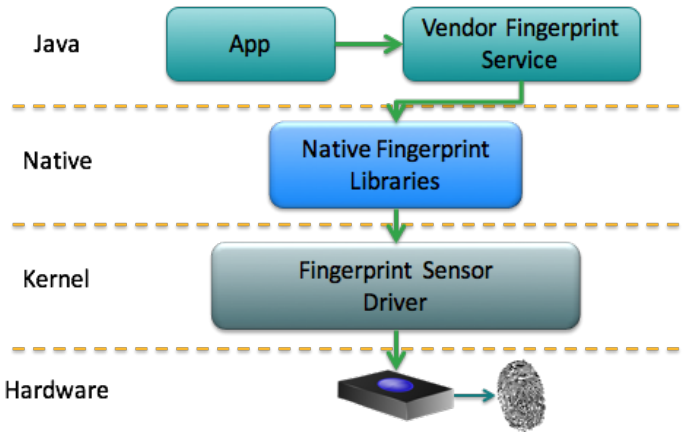


Figure 1: Original mobile fingerprint auth framework

Figure 1 illustrates the normal design of mobile fingerprint authentication/authorization framework. An example device with this design is HTC One Max. In this design, the kernel interacts directly with the fingerprint sensor, and vendor fingerprint libraries (native code) and fingerprint services (Java code) provide functional wrapping of common fingerprint auth operations. The high level auth logic is implemented in specific apps by invoking vendor exposed APIs.

There is an obvious security weakness with this design -- the fingerprints are just as secure as the kernel. If the attacker roots the device, he/she can steal the fingerprint data. Unfortunately there are quite a few public known kernel vulnerabilities that can be exploited to root the majority of Android devices, like Framaroot [4], Towelroot [5], and PingPongRoot [6], etc. Thus the vendors are now moving to a more secured design with the help of ARM TrustZone [7].

2. Mobile Fingerprint Auth Framework with TrustZone

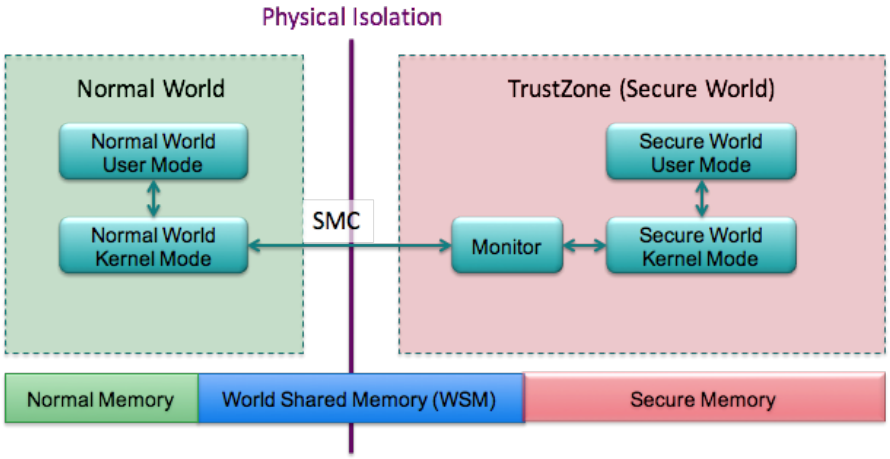


Figure 2: TrustZone isolation on mobile devices

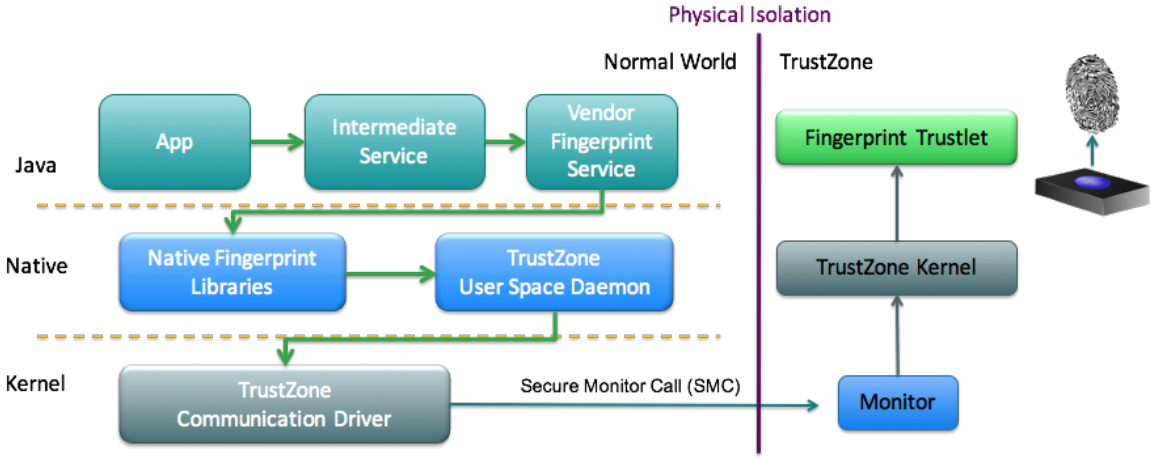


Figure 3: The improved mobile fingerprint auth framework with TrustZone isolation

Figure 2 shows the general architecture for mobile platforms equipped with TrustZone. TrustZone isolates the normal world (the normal user/kernel mode) and the secure world (Trusted Execution Environments, or TEE) by creating additional operating modes, known as the Secure mode and the Monitor mode. The Secure mode has the same capabilities with the normal world while operating in a separate memory space. The TrustZone Monitor acts as a virtual gatekeeper controlling migration between the two worlds. The normal world can issue requests to the secure world via Secure Monitor Calls (SMC).

Since the normal world cannot touch the secure world's memory, fingerprints can be well protected in TrustZone from being accessed by normal world attackers. Figure 3 depicts the enhanced fingerprint auth framework utilizing TrustZone's protection. In this design, the fingerprint sensor driver, fingerprint recognition logic, and the fingerprint data are all isolated in the secure world, so the fingerprint auth framework remains secure even if the normal world kernel is compromised.

Although the enhanced design has been much securer than the original design, in the next section we will reveal that there are still many severe security pitfalls.

III. Vulnerabilities

1. Confused Authorization Attack

The first vulnerability is that all the fingerprint frameworks are prone to the "Confused Authorization Attack", which has long been overlooked. Authorization grants access rights to resources, while authentication verifies who you are. Security systems often mistakenly treat authorization as authentication, or fail to provide context proof for the authorization objects. Without proper context proof, the attacker can mislead the victim to authorize a malicious transaction by disguising it as an authentication or another transaction. For example, as shown in Figure 4, the attacker can easily fake a lock screen to fool the victim to think that he/she is "swiping finger to unlock the device", but the fingerprint is actually used to authorize a money transfer in the background.

Currently the FIDO Alliance is developing the specification of secure authentication and authorization protocols for the mobile ecosystem. As FIDO describes in the specification [12]: "*Basically if a FIDO UAF Authenticator has a transaction confirmation display capability, FIDO UAF architecture makes sure that the system supports What You See is What You Sign mode (WYSIWYS). A number of different use cases can derive from this capability -- mainly related to authorization of transactions (send money, perform a context specific privileged action, confirmation of email/address, etc).*"

FIDO's specification requires that "*the transaction confirmation display component implementing WYSIWYS needs to be trusted*". However, the original fingerprint auth framework has no reliable way to provide the authorization context proof. The framework with TrustZone can be improved to achieve this goal (the Trustlet modules in TrustZone can be modified to provide the context proof), but so far (June 2015) we haven't seen any major vendor that implemented this feature.

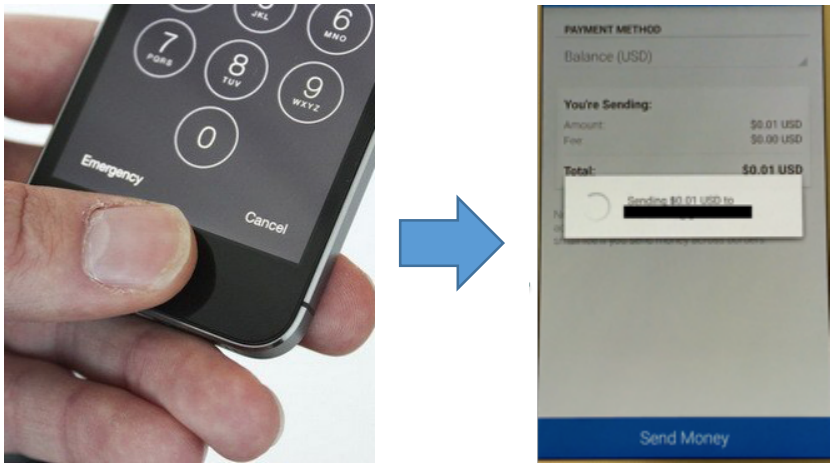


Figure 4: Illustration of using confused authorization attack to authorize money transfer[1]

2. Fingerprint Data Storage Vulnerability

The second issue is that not all the vendors store the fingerprints securely. While some vendors claimed that they store user's fingerprints encrypted in a system partition, they put users' fingerprints in plaintext and in a world-readable place by mistake. One example is HTC One Max -- the fingerprint is saved as /data/dbgraw.bmp with 0666 permission (world-readable)[2]. Any unprivileged processes or apps can steal user's fingerprints by reading this file. Other vendors store fingerprints in TrustZone or Secure Enclave, but there are still known vulnerabilities for attackers to leverage to peek into the secret world [8][9].

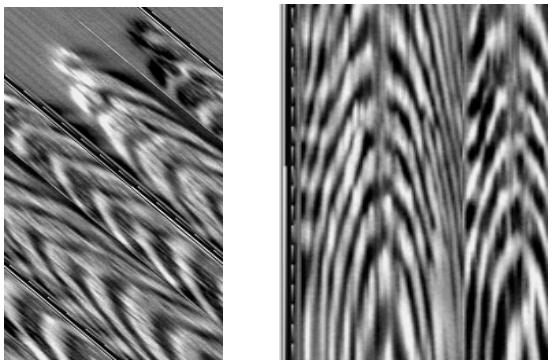
To make the situation even worse, each time the fingerprint sensor is used for auth operation, the auth framework will refresh that fingerprint bitmap to reflect the latest wiped finger. So the attacker can sit in the background and collect the fingerprint image of every swipe of the victim.

Note that there is some specialty of the format of the raw fingerprint bitmap image. Normally the size of each bitmap row is rounded up to a multiple of 4 bytes by padding. But a sample row of the raw fingerprint bitmap looks like:

```
01 FE 02 00 09 09 14 20 60 50 70 70 70 40 60 50
70 70 70 70 70 70 70 70 60 70 60 70 70 80 70 80
80 70 70 70 70 80 90 A0 A0 A0 A0 A0 A0 B0 A0 80
[...]
B0 B0 B0 A0 B0 A0 A0 A0 A0 A0 A0 A0 90 A0 A0 90
90 90 90 90 90 90 80 80 70 B0 70
```

[1] This is a general issue for fingerprint based payment for mobile. The device shown is just for illustration and is not necessarily vulnerable.
[2] HTC has patched this vulnerability per our notification

We observed that all the rows start with 0xFE01. This is probably used to mark the beginning of each fingerprint image line. And each line has 187 bytes of data (including 0xFE01), which is not 4-byte aligned. If one opens the raw fingerprint bitmap image directly, it will look like Figure 5(a). After appropriately padding, it will be correctly look like Figure 5(b). The fingerprint bitmap image is blurry in the bottom and has more details in the upper part. Also, one may notice that the image is visibly separated into two horizontal parts, and the left part is almost twice the width of the right part. This is the design nature of certain sensor chips. We just present the image as it is.



(a) (b)

Figure 5: Fingerprint bitmap obtained from HTC One Max. Both the raw bitmap (a) and the re-aligned version (b) are shown. We only pasted a small portion of the images to protect the fingerprint owner's anonymity.

3. Fingerprint Sensor Exposure Vulnerability

Even if the protection of fingerprint data in TrustZone is indeed trustworthy, it only means that the fingerprints previously registered on the devices are secured. We found that the fingerprint sensor itself in many devices is still exposed to the attackers. Although the ARM architecture enables isolating critical peripherals from being accessed outside TrustZone (e.g. by programming the TrustZone Protection Controller), most vendors fail to utilize this feature to protect fingerprint sensors[3]. To the best of our knowledge, we are the first of the world to put forward the fingerprint sensor spying threats.

[3] As of writing, we have confirmed this vulnerability on HTC One Max, Samsung Galaxy S5, etc. All vendors have provided patches per our notification.

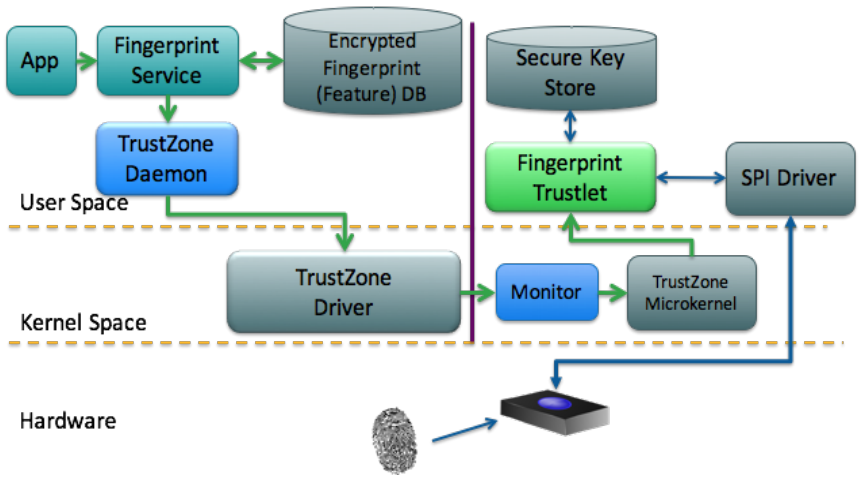


Figure 6: How normal world is supposed to interact with the fingerprint sensor (TrustZone should always act as the mediator)

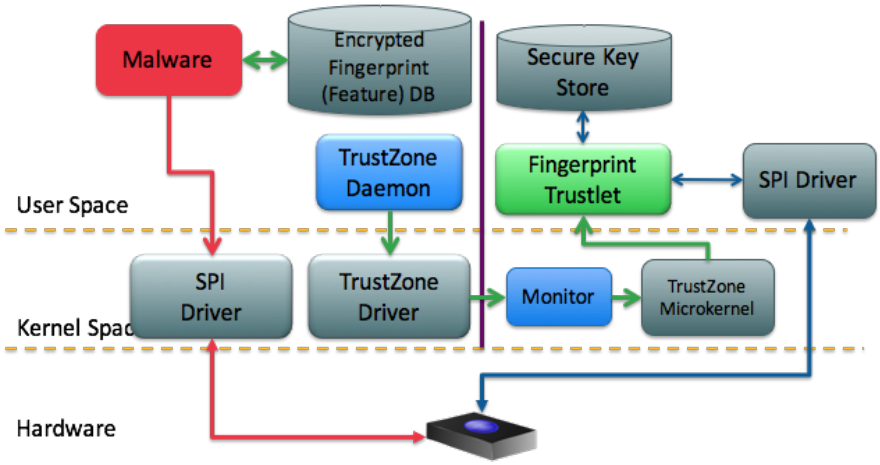


Figure 7: Malware in the normal world can directly read the fingerprint sensor

As shown in Figure 6, instead of directly communicating with the fingerprint sensor, all the normal world components are supposed to invoke the TrustZone fingerprint for sensor operations. However, most vendors fail to lock down the sensor (from being accessed by the normal world programs) when the processor switched back from the secure world. Without the proper lock-down, the attacker from normal world can directly read the fingerprint sensor (shown in Figure 7). Note that attackers can do this stealthily in the background and they can keep reading the fingerprints on every touch of the victim's fingers. This also indicates that attackers with remote code execution exploits (e.g. [11]) can remotely harvest everyone's fingerprints in a large scale, without being noticed.

Given that the fingerprint sensor is exposed to the normal world, the only protection is the access permission enforced by the normal world kernel. Unfortunately, on some phones, the sensor is only guarded by "system" privilege instead of "root". This decreases the attack difficulty since there are some easier-to-exploit vulnerabilities (e.g. CVE-2015-1474) to gain the "system" privilege compared to the "root" privilege.

After obtaining enough privilege to read/write the sensor, the remaining matter is how to configure the sensor into the state to actively read fingerprints. Usually the fingerprint sensor's IO operations are open-sourced and may include the following:

```
IOCTL_POWER_ON
IOCTL_POWER_OFF
IOCTL_DEVICE_RESET
IOCTL_SET_CLK
IOCTL_CHECK_DRDY
IOCTL_SET_DRDY_INT
IOCTL_REGISTER_DRDY_SIGNAL
IOCTL_SET_USER_DATA
IOCTL_GET_USER_DATA
IOCTL_DEVICE_SUSPEND
IOCTL_STREAM_READ_START
IOCTL_STREAM_READ_STOP
IOCTL_RW_SPI_MESSAGE
IOCTL_GET_FREQ_TABLE
IOCTL_DISABLE_SPI_CLOCK
IOCTL_SET_SPI_CONFIGURATION
IOCTL_RESET_SPI_CONFIGURATION
IOCTL_GET_SENSOR_ORIENT
```

Note that for certain implementations, although the sensor got exposed to the normal world, the normal world kernel has stripped out certain IOCTL handlers. Malware can utilize some kernel exploit to dynamically enable such functionalities (e.g. through code injection or ROP). Malware with root privilege can also flash a customized kernel with such IOCTL handlers enabled into the Android's boot partition.

Malware can use `IOCTL_REGISTER_DRDY_SIGNAL` to register as the event signal listener of the sensor device. Then it can sequentially invoke `IOCTL_POWER_ON`, `IOCTL_SET_CLK`, `IOCTL_DEVICE_RESET`, `IOCTL_SET_DRDY_INT`, `IOCTL_CHECK_DRDY`, `IOCTL_STREAM_READ_START`, etc. to initialize the device to the actively reading fingerprints state.

Once the sensor is in the active reading state, malware can keep stealing users' fingerprints silently in the background. For some devices, the fingerprint sensor is integrated onto the home button, which means that the attacker can steal users' fingerprints on every touch of the home button.

4. Pre-embedded Fingerprint Backdoor

Moreover, the attackers can stealthily embed prefabricated fingerprints in the devices as an authorization backdoor, before providing a new device to the victim. The root cause of this vulnerability is that the UI displaying the number of registered fingerprints is a separate

component (in the normal world, without TrustZone's protection) from the actual fingerprint auth framework in the secure world. Attackers can deceive the user to believe that there are only *N* fingerprint registered on the device but there are actually more than *N*. Such extra pre-embedded fingerprint can be used to bypass the auth framework like a backdoor.

It is usually the Settings app that displays the registered fingerprint number to the users, so the attacker needs to modify the Settings app. For example, on many devices, one can modify the `enrolledFingerprintNum` method of the class `com/android/settings/fingerprint/FingerprintSettings` in `SecSettings.apk`, by changing the return value of `getEnrolledFingers` to be *n-m*, where *n* is the actual registered fingerprint number and *m* is the number of fingerprints pre-embedded by the attacker.

Note that Settings is a system app, the attacker needs to signing the modified Settings app with the same private key that signs the other system apps, which is the phone vendor's private key and difficult for attackers to obtain. The attacker could also extract the ROM and resigning all the system apps (including the replaced Settings app) using the attacker's own private key.

If the attacker has root privilege, another alternative is to directly disable the system signature checking. Most of Android devices enforce the system signature checking based on the `compareSignatures` method in the class `com/android/server/pm/PackageManagerService` implemented in `/system/framework/services.jar`. It will return zero if signature match, and non-zero otherwise. Therefore, one can modify this method to always return zero, so that the system signature checking will always success.

IV. Discussion

1. Suggestions to mobile users and vendors

To avoid being attacked by malware or being exploited for remote code execution, we suggest normal users to choose mobile device vendors with timely patching/upgrading to the latest version (e.g. Android Lollipop), and always keep your device up to date. Also, it is always a good practice to install popular apps from reliable sources. Enterprise/government users should seek for professional services to get protections against advanced targeted attacks.

Mobile device vendors should improve the security design of the fingerprint auth framework with improved recognition algorithm against fake fingerprint attacks, and better protection of both fingerprint data and the scanning sensor. Moreover, vendors should figure out how to differentiate authorization with authentication and provide context proof. The existing fingerprint auth standard should be further improved to provide more detailed and secured guidelines for

developers to follow. Finally, given a security standard, vendors still need professional security vetting/audits to enforce secure implementations.

2. Suggestions to the overall fingerprint auth ecosystem

Actually all the four vulnerabilities/attacks described in this paper are commonly applicable to all the fingerprint based authentication/authorization platforms. For example, many high-end laptops equip fingerprint scanners to authenticate and authorize user login. Since the fingerprint driver is a kernel module, it is only as secure as the kernel. Attackers with kernel exploit can steal fingerprint data or collect fingerprint from the sensor in the background.

For external fingerprint scanners used for identity recognition (e.g. in the custom house, immigration office, and the DMV), door access control, or money transaction in banks, the situation is similar. So we suggest that the fingerprint auth framework for all platforms should also be improved to better protect fingerprint data and sensor (and provide defense of any other attacks described in this paper if applicable).

V. Conclusion

In this talk, we revealed some severe issues with the current Android fingerprint frameworks that have long been neglected by vendors and users. We provided in-depth security analysis of the popular mobile fingerprint authentication/authorization frameworks, and discussed the security problems of existing designs, including (1) the confused authorization attack that enables malware to bypass pay authorizations protected by fingerprints, (2) insecure fingerprint data storage, (3) fingerprint sensor exposed to the untrusted world, and (4) pre-embedded fingerprint backdoor. We also provided suggestions for vendors and users to better secure the fingerprints.

References

[1] http://www.marketresearch.com/land/product.asp?productid=8918844&progid=87404
[2] https://srlabs.de/spoofing-fingerprints/
[3] http://www.ccc.de/en/updates/2014/ursel
[4] http://forum.xda-developers.com/apps/framaroot/root-framaroot-one-click-apk-to-root-t2130276
[5] https://towelroot.com
[6] http://forum.xda-developers.com/galaxy-s6/general/root-pingpongroot-s6-root-tool-t3103016
[7] http://www.arm.com/products/processors/technologies/trustzone/index.php
[8] https://www.blackhat.com/us-14/briefings.html#reflections-on-trusting-trustzone
[9] https://www.blackhat.com/us-15/briefings.html#attacking-your-trusted-core-exploiting-trustzone-on-android
[10] http://infocenter.arm.com/help/topic/com.arm.doc.dto0015a/DTO0015_primecell_infrastructure_amba3_tzpc_bp147_to.pdf

[11] https://www.blackhat.com/docs/us-14/materials/us-14-Wei-Sidewinder-Targeted-Attack-Against-Android-In-The-Golden-Age-Of-Ad-Libs.pdf
[12] https://fidoalliance.org/wp-content/uploads/html/fido-uaf-overview-v1.0-ps-20141208.html