Freie Universität Berlin

Bachelorarbeit am Institut für Informatik der Freien Universität Berlin

AG Informationssicherheit

Deviations in Key Certificate Chains in Android

Matthias Grabner
Matrikelnummer: 5575714
m.grabner@fu-berlin.de

Betreuer/in: Lawrence Dean
Eingereicht bei: Prof. Dr. Marian Margraf
Zweitgutachter/in: Prof. Dr.-Ing. Jochen Schiller

Berlin, 17.07.2025

Abstract

Android's key attestation has given developers a valuable tool to verify the authenticity of a devices cryptographic keys and their attributes. The possibility to also add ID attestation fields to a keys attributes, providing trustworthy claims about the hardware that created the key, enables developers to have certainty about the device they are communicating with. This enables use-cases such as zero-touch remote configuration. However, the implementation of ID attestation is optional, with no data being available on it's prevalence. Additionally, certain key attributes may not be correct. This thesis aims to provide data on the prevalence of implemented fields in the keys across a wide variety of Android devices, discuss any potentially suspicious value and discover mismatches between the device properties and key attestation fields.

Eidesstattliche Erklärung

Ich versichere hiermit an Eides Statt, dass diese Arbeit von niemand anderem als meiner Person verfasst worden ist. Alle verwendeten Hilfsmittel wie Berichte, Bücher, Internetseiten oder ähnliches sind im Literaturverzeichnis angegeben, Zitate aus fremden Arbeiten sind als solche kenntlich gemacht. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungskommission vorgelegt und auch nicht veröffentlicht.

17.07.2025

Matthias Grabner

Contents

1 Introduction

Today, mobile phones are a vital part of our digital identity and are used for many security-sensitive areas. With Android holding a market share of over 70% [15], billions of users worldwide rely on the security of their Android devices for sensitive tasks such as banking, messaging, and recently also signing legally binding contracts [19]. Therefore, developers need to create reliable and cryptographically signed apps, which ensure that sensitive tasks can be executed securely when running on hardware with verified integrity. To check device and key integrity, Android utilizes its key attestation system, which enables developers to verify the properties of a key, such as whether it is stored in a device's hardware-backed keystore. Other relevant properties that can be verified with the key attestation system include the OS version or patch level [10]. The certificate chains used in the key attestation system form the foundation for verifying the integrity of the device [22].

Android has strict guidelines regarding the implementation of its key attestation system. However, manufacturers may produce deviating values as part of the attestation certificate extension data, leading to a mismatch between the build-properties of the device and the properties suggested by the attestation certificate [6]. This mismatch could indicate that the device has been modified previously, contains bugs in vendor-specific implementations, or is counterfeit. In a counterfeiting or rooting scenario, the Android Debug Bridge (ADB) data could be modified, but the key-attestation would still rely on the trusted execution environment [1].

This thesis aims to discover any potential deviations in the key attestation attributes of Android devices and to analyze their security implications, should deviations be found. If no mismatch is present, certain unset tags or old patch levels could still warrant further investigation. For example, data on the frequency of ID attestation adoption or various other fields can help developers make an informed decision about whether to include checks in their apps for those properties. By analyzing a multitude of Android devices, this thesis will shed light on how commonly various fields in the key attestation extension are implemented and how common deviations between build-properties and key extension data of Android devices are. Our practical research on a multitude of Android devices from various manufacturers, combined with our review of existing literature on the topic, will help answer this question. The specific research questions that will be answered within the thesis are: How does the Android key attestation feature work, and what security guarantees can be made by it? Additionally, how frequently are various key and ID attestation fields implemented, and are there any deviations from the build-properties present in the key attestation extension data? If not, are there other values that warrant further investigation?

To investigate these questions, we adopt a combination of empirical analysis and practical research. The practical portion of our approach involves developing a simple app that generates a key using Android's keystore system and retrieves the relevant attestation certificate. It will send the obtained data to our server to extract the key attestation extension data scheme, containing entries such as `attestationIdDevice` [10]. This data will then be compared to the build-properties of the Android device

obtained via `adb shell getprop`. Furthermore, other entries of the attestation certificate will be checked for their coherence (e.g., patch levels and the phone release date). For the theoretical part, we will investigate the Android Open Source Project's (AOSP) documentation and portions of the code of current Android releases from the AOSP Git repository [6]. Additionally, we will search for relevant publications connected to the topic to provide a theoretical base for our research. Regardless of whether deviations or any suspicious values are found, the potential implications of a mismatch between the data contained in the attestation certificates and within the build-properties will be discussed.

The thesis is structured as follows: Chapter 2 provides background information that lays the theoretical groundwork to understand our practical research and serves as an introduction to Android's key attestation system. Chapter 3 provides an overview of the experimental setup, including an introduction to the software solutions developed for this thesis. The results obtained are discussed in Chapter 4. We will then evaluate our findings in Chapter 5 and use Chapter 6 to suggest some future work, concluding in Chapter 7.

2 Fundamentals

In the following section, various fundamental concepts will be discussed, serving as the theoretical foundation for this thesis. These fundamental concepts include certificate chains, key attestation, what attributes can be verified via key attestation, and build-properties. Furthermore, a brief literature review will be conducted, highlighting existing work on the topic of key attestation, its security, and potential faulty implementations by vendors. Given the scarcity of academic papers on this topic, the AOSP documentation and other web sources will be utilized extensively, with scientific papers used where available.

2.1 Key Concepts & Definitions

2.1.1 Certificate Chains

Certificate chains are ordered lists of TLS/SSL certificates issued by potentially different certificate authorities (so-called "CA's") [9]. The intermediate certificate signs the SSL certificate at the beginning of the chain. Thus, to verify the SSL certificate at the beginning of the chain, the end-device obtains the intermediate certificate, which is, however, signed by either another intermediate certificate next in the chain or the root certificate (also referred to as the "root of trust"). If multiple intermediate certificates exist, this repeats until the root of trust is reached. This can be imagined as less powerful certificates always being vouched for and signed by more powerful ones. After the root is reached, the path is traversed backward, and signatures are verified until the SSL certificate is also verified [9].

On Android devices, the "root of trust" is embedded into secure hardware and is formed by a unique key pair, consisting of a private key `PrivT` and a public key `PubT`. This is signed with the root certificate from Google, ensuring that it can be

verified using Google's public root certificate [1]. An illustration of this concept, along with other relevant keys and data stored, can be found in Figure 1. In some cases, adversaries may manage to circumvent the Trusted Execution Environment (TEE) and extract the private key of the root of trust. In such a scenario, the attackers could then attest any desired property using a valid signature. For this purpose, if Google discovers a compromised key, it is revoked by publishing it on their site and enabling verifying parties to recognize it as invalid. Certificate chains are closely tied to key and ID attestation and play a crucial role in Android's security framework. Using the process of "traversing" the certificates, keys, and IDs can be attested and verified [10].

2.1.2 Key & ID Attestation

The security of cryptographic keys in practice also depends on whether they are stored in a secure, hardware-backed keystore or not. Key attestation provides this information and was introduced in Android 7.0 as part of Keymaster 2. With Android 8.0, Keymaster 3 was introduced, which also implements ID attestation [10] and now provides more attributes that can be (theoretically) verified, and where deviations in comparison to the build-properties could be found. This is achieved with the help of certificate chains, which establish a chain of verification from within the Trustzone to the app [1].

2.1.3 Attestation Certificate Extension Data

When a key is verified to be stored securely, the attestation certificate extension data is stored within the attestation certificate. This can include the following properties: `attestationIdBrand`, `attestationIdDevice`, `attestationIdModel`, `osVersion`, `vendorPatchLevel` and the like [10]. How the properties contained in the certificate are fetched depends on the value and Keymaster/KeyMint version. In Android's Keymaster version 4, for example, `osPatchLevel` is read from `ro.build.version.security_patch` at boot time and sent to the secure environment, where it cannot be modified until the next boot [4]. The attestation certificate extension data containing the outlined properties is guaranteed to be stored in the leaf certificate, but can also be present further up the certificate chain [24].

2.1.4 Build-Properties

Build-properties contain information about the Android device, compiled at the time of the firmware build. Several deviations from expected values have been shown in the past, including the wrong build type [20]. However, discovering inconsistencies in the build-properties is not the goal of this thesis. Build-properties will rather be used for comparison against the attestation certificate extension data and as a supporting measure for potential investigations into the causes of suspicious entries.

2.2 Related Work

The theoretical (and practical) related work available is limited, which leads to the application-oriented nature of this thesis. However, four sources of varying scopes are

closely associated with the problem being investigated and will be briefly summarized in this section.

Mayrhofer et al. introduce the Android Platform Security Model. They provide a comprehensive model for the security principles that Android relies on, as well as some more specific discussions [14]. The layers of defense of Android devices are discussed, with patching being introduced as one of them. With key attestation potentially offering insight into version and patch data, this is an interesting observation. Key and ID attestation are discussed in the context of the whole Android system, and their function is explained. Additionally, related concepts such as Android Verified Boot (AVB) and a general bootloading process are discussed.

Aldoseri et al. describe and model various remote attestation protocols used under Android, such as the meanwhile deprecated SafetyNet API by Google Play, Samsung Knox attestation, and Android key attestation [1]. The paper offers valuable insight into the inner workings of these systems, which is difficult to obtain via publicly available documentation. Additionally, the security properties of the analyzed attestation protocols are suggested, and the authors conduct a case study using Android's key attestation feature. An adapted version of their protocol flow model for the Android key attestation, tailored to our testing, is shown in Figure 1.

We were able to find one already existing application used for testing a device's key attestation and listing various properties associated with it [25]. Among other things, this app displays the certificate chain, checks if the bootloader is locked, and whether the tested device can use an attestation root certificate by Google. If this were not the case for a device, it wouldn't be Google Mobile Services (GMS) compatible [25], or its private key of the Trustzone (`PrivT`) would be considered compromised or revoked for some other reason by Google [24]. However, this application lacks several features that we consider to be vital and also has multiple drawbacks for our specific use case. First, the application is very complex, spans many thousands of lines of code, and has a much broader focus than exporting the attestation certificate extension data. Since it only runs on the phone locally and offers no option to export data, we would need to modify the app. However, this is made extremely difficult due to its large codebase. It is also not immediately clear while using the app whether ID attestation is supported by the device being tested.

Pöll and Roland offer excellent insight into the build process of Android firmware and potential issues present in the analyzed firmware [20]. Especially relevant for this paper, two issues in the build-properties are described. Both the inconsistent build types and the additional entries in property files are described as "unaccountable differences" to the design intended by the AOSP. The authors, therefore, criticize the vendors for failing to meet the specifications laid out by the AOSP. As these issues are present in the build-properties, they hint at incorrectly set attributes and, therefore, potential deviations between the attestation certificate data and build-properties. However, for a deviation to occur and for values to be comparable, a build-property and an entry in the attestation certificate must represent the same underlying data. Whether this is the case, and what values we were able to compare, will be discussed in Section 4.3.

2.3 Contribution

We identified no prior research on potential discrepancies between attestation certificate extension data and data in the build-properties. Additionally, we were not able to locate information on the prevalence of the various key attestation data fields. Some informal entries in online computer science forums suggest that there may be an issue with manufacturers not implementing ID attestation, which provides many valuable entries in the attestation certificate extension [5, 11, 21]. However, outside of this, we could not locate any formal or informal discussion on the matter beyond the official AOSP documentation. Therefore, this thesis will contribute to the discussion of the key and ID attestation implemented by vendors and offer empirical, verifiable research into the matter using a multitude of Android devices. It will provide insight into whether data in the attestation certificate extension offered by ID attestation is missing on the majority of devices, as suggested, and if there are other deviations compared to the data from the build-properties. Additionally, a discussion will be held on whether there are other inconsistencies regarding the entries in the attestation certificate.

3 Experimental Setup

My practical research will involve developing a suitable Android application and its server counterpart, which are capable of extracting and storing attestation certificate extension data. To achieve a successful extraction of information, several key steps have to be followed, which are outlined briefly below and illustrated in Figure 1. A more practical explanation of our approach, along with a justification for our decisions regarding the test app and server, can be found throughout the following subsections.

First, the application requests the creation of the attestation data. The application `Test App` consists of the package name (`Pkg`), source code, and resources (`content`), the public certificate of the developer key (`PubDk`), and the application signature (σ). To achieve a successful request, the app interfaces with the OS, which in turn requests the attestation of the key from Keymaster or KeyMint.

Once the request is registered by Keymaster/KeyMint, a key pair (`PrivK`) has to be generated, for which the hardware-backed keystore also creates a certificate (`CertT`). To do this, Keymaster/KeyMint has a unique, private certificate key (`PrivT`) and a certificate (`CertRoot(PubT)`). The certificate `CertRoot(PubT)` has Google's root certificate as a root of trust [1]. Therefore, the key created within the Trustzone can be verified with Google's root certificate.

Once requested and created, the key certificate pair and the certificate chain (`CertsChain`), including the attestation certificate, are sent from the Trustzone back to the test app. `CertT` contains the public key (`PubK`) of the app, device information (`d = device status`), and information about the app that requested its creation (`a = <Pkg,` σ`>`).

The test app now extracts the newly created attestation certificate from the certificate chain (`CertChain[0]`). Once it is successfully extracted, the attestation certificate is then sent to an external server. There, the certificate is saved, and the key extension

data is parsed. A brief snippet of pseudocode illustrating the process relevant to understanding the extraction of key extension data will be included. The repository containing the whole code, as well as a finished .apk build of the attestation app, can be found in Appendix B.

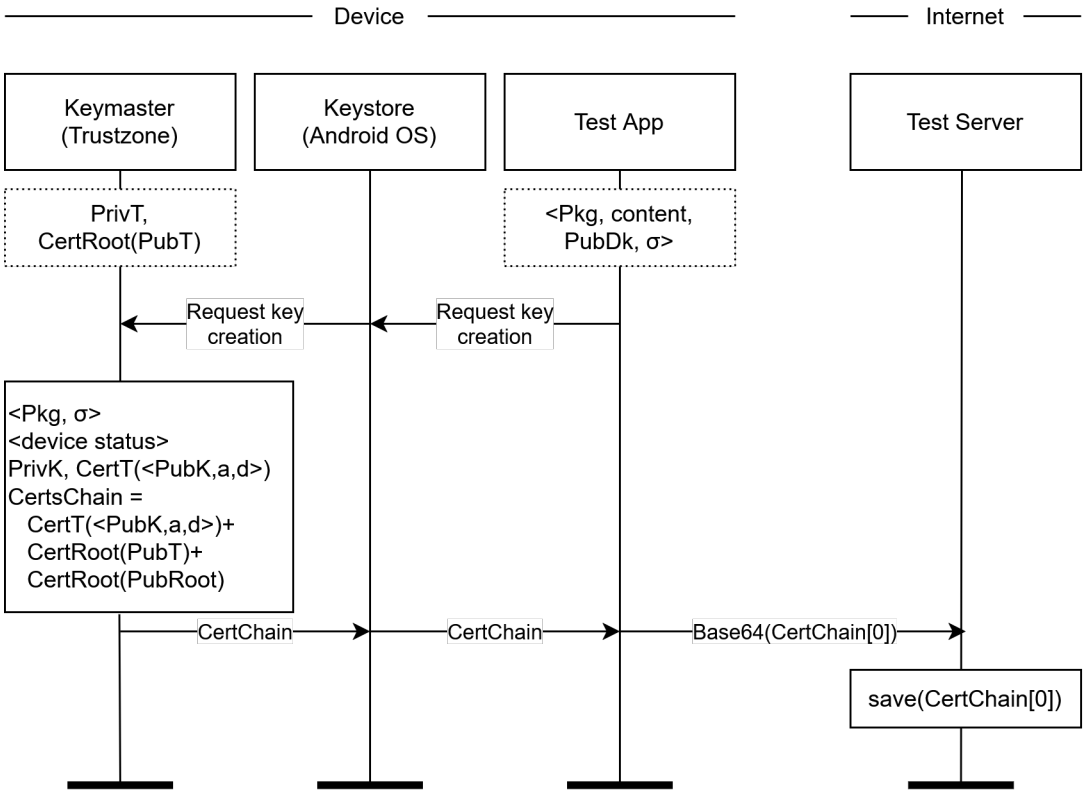

Figure 1: Flow of our test app. Modified and adapted from [1]

3.1 App: Generating a key pair

The app first generates a key pair, the certificate of which can then be examined in the following steps. To generate the key, we utilize the Android Keystore system, which allows apps to store their own keys, in contrast to Android Keychain's system-wide credentials [2]. To research key extension data, the cryptographic algorithm used for the key is irrelevant. Similarly, the attestation challenge used also has no impact on the results. The set purpose of the key has more importance, due to some properties in the attestation certificate only being included for a specific purpose [10]. Additionally, some purposes were only implemented for later versions of Android. As we are interested in researching key attestation, we used `PURPOSE_ATTEST` for our app.

3.2 App: Extracting the certificate

Which certificate is to be extracted for attestation purposes depends on the use case of the attestation. To verify hardware-backed keys using key attestation, only the first occurrence of the key extension data in the certificate chain should be used. This is

due to key extensions further along the chain not having been created by the secure hardware and being potentially vulnerable to tampering [24]. However, as we are using what we assume to be uncompromised smartphones used solely for security research, we adopted the simpler approach of using the key extension data from the leaf certificate. Therefore, we also don't send the entire certificate chain to our server, but limit the data to the first entry in the chain: the leaf certificate.

To summarize, the functionality of the app can be described as follows:

```
PROCEDURE GenerateKeyPair()
    DEFINE key parameters (alias, purpose, algorithm, digest,
       challenge)
    LOAD Android Keystore
    INITIALIZE key generator with parameters
    GENERATE key pair
END PROCEDURE

PROCEDURE ExtractLeafCertificate()
    GET certificate chain from Keystore using alias
    SELECT first certificate as attestation certificate
    RETURN attestation certificate
END PROCEDURE
```

The data could be extracted either locally and then sent to the server, or sent first and extracted later. To achieve better traceability of the data parsing and greater flexibility, we decided to retain a copy of the certificate, as well as the parsed data. Therefore, the certificate was stored and parsed remotely on our server.

3.3 Server: Extracting key extension data

Due to the testing app being installed on multiple devices, a centralized approach was used, where a server stores the data of all the tested devices. To transmit the data, we used standard Base64 encoding of the certificate without headers (DER format), and also included the model of the smartphone in the transmission. This allows the server to store the certificate under a uniquely identifiable filename.

However, extracting key extension data is not straightforward and requires parsing the ASN.1 structure of the certificate. Again, current options for parsing the attestation certificate consist of thousands of lines of code [8, 12], and the most promising option, designed as an example, is not intended for use as a program. Thus, to facilitate the extraction of the key extension data, a small Java program was written. We used insights from a web-based parser [12] to identify the structure of a sample certificate created by our app to parse it correctly. This program extracts the fields we consider to be relevant for our analysis and already includes the conversion from the tag-numbers to the name of the fields specified by AOSP [10].

3.4 Additional notes

The `getprop` Data needed for comparison with the attestation fields is far easier to extract and can be simply obtained via the command `adb shell getprop` after connect-

ing to the device via `adb connect`. This data was obtained using a simple automated shell script created by us.

It should also be noted that the application only serves testing purposes and therefore does not meet the security standard of Android applications for production. One example would be the use of HTTP, which is not allowed on Android versions above Android 9, without specifying the connecting servers in `network_security_config.xml` [16].

4 Results

In this section, the findings obtained from the experimental analysis of the certificate data across the 47 tested devices are presented. These devices were mostly released between 2020 and 2023. Only four of the tested phones were released between 2017 and 2019, and none were older than that. The phones tested spanned the low to high range and were running Android 10 to 13, with the majority using Android 11 to 13. We tested phones from the following brands: Fairphone, Google, Motorola, Nokia, OnePlus, Oppo, Realme, Samsung, TECNO, Sony, Xiaomi, and ZTE. The number of phones tested was not equally distributed by brand. Table 4 in Appendix A lists the brand, announcement date, and whether each phone has ID attestation for easy comparison.

All data was collected using the custom test application described in the previous chapter. The primary goal of the analysis was to identify deviations from the expected behavior. This includes, but is not limited to, the comparison against the device-property data of the respective devices. Another research question was how frequently, if at all, various attestation extension fields were implemented. Potential issues caused by deviations or a missing implementation of tags will be outlined in Chapter 5.

First, we will present our findings regarding the structure of the attestation extension, before moving on to the frequencies with which fields in the attestation certificates were included. The results in Table 1 and in this subsection are not limited to the fields compared against the build-properties, but include several other fields we consider to be interesting for reasons outlined below. Lastly, we will present which attestation flag we compared with which build-property and whether we were able to find any mismatches.

4.1 Structure of the attestation extension

Our server only receives a portion of the certificate chain, whose creation was requested by our app. This portion is called the attestation certificate, and is always the first entry of the certificate chain. This attestation certificate is specified using the ASN.1 structure, used to specify X.509 certificates in general. However, the specification in the Android documentation does not display all relevant data types and only shows part of the certificate's structure. The structure, omitting some entries at the same level and focused on showcasing the position of the data relevant to our re-

search, is shown in Figure 2. It was found using a mixture of available documentation [10], a web-parsing tool for general ASN.1 certificates [12], and trial and error with our own parsing tool.

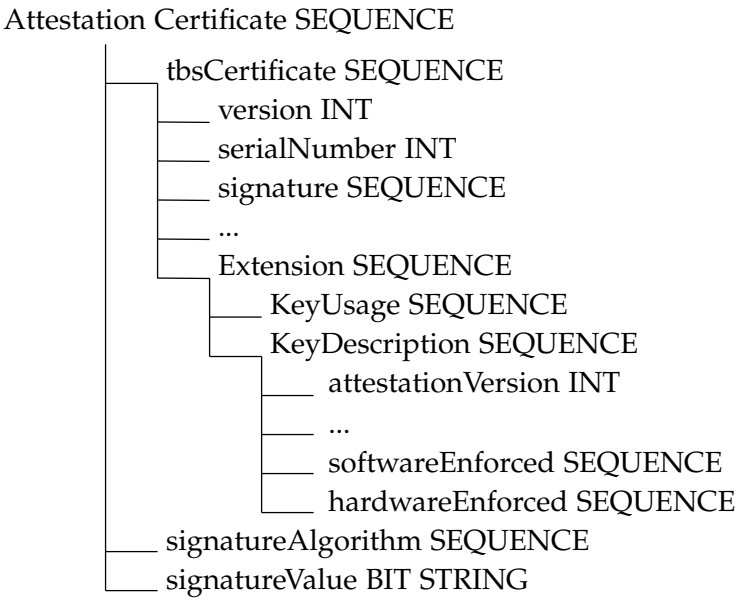

Figure 2: Structure of the attestation certificate

The KeyDescription extension (OID 1.3.6.1.4.1.11129.2.1.17) [10] contains the data especially relevant to our research. The ID attestation and all other attributes, except for information about the security of the attestation itself, can be found in `softwareEnforced` and `hardwareEnforced`, respectively. To decode these fields, Android provides developers with ASN.1 information, with which specialized parsers decoding the included integer tags to attributes can be created [10].

4.2 Frequency of Fields in the Certificates

Table 1 showcases the frequency with which vendors implemented some of the most common attestation fields and categories. Entries in "Not Applicable" include phones that send their certificate in a version where the field is not yet or no longer implemented. The certificate version translates to the Keymaster or KeyMint versions used to create the attestation certificate.

While the AOSP documentation lists 40 fields that can theoretically be set with the newest certificate version 400 in `softwareEnforced` and `hardwareEnforced`, we discovered that only a handful of them are used in practice. However, eight of the 40 fields are listed as NULL fields, meaning that their absence also provides potentially useful information. The majority of our tested certificate extensions only include about 10-15 entries enforced by hardware and software combined. While all 40 fields are listed as optional, setting them can provide developers relying on the key attestation feature with valuable information and security functionality explained in Chapter 5. We consider the following fields, which were at least present on some tested devices, to be especially relevant to our research:

	Implemented	Not Implemented	Not Applicable
ID Attestation	19	28	0
creationDateTime	46	1	0
osVersion	47	0	0
osPatchLevel	47	0	0
vendorPatchLevel	40	6	1
bootPatchLevel	40	6	1

Table 1: Frequency of implementation for fields in attestation certificate

- **attestationVersion:** This field provides an overview of the most prevalent versions in Android devices, released primarily between 2020 and 2023. The available versions include: 1, 2, 3, 4 (for use with Keymaster) and 100, 200, 300, 400 (for use with KeyMint) [10]. Certificates with a more advanced `attestationVersion` include additional entries that can be used to verify additional properties of the phone. For example, ID attestation was added in version 2 (corresponding to Keymaster version 3.0), and version 300 (corresponding to KeyMint version 3.0) introduced ID attestation of the second IMEI. Some tags present in earlier versions were also removed [10]. The attestation version is also useful for parsing the certificate, as the available tags and tag names can vary between versions. For an easy overview, a histogram is included in Figure 3. Additionally, we consider it to be interesting as a stand-alone value, as it showcases the speed of adoption (or lack thereof) of Android's key management utilities.

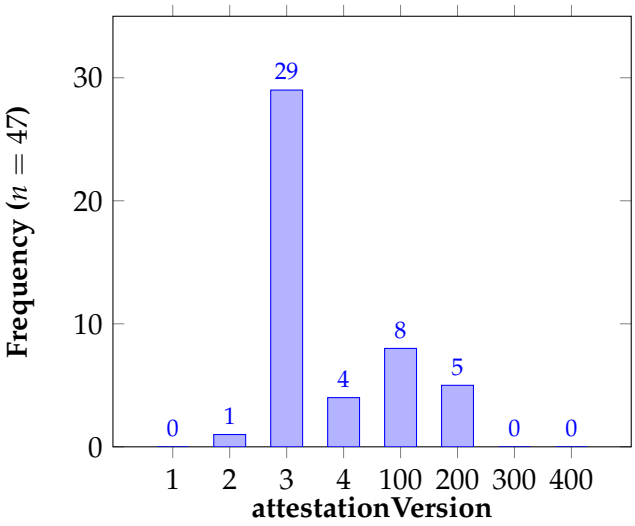

Figure 3: Histogram of `attestationVersion` across all tested devices.

- **creationDateTime (Tag 701):** The `creationDateTime` specifies the time at which the certificate was created. In case the date for the start of the validity is not set outside the attestation (in the validity field under `tbsCertificate`), the validity is set to `creationDateTime` [10]. While the start of the validity can include the creation date, it doesn't have to. Except for one device, the tag

`creationDateTime` was present on all devices. The implications of the absence of this tag will be discussed in Section 5.3.

- **osVersion (Tag 705):** As both the attestation certificate and the `getprop` data provide information about the installed version of the operating system, both entries should match and were compared against each other. Our results show that no device suffered from mismatches between the fields.
- **bootPatchLevel (Tag 719)**: BootPatchLevel is an entry available only for attestation version 3 and later. As shown in Table 1, it is set relatively commonly, especially if the attestation version is high enough. As the name suggests, it displays the kernel image security patch level. Therefore, it provides important information on the resistance against new and existing attacks. Still, as will be discussed in Section 5.2, even with the patch level, only some deductions about the devices' security can be made.

4.3 Comparison of attestation values and device-properties

To address our original research question of whether there are deviations between key attestation values and build-properties, we conducted a comparison between the two. Which attestation flags we compared to which device property, and the number of comparisons for our sample size of 47 phones, is shown in Table 2.

Attestation Flag	Build-Property	Compared Devices
osVersion	ro.build.version.release	47
osPatchLevel	ro.build.version.security_patch	47
vendorPatchLevel	ro.vendor.build.security_patch	39
attestationIdDevice	ro.product.device	19
attestationIdModel	ro.product.model	19
attestationIdManufacturer	ro.product.manufacturer	19
attestationIdBrand	ro.product.brand	19
attestationIdProduct	ro.product.product	19

Table 2: Comparison of attestation flags and build-properties. All set entries match.

Our original research question of whether there are deviations between key attestation values and build-properties can be answered with "no" for our sample of phones. For the phones with implemented ID attestation, we were able to verify the device, model, manufacturer, brand, and the product name. On the majority of phones, we were also able to verify their patch levels and OS versions.

5 Evaluation

We discovered several interesting details by testing the devices, which we will evaluate in this section. Our findings can be roughly divided into the following categories: lackluster use of the ID attestation system, old bootPatchLevel entries, an unset tag on the Poco M5, and matching key attestation and build-property values. We will try to

identify possible causes and effects that these results may have. We will also include a short evaluation of potential threats to the validity of our findings and methodological weaknesses.

5.1 Lackluster Implementation of ID Attestation

Even though ID attestation was available only one Android version later than key attestation, it remains unused in the majority of our tested devices. When enabled, it can help to authenticate the hardware to the OS, the usefulness of which was outlined in Chapter 1.

Vendors provide no justification for implementing or not implementing ID attestation. However, a likely explanation would be that ID attestation could result in modifications to the manufacturing and servicing process. For ID attestation to work, the read-only storage within the Trusted Execution Environment (TEE) has to be populated with hashes of relevant device information during the manufacture of the device [10]. The knowledge of the data being immutable once written allows the device to "prove" its correctness due to the guarantees made by the hardware.

However, it could be argued that key attestation also requires the population of small parts of read-only storage to store the private key for the Keymaster/KeyMint implementation (`PrivT`) as well as a certificate (`CertRoot(PubT)`). Still, key attestation is mandatory (since Android 8.0), and the hardware requirements for the storage are likely also different, considering that (`PrivT`) should not be extractable by any means [2].

Modifying the manufacturing process is likely expensive, and as ID attestation is optional, vendors might opt to keep older solutions. Additionally, the AOSP dictates that Return Merchandise Authorization (RMA) facilities should be able to restore the functionality of the device's ID attestation if it was previously disabled using the built-in `destroyAttestationIds()` killswitch [10]. Therefore, the processes at RMA facilities would also need to be modified.

However, some vendors seem to slowly modify their manufacturing process and introduce ID attestation to their devices. Samsung, for example, seems to have made a switch to releasing phones that support ID Attestation in 2023. Both the Galaxy A and the Galaxy S series seem to ship with this security feature embedded now. Sony, too, appears to have switched to embedding ID attestation in their devices in 2023. When testing phones from Xiaomi, however, we could not identify a single phone with functioning ID Attestation. It should be noted that the Android Compatibility Definition Document (CDD), which details the requirements vendors should and sometimes must follow, did not change to require ID attestation as of Android 16 [2].

See Table 3 for the full results regarding the frequency with which ID attestation was implemented on the devices we tested.

5.2 BootPatchLevel

While most devices display expected, matching values for the fields `osPatchLevel`, `vendorPatchLevel`, and `bootPatchLevel`, the `bootPatchLevel` field of three phones points to an older, identical date: 06.06.2019. These devices are the TECNO Spark 10 Pro, Motorola Edge 40, and the Moto G23. In all instances, the value was hardware verified, as required by Keymaster 4.0 [4]. With all phones being released in 2023 [23, 17, 18], four years after the suggested patch level, this value warrants further investigation. Some phones also did not specify a `bootPatchLevel`, even though their attestation version did support the field.

The implementation of Keymaster 4.0 on Android 13, which was used in all affected phones, specifies that this value must be provided to the secure environment by the bootloader [4]. The phones tested by us only contained four phones with Keymaster version 4.0, three of which contained the old bootPatchLevel and one which didn't include information about the bootPatchLevel in its attestation certificate. Whether there is a link or not would have to be established with a larger sample of phones using Keymaster 4.0, as outlined in 5.5.

One concerning explanation would be that the BootPatchLevel displayed is indeed accurate and that a patch level already four years old at the time of implementation was still used. This would result in the device being vulnerable to a wide variety of kernel-related exploits and publicly disclosed weaknesses [13]. While the security of low versions of the kernel cannot be solely determined by checking its version, a correlation between newer kernel versions and more secure devices appears to hold [13]. This is due to more kernel defense features being available in newer kernel versions. Still, only a few kernels use most or all of the defense features available, according to Maar et al. [13].

All three affected devices run on kernel version 4.19, according to their build-property `ro.kernel.version`. This version was released in 2018 [13]. Experimental data from 2024 suggests that devices running kernel version 4.19 were susceptible to two to four times more exploitation flows than the newest kernel version across all vendors tested. With all available defenses enabled, the authors suggest the number of successful exploitation flows to be double that of the newest kernel version as of 2024 [13]. It should also be noted that the devices all launched with Android 13 [23, 17, 18], which does not include kernel version 4.19 as a valid launch kernel [3].

As the bootPatchLevel is "only" determined by the bootloader, and its determination is vendor specific, a scenario where a bug in the bootloader leads to an incorrect bootPatchLevel being sent to the secure environment or does not arrive there remains a theoretical possibility. However, as outlined in Chapter 6, the code of bootloaders is generally not publicly available. Also, this would have to affect the bootloaders of two different vendors: Motorola (owned by Lenovo) and TECNO (owned by Transsion).

5.3 Unset Tag on Poco M5

While all key attestation extension tags are optional [10], one tag appears to be implemented on every phone tested by us, except for the Poco M5. This tag is the

`creationDateTime`, which is available in all certificate versions. Curiously, the Poco M3 and M4 have the `creationDateTime` tag set. As the name suggests, this tag specifies the creation time of the attestation certificate [4]. The consequence of this tag being unset, together with the certificate validity date, is the certificate validity being automatically set from 1970 to 2048 [10]. This is the case for the Poco M5, where we have created a certificate valid from January 1, 1970, to January 1, 2048, using our test app. Usually, the `creationDateTime` serves as the start date of the certificate validity, where the validity flag is rarely specified.

Additionally, it appears that the creation of the mandatory `UniqueId` tag is impacted. Android's documentation states that it is a "privacy-sensitive identifier", which can be requested by system apps but which identifies the device "only for a limited period of time" [10]. The `UniqueId` is constructed using the following formula:

$$HMAC\_SHA256(T||C||R, HBK)$$

Where T is the temporal counter value, calculated by `creationDateTime`/2592000000. "C" is the app-specific `attestationApplicationId` and "R" is set to either 0 or 1, depending on whether a factory reset occurred since the last unique ID rotation. The unique ID rotation should occur once a month, with the result of T increasing by one. "HBK" refers to the hardware-bound secret, an unchanging value [10].

We can therefore observe that the documentation suggests a problem in the case of `creationDateTime` being unset. While "C" would still be different for each system app, the remaining variables would remain unchanged. Thus, enabling the identification of the device beyond the intended 30 days, forcing the delivery of an empty `UniqueId` even if requested, or throwing an error if a system app requests the creation of a key using the `UniqueId` flag.

5.4 Matching key attestation values and device properties

All values compared in Section 4.3 matched for the devices we tested. However, for some phones we could only compare a very limited number of attestation flags with their corresponding build-properties. Seven phones only offered `osVersion` and `osPatchLevel`. The majority of phones additionally contained `vendorPatchLevel`, and the verification of ID attestation values was possible on 19 phones.

It should be noted that `vendorPatchLevel` and `ro.vendor.build.security_patch` could not be compared on the Samsung S10 (SM-G973F), due to the device property counterpart being an empty field. This was the only instance of a missing device property hindering a comparison. Except for this case, all device properties relevant to the comparison made were present on all devices.

5.5 Threats to validity

Next, we critically examine how our data collection and analysis may have biased the findings and how generalizable we consider them to be. As these limitations could

be potentially avoided, some of them will also be discussed in relation to the future work outlined in Chapter 6.

One limitation of our data collection is the varying update status of the devices used. Table 1 shows the version of Android on each device, with some devices being several major upgrades behind their newest available software. One example of this is the Fairphone 4, which is currently tested on Android 11 but has the upgrade to Android 13 available [7]. While version upgrades cannot add ID attestation [10] and thus don't influence the frequency with which we found the ID attestation feature to be implemented, changing software attested values likely seems possible in case the responsible component is updated.

Additionally, the comparison of the correct values for `osPatchLevel` and `vendorPatchLevel` change with each security patch applied to the system. Thus, our comparisons for these values only show a snapshot of whether all values were updated correctly. This is also true to a lesser extent for `osVersion`, as a device only receives a few major version upgrades. Still, the comparison of the ID attestation values to the product information in the build-properties should remain valid for the entire lifecycle of the device.

Another limitation is the limited variety in release times for the phones used to conduct the analysis. With the data indicating a trend towards greater adoption of ID attestation by vendors in 2023, the frequency with which ID attestation is implemented on phones released in 2025 may not accurately reflect the findings of this thesis.

Unfortunately, we only had four phones with the certificate version 4 available, and thus, we could not determine whether all phones using this version had an old `bootPatchLevel` entry reliably.

The software we used to extract the certificate data suffers from the flaw of not sending the first instance of the attestation extension data to our server. While we do not consider this to be a problem for our setup, as explained in Chapter 3, this would have to be fixed before using our software in another context where the integrity of the tested devices cannot be guaranteed.

6 Future Work

As the device's bootloaders are proprietary software, we were unable to access them to check how the boot patch level is passed to the secure environment, and if there may be fallbacks to default values in the case of some errors. The actual boot patch level present on the device would require an investigation of Android Verified Boot [2]. If it turns out that the old boot patch levels are not accurate, the issue likely stems from the interaction between the bootloader and the secure environment, as outlined in Chapter 5.

We were also unable to determine the exact effects the unset tag `UniqueId` on the Poco M5 had on other components of the attestation. While we suggested some potential effects by analyzing the code, determining the exact effects could be a part of a future

investigation.

Future work could also focus on analyzing more phones for abnormalities regarding the values provided by key attestation to get a more comprehensive picture. This could also address a limitation of this thesis: the almost exclusive use of phones released from 2020 to 2023. While phones older than 2020 might not have ID attestation implemented, thus reducing the potentially attested entries, newer phones released after 2023 could offer later Keymaster/KeyMint versions and a higher ratio of phones implementing ID attestation. Such an undertaking would now be possible with the AWS device farm, for a comparatively low cost, compared to physically buying all the tested phones [26].

7 Conclusion

The objective of this paper was to investigate how Android implements key attestation and to determine whether deviations exist between key extension data and build-properties across a multitude of Android devices. Additionally, an overview of the frequency with which key extension fields (including ID attestation) were implemented should be provided, and any deviations from expected values should be outlined.

To conduct this research, a testing app capable of obtaining key and ID attestation data was developed, and a centralized server was set up to receive this data. We obtained data from 47 Android phones from various brands, which we analyzed according to our objectives stated above.

In the process, several interesting findings were discovered: ID attestation was implemented in less than half of the tested phones, several relatively new phones showed boot patch levels dating back six years, and the POCO M5 had an unusual tag unset in its attestation extension.

This research shows that developers still cannot rely on ID attestation to verify the identity of devices in the majority of cases. Additionally, patch levels of various components, which can suggest a resistance against new exploits, are only accessible on about 85% of the devices we analyzed. Therefore, this thesis emphasizes the importance of collecting data on the availability of security mechanisms across different devices, as a mechanism's overall effectiveness depends on both its presence and its individual performance.

Positively, no deviations were found between the key attestation fields implemented and the build-properties. However, it should be noted that there was only sparse overlap in the represented data and almost no overlap in some devices. This led to a situation where only some values could be compared. The values compared were: Android security patch date, boot image patch level, OS version, and device information, including model and manufacturer.

However, our sample size was limited to 47 phones, all from a specific time period, and may not be representative of the Android phones used by the general public.

Additionally, we were unable to conduct an investigation into the possible causes and effects of our findings regarding the patch-level and the unset tag. This could be resolved by future work on the area of Android key and ID attestation.

In conclusion, we were not able to find deviations between build-properties and the properties attested via key or ID attestation. Nonetheless, we provided a relatively large dataset on the values present in key extension certificates, discussed interesting deviations from expected values, and presented our research into the inner workings and structures associated with Android's key attestation feature.

References

[1] Abdulla Aldoseri et al. "Symbolic modelling of remote attestation protocols for device and app integrity on Android". In: *Proceedings of the 2023 ACM Asia Conference on Computer and Communications Security*. ASIA CCS '23. New York, NY, USA: Association for Computing Machinery, July 2023, pp. 218–231. ISBN: 979-8-4007-0098-9. DOI: 10.1145/3579856.3582812. URL: https://dl.acm.org/doi/10.1145/3579856.3582812 (visited on 02/19/2025).

[2] *Android 16 Compatibility Definition | Android Open Source Project*. en. URL: https://source.android.com/docs/compatibility/16/android-16-cdd (visited on 07/04/2025).

[3] *Android 8.0 Compatibility Definition | Android Open Source Project*. en. URL: https://source.android.com/docs/compatibility/8.0/android-8.0-cdd (visited on 07/04/2025).

[4] AOSP. *keymaster/4.0/types.hal - platform/hardware/interfaces - Git at Google*. URL: https://android.googlesource.com/platform/hardware/interfaces/+/refs/heads/android13-release/keymaster/4.0/types.hal (visited on 05/27/2025).

[5] Justin Bailey. *Sample code showing how to use Android ID Attestation*. Forum post. Mar. 2018. URL: https://stackoverflow.com/q/49184356 (visited on 03/25/2025).

[6] *core/java/android/os/Build.java - platform/frameworks/base - Git at Google*. URL: https://android.googlesource.com/platform/frameworks/base/+/refs/tags/android-15.0.0_r23/core/java/android/os/Build.java (visited on 03/10/2025).

[7] *Fairphone's Operating System*. en-US. URL: https://support.fairphone.com/hc/en-us/articles/9979180437393-Fairphone-s-Operating-System (visited on 05/28/2025).

[8] *google/android-key-attestation*. original-date: 2016-12-20T02:06:51Z. May 2025. URL: https://github.com/google/android-key-attestation (visited on 05/15/2025).

[9] *How Certificate Chains Work*. en-US. URL: https://knowledge.digicert.com/solution/how-certificate-chains-work (visited on 03/17/2025).

[10] *Key and ID attestation*. en. URL: https://source.android.com/docs/security/features/keystore/attestation (visited on 02/19/2025).

[11] kode48. *Did someone managed to use the Android ID attestation*. Forum post. Dec. 2021. URL: https://stackoverflow.com/q/70392232 (visited on 03/25/2025).

[12] Lapo Luchini. *ASN.1 JavaScript decoder*. URL: https://lapo.it/asn1js/ (visited on 04/07/2025).

[13] Lukas Maar et al. "{Defects-in-Depth}: Analyzing the Integration of Effective Defenses against {One-Day} Exploits in Android Kernels". en. In: 2024, pp. 4517–4534. ISBN: 978-1-939133-44-1. URL: https://www.usenix.org/conference/usenixsecurity24/presentation/maar-defects (visited on 05/15/2025).

[14] René Mayrhofer et al. "The Android Platform Security Model". In: *ACM Trans. Priv. Secur.* 24.3 (Apr. 2021), 19:1–19:35. ISSN: 2471-2566. DOI: 10.1145/3448609. URL: https://dl.acm.org/doi/10.1145/3448609 (visited on 02/19/2025).

[15] *Mobile OS market share worldwide 2009-2024*. en. URL: https://www.statista.com/statistics/272698/global-market-share-held-by-mobile-operating-systems-since-2009/ (visited on 03/07/2025).

[16] James Montemagno. *Managing HTTP & Cleartext Traffic on Android with Network Security Configuration*. en-US. May 2019. URL: https://devblogs.microsoft.com/xamarin/cleartext-http-android-network-security/ (visited on 05/15/2025).

[17] *Motorola Edge 40 - Full phone specifications*. URL: https://www.gsmarena.com/motorola_edge_40-12204.php (visited on 05/25/2025).

[18] *Motorola Moto G23 - Full phone specifications*. URL: https://www.gsmarena.com/motorola_moto_g23-12088.php (visited on 05/25/2025).

[19] *PDF Signatur - Ein Service der App „Digitales Amt"*. de. URL: https://www.oesterreich.gv.at/landingpages/pdf_signatur.html (visited on 03/09/2025).

[20] Manuel Pöll and Michael Roland. "Automating the Quantitative Analysis of Reproducibility for Build Artifacts derived from the Android Open Source Project". In: *Proceedings of the 15th ACM Conference on Security and Privacy in Wireless and Mobile Networks*. WiSec '22. New York, NY, USA: Association for Computing Machinery, May 2022, pp. 6–19. ISBN: 978-1-4503-9216-7. DOI: 10.1145/3507657.3528537. URL: https://dl.acm.org/doi/10.1145/3507657.3528537 (visited on 03/28/2025).

[21] Andrea Primativo. *Answer to "Sample code showing how to use Android ID Attestation"*. Sept. 2020. URL: https://stackoverflow.com/a/63694924 (visited on 03/25/2025).

[22] Bernd Prünster, Gerald Palfinger, and Christian Paul Kollmann. "Fides – Unleashing the Full Potential of Remote Attestation". In: *Proceedings of the 16th International Joint Conference on e-Business and Telecommunications* 2: SECRYPT (July 2019). Publisher: SciTePress - Science and Technology Publications, pp. 314–321. DOI: 10.5220/0008121003140321.

[23] *Tecno Spark 10 Pro - Full phone specifications*. URL: https://www.gsmarena.com/tecno_spark_10_pro-12156.php (visited on 05/25/2025).

[24] *Verify hardware-backed key pairs with key attestation | Security*. en. URL: https://developer.android.com/privacy-and-security/security-key-attestation (visited on 03/25/2025).

[25] vvb2060. *vvb2060/KeyAttestation*. Jan. 2025. URL: https://github.com/vvb2060/KeyAttestation/tree/b223a9be86cb2a747625d8e47f3456646d55bd7a (visited on 04/08/2025).

[26] *What is AWS Device Farm? - AWS Device Farm*. URL: https://docs.aws.amazon.com/devicefarm/latest/developerguide/welcome.html (visited on 06/10/2025).

A Appendix: Data

	Attestat. Version	ID Attest.	creationDateTime	osVersion	osPatchLevel	vendorPatchLevel	bootPatchLevel
Blade V30 Vita	3	No	Yes	110000	202112	No	No
Blade V40	3	No	Yes	110000	202402	No	No
Edge 30 Fusion	3	Yes	Yes	120000	202303	20230301	20230301
Edge 40	4	Yes	Yes	130000	202308	20230801	20190606
Fairphone 4	3	No	Yes	110000	202209	20220905	20220905
Fairphone 5	3	No	Yes	130000	202308	20230805	20230805
Find X3 Lite	3	No	Yes	110000	202111	20211105	20211105
Find X5	3	No	Yes	120000	202202	20220205	20220205
Galaxy A14 5G	100	Yes	Yes	130000	202301	20230101	20230101
Galaxy A34	100	Yes	Yes	130000	202306	20230601	20230601
Galaxy A52	3	No	Yes	110000	202109	20210901	20210901
Galaxy A54	200	Yes	Yes	130000	202308	20230801	20230801
Galaxy S10	3	No	Yes	110000	202101	20210101	20210101
Galaxy S20	3	No	Yes	110000	202102	20210201	20210201
Galaxy S22 5G	100	Yes	Yes	140000	202503	20250301	20250301
Galaxy S23	200	Yes	Yes	130000	202112	20230601	20230601
Moto G23	4	Yes	Yes	130000	202305	20230505	20190606
Moto G62	3	Yes	Yes	120000	202211	20221101	20221101
Nokia 3.4	3	Yes	Yes	120000	202306	20230601	20230601
Nokia 5.4	3	Yes	Yes	120000	202309	20230901	20230901
Nokia 7.2	3	No	Yes	110000	202210	20221001	20221001
Nokia 8.3 5G	3	No	Yes	120000	202211	20221101	20221101
Nokia G22 9C NFC	200	No	Yes	140000	202409	20240905	20240905
Nokia G42 5G	3	Yes	Yes	140000	202403	20240305	20240305
Nokia X10	3	No	Yes	140000	202408	20240801	20240801
Nord 3	100	No	Yes	130000	202308	20230805	20230805
Oppo A72	3	No	Yes	100000	202012	20201205	20201205
Pixel 2	2	No	Yes	110000	202010	NA	NA
Pixel 3	3	No	Yes	110000	202105	20210505	20210505
Pixel 4a	3	Yes	Yes	130000	202308	20230805	20230805
Pixel 5	3	Yes	Yes	140000	202310	20231005	20231005
Pixel 6	100	Yes	Yes	120000	202203	20220305	20220305
Poco F3	3	No	Yes	110000	202201	20220101	20220101
Poco F5	100	No	Yes	130000	202304	20230401	20230401
Poco M3 Pro 5G	100	No	Yes	110000	202111	No	No
Poco M4 5G	4	No	Yes	120000	202210	No	No
Poco M5	100	No	No	120000	202208	No	No
Realme GT 2	3	No	Yes	120000	202207	20220705	20220705
Redmi 12	100	No	Yes	130000	202309	20230901	20230901
Redmi 9C NFC	3	No	Yes	100000	202105	No	No
Redmi 9T	3	No	Yes	100000	202104	20210401	20210401
SPARK 10 Pro	4	No	Yes	130000	202309	20230905	20190606
Xperia 1 V	100	Yes	Yes	130000	202307	20230701	20230701
Xperia 10 III	3	No	Yes	110000	202112	20211201	20211201
Xperia 10 IV	3	Yes	Yes	130000	202304	20230401	20230401
Xperia 10 V	3	Yes	Yes	130000	202308	20230801	20230801
Xperia 5 V	200	Yes	Yes	130000	202307	20230701	20230701

Table 3: ID attestation data of all tested phones

	Brand	Announcement Date	ID Attestation
Pixel 2	Google	04.10.2017	No
Pixel 3	Google	09.10.2018	No
Galaxy S10	Samsung	20.02.2019	No
Nokia 7.2	Nokia	05.09.2019	No
Galaxy S20	Samsung	11.02.2020	No
Nokia 8.3 5G	Nokia	19.03.2020	No
Oppo A72	Oppo	21.04.2020	No
Pixel 4a	Google	03.08.2020	Yes
Redmi 9C NFC	Xiaomi	27.08.2020	No
Nokia 3.4	Nokia	22.09.2020	Yes
Pixel 5	Google	30.09.2020	Yes
Nokia 5.4	Nokia	15.12.2020	Yes
Redmi 9T	Xiaomi	08.01.2021	No
Find X3 Lite	Oppo	11.03.2021	No
Galaxy A52	Samsung	17.03.2021	No
Poco F3	Xiaomi	22.03.2021	No
Nokia X10	Nokia	08.04.2021	No
Xperia 10 III	Sony	14.04.2021	No
Poco M3 Pro 5G	Xiaomi	19.05.2021	No
Blade V30 Vita	ZTE	16.07.2021	No
Fairphone 4	Fairphone	30.09.2021	No
Pixel 6	Google	19.10.2021	Yes
Realme GT 2	Realme	04.01.2022	No
Galaxy S22 5G	Samsung	09.02.2022	Yes
Find X5	Oppo	24.02.2022	No
Blade V40	ZTE	28.02.2022	No
Xperia 10 IV	Sony	11.05.2022	Yes
Moto G62	Motorola	09.06.2022	Yes
Poco M4 5G	Xiaomi	15.08.2022	No
Poco M5	Xiaomi	05.09.2022	No
Edge 30 Fusion	Motorola	08.09.2022	Yes
Galaxy A14 5G	Samsung	04.01.2023	Yes
Moto G23	Motorola	24.01.2023	Yes
Galaxy S23	Samsung	01.02.2023	Yes
Nokia G22 9C NFC	Nokia	25.02.2023	No
SPARK 10 Pro	TECNO	06.03.2023	No
Galaxy A34	Samsung	14.03.2023	Yes
Galaxy A54	Samsung	15.03.2023	Yes
Edge 40	Motorola	04.05.2023	Yes
Poco F5	Xiaomi	09.05.2023	No
Xperia 1 V	Sony	11.05.2023	Yes
Xperia 10 V	Sony	11.05.2023	Yes
Redmi 12	Xiaomi	15.06.2023	No
Nokia G42 5G	Nokia	28.06.2023	Yes
Nord 3	OnePlus	05.07.2023	No
Fairphone 5	Fairphone	30.08.2023	No
Xperia 5 V	Sony	01.09.2023	Yes

Table 4: Miscellaneous information of all tested phones. Ordered by announcement date. Announcement dates retrieved from GSMArena

B Appendix: Software

The source-code for the application generating and extracting the key can be found on Github: https://github.com/Kampfdackel5/attestationchecker. The compiled apk to send data to my server is downloadable under the "release" section. If a custom server should be used, the only value that needs to be modified is the url, located at line 135 of MainActivity.java. The server receives the certificate in the DER format, encoded in Base64. We simply used nodejs to receive the information from the app.