Certificate reuse in Android applications

Fatemeh Nezhadian[1][0009−0009−7206−8614], Enrico Branca[2][0000−0001−6316−7789], and Natalia Stakhanova[3][0000−0003−1923−319X]

[1] University of Saskatchewan, Saskatoon, Canada flor.nezhadian@usask.ca
[2] University of Saskatchewan, Saskatoon, Canada enb733@usask.ca
[3] University of Saskatchewan, Saskatoon, Canada natalia@cs.usask.ca

Abstract. The widespread adoption of Android apps has led to increasing concerns about the concept of "recycled trust" derived from the reuse of digital certificates. Android app developers frequently depend on digital certificates to sign their applications, and users place their trust in an app when they recognize the owner provided by the same certificate. Although the presence of cryptographic misuse has been acknowledged by several studies, its extent and characteristics are not well understood. In this work, we perform a large-scale analysis of certificate reuse across the Android ecosystem and malware binaries on a collection of over 19 million certificates and over 9 million keys extracted from PE files and Android applications collected over several years. Our results reveal that despite the growing nature of the Android ecosystem, the misuse of cryptographic elements is common and persistent. Our findings uncover several issues and enable us to provide a series of applicable solutions to the seen security flaws.

Keywords: Cryptography · Android · Malware · Digital Certificates

1 Introduction

Cryptography plays a crucial role in the Android ecosystem. Cryptographic operations are used to enable secure storage of sensitive information, verify the authenticity of the applications, and protect communication. The underlying system that enables this functionality is Public Key Infrastructure (PKI). PKI is a system that manages digital certificates and cryptographic keys and establishes trust between participating parties by associating a public key with an entity and enabling the verification of their identity. The trusted Certificate Authority (CA) plays a vital role in this infrastructure issuing certificates to software vendors and attesting to their identity.

Digital certificates and keys are essential components that enable secure communication, code signing, authentication, and other security-related features within apps. They play a vital role in establishing trust and ensuring the integrity of the Android ecosystem. However, the improper use or mishandling of these certificates and keys poses a substantial risk to the overall security of the ecosystem.

Instances of compromised certificates and keys within the PKI ecosystem are not uncommon. The infamous Stuxnet worm and Diqu malware were signed with legitimate digital certificates [6, 15]. The study by Kim et al. [9] demonstrated that malware uses valid certificates to evade anti-virus programs and bypass system protection mechanisms. Kang et al. [8] showed that analyzing the serial numbers of certificates can potentially reveal indicators of Android malware. While these examples provide evidence of valid digital certificates being misused, it remains unclear whether this phenomenon is limited to a specific domain and what the broader security implications are.

In this work, we conduct a systematic study to measure and characterize the use of compromised digital certificates across two domains. We focus on Android applications (apps) and Windows executable files.

One of the challenges in this context is to collect compromised certificates and keys, as there is no official service that provides a list of all compromised certificates. To overcome this problem, we collect malicious binaries and Android apps to extract their corresponding digital certificates. We analyze the reuse of these deemed to be compromised certificates among the apps.

Our findings show that certificate reuse is more pervasive and widespread than previously observed, 48% of our collected certificates (over 9 million) are reused across the collected sets of APK and malicious binary files. Among them, 40% of the certificates used to sign malware binaries are also reused in Android malicious apps for various purposes. In other words, these certificates are extensively reused in malware across domains.

Although using the same certificate for signing multiple Android applications is generally discouraged by Google, we found this practice commonly ignored by both benign and malicious apps. For example, 59% benign apps in our collections were signed with duplicate certificates. At the same time, we discovered 9,931 unique certificates employed to sign 142,579 malicious and 84,922 benign apps.

To summarize, this study makes the following contributions:

- We conduct the most comprehensive and the largest analysis of cryptographic elements across Android applications.
- We measure and characterize the extent of certificates and RSA public keys sharing across APKs and malware binaries on a diverse set of over 19 million valid certificates and over 9 million reused keys collected from multiple sources over a period of several years (up to eleven years in some cases).
- Based on our analysis, we provide a set of recommendations to help security practitioners improve overall cryptographic security.
- To facilitate analysis of cryptographic reuse, we make the set of reused certificates publicly available[4,5].

[4] https://key-explorer.com/
[5] https://github.com/thecyberlab/RSA-keys-analysis

2 Background

APK (Android Package Kit) file acts as a bundle containing all the necessary components and resources of an Android application, including the compiled code, assets, resources, cryptographic files (e.g., digital certificates, keys), etc. In Android applications, digital certificates and keys are used for various purposes:

Integrity and authenticity of the APK: The Android apps have to be signed, using unique digital certificates, before distribution. This mechanism not only ensures the integrity of the application but also provides Google, as the official market, with confidence that the owner's identity has been verified. However, Android apps can be self-signed by the owner, or signed with a verified third-party certificate.

There are two ways to create a key pair for signing apps[6]. The first involves using embedded manager tools in the development environment, such as Android Studio (apksigner, jarsigner) or Microsoft Visual Studio (archive manager), to automatically generate a keystore (a secure storage container where applications store and manage cryptographic keys and certificates) and a certificate with the identity information of the app's owner. Alternatively, developers can configure a personalized cryptographic key pair to create a custom key, allowing them to define their preferred signature and digest algorithms and key size. While this approach provides greater flexibility, it may also allow for weaker configurations. In the end, the cryptographic key pair, along with the owner's identity information, forms a signing certificate used to sign the APK file and consequently perform APK validation during installation.

Data protection and privacy: Cryptographic keys are employed to encrypt sensitive data within Android apps, safeguarding user information, passwords, and app-specific data from unauthorized access.

Authentication: Android applications that integrate with external services or APIs may use authentication and authorization services, e.g., OAuth. These credentials are used to authenticate the app and obtain access tokens for accessing protected resources.

Portable Executable (PE) Like APKs, other software applications, including executable files, utilize digital certificates and cryptographic keys for ensuring data integrity and authentication. PE serves as the standard format for Windows executables (.exe) and dynamic link libraries (.dll). Microsoft code-signing technologies, Authenticode and SignTool, are widely used for code signing and digital signature verification of Windows executable files, including PE (.exe), dynamic link libraries (.dll), installers (.msi), and other file types.

Cryptographic infrastructure *Digital certificates:* serve as an attestation of the identity of a certificate's owner (e.g., hostname, organization) bound to its public key. The X.509 format is one of the most widely used standards for digital certificates that, in addition to the public key and owner's identity, contains a

[6] https://developer.android.com/studio/publish/app-signing

period during which the certificate is considered valid, and a digital signature of the issuing certificate authority (CA). Generally, CA is a trusted third party that can vouch for the identity of a server/certificate's owner by signing the leaf certificate with its private key.

Pretty Good Privacy (PGP) and GNU Privacy Guard (GPG): PGP is cryptographic technology for secure communication. GPG, often seen as an alternative to PGP, is an open-source implementation of the OpenPGP standard. Both are widely used for securing email and data encryption. The standard extensions for such files include pgp, asc, sig, gpg, pubkey_pgp, seckey_pgp, and secring_pgp.

3 Related work

Over the past decade, several studies explored the misuse of cryptographic APIs in Android applications. The vast majority of the approaches use verification-based analysis that offers assurances for the correct implementation of cryptographic primitives. For example, the static analysis frameworks CryptoLint [4] and BinSight [12] perform analysis of cryptographic misuse at scale. CryptoLint [4] discovered that 88% of 11,748 Android applications had at least one mistake in API use. BinSight [12] in its follow-up analysis showed that cryptographic API misuse originated in third-party libraries. Specifically, they observed that 90% of 132,000 Android apps violated cryptographic API guidelines. Similarly, Gao et al. [7] found that 96% of the analyzed 8 million APKs from the AndroZoo [1] dataset exhibited some crypto-API misuses.

CRYLOGGER [13] employed dynamic analysis for the detection of cryptographic misuse in Android apps. Similar to CryptoLint and BinSight, CRYLOGGER explored the correctness of cryptographic API calls based on the defined rules (e.g., constant keys, weak passwords).

Zhang et al. [18] proposed LibExtractor to detect potentially malicious libraries and identify malware families based on their digital certificates.

Wickert et al. [16] focused on the crypto misuses of two Java libraries, the Java Cryptography Architecture (JCA) and Bouncy Castle (BC). Their study of 936 open-source Android apps showed that 88% of their collected apps failed to use cryptographic APIs securely.

Numerous studies explored more generic detection of cryptographic misuses. Li et al. [11] conducted a large-scale analysis of API misuse in GitHub projects. K-Hunt focused on weak cryptographic keys through analysis of binaries [10]. CryptoGuard [14] detected cryptographic misuse in Java programs including 6,181 Android apps. Similar to other studies, CryptoGuard discovered that around 95% of discovered vulnerabilities originate from libraries that are packaged with the application code.

Zhang et al. [17] have proposed CryptoREX, a framework to identify cryptographic misuse of IoT devices. Analyzing 521 firmware images with 165 predefined cryptographic APIs, CryptoREX showed that 24.2% of firmware images violate at least one misuse rule.

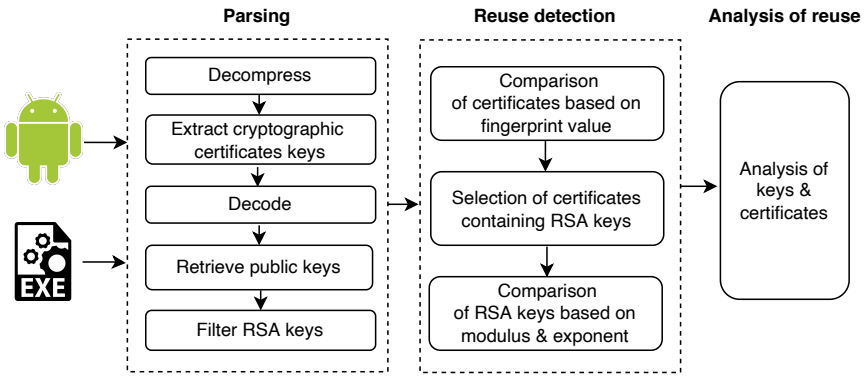

Fig. 1: The flow of the analysis.

All these approaches focus on whether cryptographic functionality is implemented by the cryptographic libraries embedded in Android correctly. Hence, their primary focus is on benign applications. We, on the other hand, investigate the reuse of certificates. Kim et al. [9] was the first study to explore the presence of PE files signed by malicious certificates. We take this further and investigate the extent and characteristics of malicious certificates across domains of Windows binaries and Android apps including malicious and benign apps.

4 Methodology

The goal of our study is to measure and characterize the extent of compromised certificates and key reuse across Android apps. The flow of the analysis, illustrated in Figure 1, includes 3 main steps.

In this study, we analyzed a large collection of Android apps and Windows executable files. For each of the files, we extracted all cryptographic elements such as digital certificates, and public and private keys that were included in the Android application packages and PE files.

The Android apps were unpacked and analyzed for the presence of cryptographic keys, i.e., files with standard extensions indicating the presence of cryptographic materials such as rsa, pem, crt, and cer. During this process, we observed the presence of files that did not match any of the standard extensions, yet appeared to contain digital certificates and keys. We thus parsed the remaining files using the Linux *file* command to identify hidden files that previously had standard cryptographic extensions that were later changed.

Malware binaries were parsed using GoLang's sigtool[7], which is a PE package designed to extract information from PE files. The certificates used for signing code are contained in the "Attribute certificate" section of PE files in DER

[7] https://github.com/doowon/sigtool

Table 1: The summary of collected Android apks

Source	Collection Period	#apks	Valid apks	#apks with crypto
AndroGalaxy	2017 - 2019	7,462	6,845	6,839
AndroidAPKsFree	2020	1,333	1,316	1,312
Anzhi Market	2017, 2020	5,894	5,842	5,840
APKGOD	2020	4,690	4,046	4,044
Apkmaza	2020	111	109	109
APKPure	2020, 2021, 2023	109,216	109,048	108,512
AppsApk	2020	6,146	5,848	5,845
Appvn	2020	33,986	33,311	33,304
CracksHash	2021, 2022	3,486	3,469	3,461
F-Droid	2020	7,073	7,065	7,065
Google Play Store	2020, 2023	5,468	5,283	5,222
1Mobile Market	2020	1,370	1,370	1,370
Mob.org	2020	1,147	1,141	1,141
SlideME	2020	18,052	18,049	18,049
Uptodown	2020	59,717	56,819	56,686
VirusShare	2012 - 2023	440,106	411,629	411,214
VirusTotal	2020, 2021	8,160	98	85
Xiaomi	2020	1,199	1,175	1,175
Total	-	714,616	672,463	671,273

format. For consistency, the extracted certificates were converted to the PEM format, which was then parsed to extract the certificate and key information.

In the next step, we obtained and verified signature schemes, package integrity, and package manifest using "apksigner" and AAPT2 (Android Asset Packaging Tool) included in Android Studio SDK build tools and APK parser3[8]. At this stage, we discarded a set of 32,003 APKs that were not successfully compiled or verified.

Several cryptographic libraries were used to parse the files collected from the APKs and the PE files for certificates and keys. These libraries include the Java library keytool[9], and the Python libraries: CERT Keyfinder[10], PyOpenSSL, Cryptography, and PyJKS. However, not all the certificates and keys are parsable due to corrupted formats, password protections, or outdated (no longer supported by libraries) standards.

In order to assess the reuse within the collected data, we conducted a pairwise analysis of the collected certificates using their fingerprints, i.e., the SHA-1 hash of the certificates. Along with matching certificates, we also compared valid RSA keys.

5 Data

To ensure a robust dataset, we collected Android apk files from several Android app distribution platforms, including Google's official market called Google Play Store, and Chinese app stores, which cater primarily to Chinese users, such as

[8] https://github.com/itomsu/apk_parse3
[9] https://docs.oracle.com/javase/8/docs/technotes/tools/unix/keytool.html
[10] https://github.com/CERTCC/keyfinder

Xiaomi and Anzhi; free open-source repositories for Android apps such as F-Droid and AndroidAPKsFree; markets focused on game apps such as Mob.org; several unofficial repositories such as APKPure, Appvn, AppsApk, SlideME, Uptodown, APKGOD, Apkmaza, and CracksHash; and stores allowing a direct download of apps such as AndroGalaxy, and 1Mobile Market.

We also collected 448,266 apps from malware repositories VirusShare and VirusTotal.

By utilizing a combination of these sources, we aimed to gather a representative sample of Android apk files, encompassing benign used by different categories of users and malicious apps. Our set included benign applications collected between 2017 and 2023 thus presumably following the most recent standards and malicious apps from 2012 to 2022. Our final set consists of 714,616 of apk files summarized in Table 1.

From this set, 672,463 were found to be valid, i.e., parsable by the official Android tool, AAPT2, which verifies package correctness and integrity. Surprisingly, 1,190 apps did not contain any cryptographic components, which was unexpected. Although Google installation requirements require apps to be signed for distribution through the official Google Play application[11], we expected all Android developers to follow this security practice.

Overall, we were left with 671,273 applications containing cryptographic elements for our analysis. Although benign apps were collected from legitimate sources, we verified them using VirusTotal service and Malshare and VirusShare hashes. We gathered 3,266,932 hash values of malicious binaries from the Malshare Daily Digest[12] (covering the period from September 2017 to July 2023) along with 40,894,458 hash values of the malware samples provided by VirusShare[13]. We further matched our benign set against these hashes. As a result, 247 apk files from benign sources were detected as malicious. The rest of our analysis was conducted on a set of 259,677 benign apps (38.7%) and 411,596 malicious apk files (61.3%).

In addition to Android apps, we collected a set of 40,270,387 files in PE format reported as malicious from VX underground's APT collection[14] and VirusShare repository[15] ranging from 2012 to 2021.

6 Analysis

In our study, the collected cryptographic files were parsed to identify the presence of certificates and keys. In cases where APK files included the signing certificate within the signature block rather than a distinct cryptographic file, we also extracted those certificates.

[11] https://developer.android.com/google/play/requirements/target-sdk
[12] https://malshare.com/daily/
[13] https://virusshare.com/hashes
[14] https://vx-underground.org/apts.html
[15] https://virusshare.com

Table 2: Summary of RSA certificates and public keys from APK files

Source	Certificate			Public key		
	Total	**In files**	**In signature block**	**Total**	**From certificates**	**In files**
Malicious APK	789,117	789,117	0	802,117	789,117	13,000
Benign APK	778,260	778,044	216	793,980	778,260	15,720
Total *	1,567,377	1,567,161	216	1,596,097	1,567,377	28,720

* duplicates within sets are removed, across sets retained

After filtering only RSA certificates and keys, we obtained 789,117 certificates and 802,117 public keys distributed across 411,596 malicious Android apps and 778,260 certificates and 793,980 public keys extracted from 259,677 benign apps (Table 2). Parsing 40,270,387 malware binary files, we identified 18,081,489 RSA certificates and their corresponding public keys. Overall, we derived 19,648,866 certificates for our analysis (Tables 5).

In the absence of an official repository providing a comprehensive list of compromised certificates, we focused on the certificates associated with instances of PE files and APKs that were officially reported as malicious. We consider these certificates to be compromised (as the adversary likely has access to the corresponding private key), and in short, we refer to them as *malicious certificates*.

6.1 Cryptographic file formats

Out of 671,273 APKs analyzed, we discovered 2,376,721 files that may contain cryptographic components indicating digital certificates and keys (Table 3).

APKs typically incorporate a range of cryptographic components, utilizing diverse encryption algorithms, each serving specific purposes. Cryptographic file formats can exhibit varying configurations of cryptographic elements. As our analysis showed, cryptographic components may appear in file formats not related to cryptographic extensions, hence we parsed all collected files.

We discovered that not all of the initially identified file extensions within our collected APKs reflected the actual content of the file, i.e., many appeared to be renamed. This phenomenon typically happens when the original file extensions are changed, potentially to disguise or obfuscate the file content. Overall, out of 2,376,721 crypto-related files, 1,433,458 (60.3%) have been found renamed. The summary of renaming instances is presented in Table 4, where we can clearly observe two major recurring patterns.

Files containing certificates and keys (i.e., with file extensions appkey, pubkey_pgp, and seckey_pgp, along with others like pem, jks, exe, key, der, and csr) are commonly stripped of their original extensions (showed as <None>) or changed to pose as innocuous extensions. For example, a large number of files containing pgp keys were renamed to appear as image files, i.e., with png and jpg extensions.

Among the 104,044 files without extensions, 87,072 (83.7%) were identified as being in the "appkey" format. While the remaining formats were distributed

Table 3: Summary of parsable cryptographic files in apps

Category	Unique	Total	Benign apps	Malicious apps
All files	1,216,354	2,376,721	1,246,846 (52.46%)	1,129,875 (47.54%)
Files containing certificate(s)	646,176	826,674	294,323 (35.60%)	532,351 (64.40%)
Files containing public key(s)	2,150	28,872	15,764 (54.60%)	13,108 (45.40%)
Files containing private key(s)	500	1,604	425 (26.50%)	1,179 (73.50%)

Table 4: The summary of renamed file extensions

Renamed extensions	Total files	Unique	Malicious APK		Benign APK		Most frequent original extensions
			Total	Unique	Total	Unique	
exe	979,127	240,857	272,002	78,253	707,125	178,900	.dll, .temp, .binary, <None>, .so
seckey_pgp	217,462	145,336	110,759	74,191	106,703	68,564	.enc, .bin, .png, .html, .lhs
pubkey_pgp	139,184	89,700	84,809	55,307	83,998	55,969	.enc, .html, .png, <None>, .jpg
appkey	87,072	56,615	55,186	35,092	5,253	2,170	<None>
pem	10,116	1,244	4,863	562	2,263	911	<None>, .0, .jpg, .cer, .txt
jks	389	138	250	88	139	60	<None>, .jilin, .pro, .ts, .keystore
key	56	31	27	12	29	20	<None>, .txt, .mqtt, .dat
der	16	5	10	1	6	4	<None>, .pk, .ab, .split4
bks	13	1	9	1	4	1	<None>
cer	7	1	0	0	7	1	<None>
pfx	7	2	3	1	4	1	<None>
crt	6	2	4	1	2	1	<None>
keystore	2	1	2	1	0	0	<None>
csr	1	1	1	1	0	0	<None>
Total	1,433,458	533,934	527,925	243,511	905,533	306,602	.dll, .enc, <None>, .png, .bin

randomly throughout the apps' file structures, the "appkey" files were specifically located either in the "assets" or the "assets/res" folders. Generally, the application key is the signature of the public key certificate of the private key, that is used to sign the APK, stored in a text format. Devices should only accept updates from an app when its signature matches the installed app's signature as a secure process. Another visible pattern is the renaming of Windows executables from exe extension to dll extension. This practice can help evade security measures and mislead users or analysts by disguising standalone executables.

Malicious apps appear to have fewer certificates in general, 817,479 in 411,596 apps, compared to benign apps, 820,997 in 259,677 apps (see Table 9). Having fewer certificates can help malicious apps avoid detection and maintain a low profile in their malicious activities. Similar behavior has been observed with public keys.

On the other hand, more private keys have been seen in malicious apps, (1,174 compared to 423) (see Table 9) which may be necessary to facilitate the decryption of encrypted malware data (e.g., in ransomware cases).

6.2 Reused certificates

Out of 19,648,866 RSA certificates, 9,412,099 (48%) are reused across the collected sets of APK and malicious PE files, with 11,251 (0.12%) of them being unique instances. As the results in Table 5 show, there is significant duplication of certificates within and across sets. Even more interesting is the presence of

Table 5: Shared Certificates

Source	Total Certificates	Unique Certificates	Unique shared per set**	Shared across			
				Total	Malware binaries	Malicious APKs	Benign APKs
Malware binaries	18,081,489	41,282 (0.23%)	194	421,175	*	166,844	254,331
Malicious APKs	789,117	146,329 (18.54%)	11,234	5,224,399	4,629,047	*	595,352
Benign APKs	778,260	135,895 (17.46%)	11,213	3,766,525	3,256,951	509,574	*
Total	19,648,866	323,506	22,641	9,412,099 (48%)	11,251 (0.12%) are distinct across sets		

** duplicates within a set are removed, across sets retained

significant overlap between sets, which highlights the extensive reuse of certificates between benign apps and malware, including malicious apps and binaries. We will explore each of these aspects further.

Reuse of signing certificates from malware binaries As shown in Table 5, out of a total of 18,081,489 signing certificates extracted from malware binaries, 2% (421,175) were found to be reused in our collected set of APKs. To our surprise, only 3 of these certificates have been used for signing malicious apps, the rest were widely used for other purposes.

Around 60% (254,331) of these compromised certificates were reused in benign apps and as we saw in other instances of reuses, these certificates were heavily duplicated, where only 156 were unique. These apps are present in our Google Play Store and alternative market collections indicating that this reuse practice has been continuing over time.

In our analysis, we found that 40% (166,844) of the certificates found in malware binaries are also reused in Android malicious apps. The use of signed malware is not a new phenomenon, numerous sources reported that legitimate certificates are readily available for purchase in underground markets[16]. The previous study by Kim et al. [9] showed the use of legitimate certificates to sign malicious Windows binaries. However, our latest findings demonstrate that this practice is even more pervasive and widespread than previously observed. These certificates are being employed in malware across various domains and are extensively used.

During our investigation, we found 45 benign apps that were reported by Malshare and VirusShare as malware samples due to the contained cryptographic content. These benign apps appeared in both the official Google Play Store and alternative markets over several years (2019 to 2023). Further investigation revealed a total of 11 unique certificates were embedded in these apps. These files were flagged as malicious by multiple vendors and reported by VirusTotal. A pairwise match of certificates disclosed the usage of such certificates for signing 1,993 apps including 1,920 malicious apps and 73 benign apps.

[16] https://cyware.com/news/certificate-authorities-duped-to-sell-legitimate-digital-certificates-that-can-spread-malware-bcf63b15

Reuse of APK Signing Certificates APKs are structured files that can include a signing digital certificate as a cryptographic file introduced in either a stand-alone META-INF file or included inside a signature block, depending on the version of the signature scheme. We parsed each APK to identify the presence of all signing certificates. As a result, out of 671,273 valid APKs, the majority (668,392) were digitally signed, while $< 1\%$ (2,881) lacked signing certificates, including 2,134 malicious apps, and 747 benign apps.

During this process, we discovered that the "jarsigner" tool treats a significant number of 594,971 (89%) signed apps as unsigned, issuing warnings due to deprecated signature algorithms and weak key sizes included in the signing certificate.

Out of 258,930 digitally signed benign apps, 59% (153,294) apps were signed with 25,135 certificates indicating significant reuse of certificates among benign apps. Using the same certificate for multiple Android applications is generally discouraged. Reusing certificate makes it challenging to determine the true source and verify the integrity of the application. If one app signed with a shared certificate becomes compromised, it can have significant implications for the security of all other apps that utilize the same signing certificate.

However, there are instances where a developer might reuse a certificate, for example, for different versions of their application or to facilitate communication between apps that belong to the same organization. A closer manual analysis of reused benign certificates showed that these legitimate cases are only responsible for a small portion of reuse. For example, one certificate has been used 6 times to sign apps belonging to Amazon Mobile LLC (Amazon Prime Video, Amazon Shopping, and Amazon Music). Similarly, another certificate has been used 4 times to sign apps published by Microsoft Corporation (Microsoft 365 and Microsoft Teams). Yet, our findings show that not all signing certificate reuse cases are related to developers following these legitimate practices.

Surprisingly, 9,931 unique certificates have been employed to sign 142,579 malicious apps, 34% of total 411,596 apps, while at the same time, these certificates have been also used to sign 84,922 benign apps, 32% of 259,677 apps. These benign apps were collected from all sets, excluding the SlideMe market which appeared to repackage and sign all posted apps with its market's certificate.

In November 2022, several platform certificates have been discovered to be used for signing malware[17]. The so-called platform certificates are used to sign the system Android apps, and thus give elevated privileges to apps signed with these certificates. Hence, if a malicious application is signed with such platform certificate, the Android OS will treat the malicious app with the same elevated access as a legitimate system app. Surprisingly, we found 332 apps in our collected set signed with 5 of the reported leaked platform certificates, within both our benign and malicious set of APKs, corresponding to apps released in 2023 and present in Google Play Store, and in sets dating as back as 2014.

The most shared default signing certificates are shown in Table 6. The top most widely used certificate is the default certificate of Android Studio, used

[17] https://bugs.chromium.org/p/apvi/issues/detail?id=100

Table 6: Use of known and default certificates in apps

SHA-1	Name	Total APKs	Benign APKs	Malicious APKs	Apps' Sources
61ED377E85D386A8DFEE6B864BD85B0BFAA5AF81	testkey	46,930	12,639	34,291	AndroGalaxy, AndroidAPKsFree, Anzhi Market, APKGOD, Apkmaza, APKPure, AppsApk, Appvn, CracksHash, Uptodown, VirusShare, VirusTotal, Xiaomi
27196E386B875E76ADF700E7EA84E4C6EEE33DFA	platform	1,230	9	1,221	APKPure, Appvn, Uptodown, VirusShare
5B368CFF2DA2686996BC95EAC190EAA4F5630FE5	shared	927	781	146	AndroGalaxy, Anzhi Market, APKPure, Appvn, Uptodown, VirusShare
B79DF4A82E90B57EA76525AB7037AB238A42F5D3	media	236	121	115	AndroGalaxy, APKGOD, Appvn, Uptodown, VirusShare
C0DE76E80C8F1BFEDAC64231B9582DF0EBC4F19E	SlideME	18,049	18,049	0	SlideME
9EDF7FE12ED2A2472FB07DF1E398D1039B9D2F5D	Qbiki Networks	1,590	1,441	149	AndroidAPKsFree, APKPure, Appvn, Google Play Store, 1Mobile Market, Mob.org, Uptodown, VirusShare
Total	-	68,962	33,040	35,922	-

in 12,639 benign and 34,291 malicious apps. This certificate, also known as "testkey", is one of the four key pairs that are generated by the Android team in the Android Open Source Project (AOSP) and are located in the "release-keys" folder. The other three pairs ("platform", "shared", and "media") are used to sign 911 benign apps and 1,482 malicious apps in total. However, it is crucial for developers to avoid using these default keys since they are publicly known. When multiple apps are signed with such certificates, they often gain a privileged position, granting them special access to those apps. As a result, if a malicious app is signed with the same certificate, it may gain elevated access to sensitive resources that would otherwise be inaccessible.

We also discovered 18,049 apps signed with a certificate associated with the SlideMe market. It appears that this certificate has been used to replace the original signing certificate in order to publish apps in the market. Another case of certificate reuse involves a service provider named "Qbiki Networks". The provider enables customers to create mobile apps with minimal coding and signs these apps on their behalf. This case was initially reported by Fahl et al. [5] back in 2014. Interestingly, after several years we still observe a similar situation in 1,590 apps in our benign and malicious sets containing apps from 2014 to 2022. The practice of certificate reuse by Qbiki Networks' customers seems to persist over time.

We were able to detect the presence of a total of 68,962 (10.27%) apps signed with these known key pairs (Table 6).

Reuse beyond signing certificate Another concern in this context is the reuse of signing certificates for other purposes beyond their intended use. Signing certificates are meant to verify the authenticity and integrity of specific software applications or digital documents.

We further examined the reuse of signing certificates for other operations. As a result, we found 297 unique signing certificates reused within 70,077 apps, including 20,758 benign and 49,319 malicious apps.

The CAs (Certificate Authorities) define the purpose of the keys when issuing digital certificates through designated fields known as *Key Usage*, *Extended Key*

Table 7: Indented purposes of reused certificates

Characteristic	Unique	Total	APK signing certificate	Across sources		
				Benign APKs	Malicious APKs	Malware binaries
Key Usage	940	5,188,368	1,731	393,917	291,290	4,503,161
Digital Signature	524	852,104	1,731	98,494	73,862	679,748
Certificate Sign	568	5,179,277	0	390,262	285,865	4,503,150
Key Encipherment	342	22,210	10	9,938	6,143	6,129
Data Encipherment	31	1,774	0	832	942	0
Key Agreement	24	14,766	0	6,462	2,182	6,122
Extended Key Usage	492	810,480	1,954	16,205	15,983	778,292
Code Signing	116	764,571	1,834	1,672	2,777	760,122
TLS Web Client Authentication	373	43,911	55	4,322	11,263	28,326
TLS Web Server Authentication	374	56,568	0	14,851	13,389	28,328
Time Stamping	21	99,501	0	1,218	854	97,429
E-mail Protection	22	12,535	0	1,346	1,025	10,164
Microsoft Commercial Code Signing	12	168	0	42	126	0
Basic Constraints	1,322	5,591,123	63,589	458,363	376,025	4,756,735
CA: True	962	5,587,165	63,285	458,363	372,079	4,756,723
CA: False	360	7,384	304	3,426	3,946	12

Usage, and *Basic Constraints*. These extensions provide additional insights into permitted cryptographic operations and the intended purposes of the associated public key such as digital signature, key encipherment, client authentication, or code signing. These extensions enable certificate verifiers to assess the suitability of cryptographic operations and enforce robust security measures.

In other words, keys designated for signing code cannot be reused for other purposes. Yet, as our results show the key purpose does not appear to be properly verified.

Out of 9,412,099 reused certificates, 202,997 (2.2%) certificates were found to be lacking any extensions, i.e., theoretically should not have been signed by CAs. Out of the remaining certificates, 5,605,334 certificates have at least one of the extensions which means at least some constraints have been declared regarding their usage.

The extension characteristics of all reused certificates are summarized in Table 7. The results indicate that the absence of proper configurations and clear constraints for a signing certificate can result in the same certificate being reused across multiple domains.

Surprisingly, 5,179,277 (55%) certificates were labeled with "Certificate Sign" in their extensions, allowing them to sign other certificates and create a certificate hierarchy. Such certificates enable the certificate holders to act as trusted entities, issuing and signing certificates for subordinate authorities or entities. These certificates typically belong to CAs, and the presence of these 4,503,150 certificates in malware binaries raises concerns. We have only extracted code-signing certificates from malicious binary files, hence, the presence of these privileged certificates in signing malicious apps suggests potential unauthorized certificate use.

Starting from 2008, certificate extensions have been categorized as either critical or non-critical. If a certificate-using system encounters critical extensions

Table 8: Public key size of reused certificates

Key Size	Unique Keys	Total Keys	Benign APK	Malicious APK	Malware binaries
0-1023	9	728	649	47	32
1024-2047	4,540	731,740	306,780	325,600	99,360
2048-4095	6,294	8,208,281	3,256,459	4,665,343	286,479
4096-8191	287	471,348	202,636	233,408	35304
8192-up	1	2	1	1	0
Total	11,131	9,412,099	3,766,525	5,224,399	421,175

or information it cannot handle, it must reject the certificate. On the other hand, non-critical extensions can be disregarded if they are unrecognized, but they should be processed if they are recognized [3]. To ensure backward compatibility between applications and older versions of Android, applications may decide to implement a custom SDK overwrite that forcefully disables the verification of certificate extensions flagged as critical. Our analysis showed that malicious apps tend to use key extensions flagged as critical more often than benign apps. Out of 4,730,060 (50.2%) certificates set as critical, the vast majority (4,160,892) belongs to malware binaries, 245,592 to malicious apps, and 323,576 to benign apps.

Thus it appears that malware not only uses privileged certificates (i.e., the certificates issued to allow the signing of other certificates) but also commonly requests certificate verification to fit the target profile.

In the context of APKs and PE files, the presence of the CA flag set to True in the *Basic Constraints* extension indicates that the certificate is associated with a CA, signifying it as a trusted organization that has verified and signed the application or software from the vendor or developer. On the other hand, Android does not mandate apps to be signed by a CA and does not currently perform CA verification. It also provides code signing using self-signed certificates that developers can generate without external assistance or permission. However, a self-signed CA certificate implies that the owner of an apk file assumes the role of a certificate authority and has the authority to issue, validate, and sign other certificates for various purposes.

Out of 5,587,165 reused certificates found to be flagged as CA, we discovered 2,869,140 (51.35%) are self-signed distributed as 2,106,072 in malware binaries, 450,346 in benign apps, and 312,722 in malicious apps.

These findings highlight the significant reuse of signing certificates in the ecosystem of Android applications.

Public keys present in reused certificates We conducted a deeper analysis of the key strength of the reused cryptographic elements found in our set to gain insights into the level of protection they offered. Table 8 presents the distribution of key size ranges for the public keys extracted from the reused certificates.

Among the reused keys, 732,468 (8%) are less than 2048 bits in length. They are considered cryptographically weak and should not be used for cryptographic protections. For example, NIST-compliant RSA keys are required to have a length greater or equal to 2048 bits [2]. NIST also recommended deprecating

signing certificates that contained RSA keys of 1024 bits by the end of 2013. However, across all our scans, 528 signing certificates were found using a deprecated public key with a length of less than 1024 bits.

During our analysis, we encountered 1,597 private keys, out of which only 418 are unique. The presence of unencrypted and reused private keys in apps is concerning. Depending on the intended usage, the presence of these shared private keys opens up the possibility for misuse, allowing to decrypt protected data or hijack another app identity.

Out of these 418 unique private keys, we successfully reconstructed 251 RSA public keys, and by pairwise comparison with our existing collections, 34 shared public keys and 29 shared certificates were found to be matched to these private keys. Overall, 563 certificates and 1,108 public keys were found in 819 apps, distributed as 617 malicious apps and 202 benign apps, dated from 2012 to 2023.

7 Observations and Recommendations

The prevalence of compromised certificates being reused across Android apps and malicious binaries is substantial. Yet, our analysis highlights several observations that underline the existing problems and enable us to propose the following potential countermeasures:

Adequate context-relevant extensions: We suggest defining context-relevant extensions with careful use of CA and *critical* flags to diminish the likelihood of potential certificate reuse for various purposes and in multiple domains. More specifically,

- *Specified/non-generic certificates:* Our analysis shows that only 5,605,334 certificates (59.5%) in our large-scale collection are well-defined. Without well-defined certificate extensions, relying parties have limited insight into the intended or recommended use of the certificate. This makes it challenging to enforce appropriate security measures and determine whether the certificate is suitable for specific operations or applications.
- *Non-CA signing certificates:* 4,756,723 certificates of malware binaries along with 63,285 signing certificates in APKs are set to be CA. If a signing certificate is designated in this way, it inherits the elevated and arguably unnecessary authority to issue new certificates.
- *Mandated purpose-related extensions*: Only 4,730,060 certificates (50.25%) mandate verifiers to process the purpose-related components of certificates. If a verifier lacks support for a critical extension, it can safely ignore such extension without affecting the overall validation process. Properly setting critical flags enhances the certificate's reliability while allowing for graceful handling of unsupported extensions by verifiers.

Use of prevention mechanisms may serve as a simple solution to vet apps, such as

Algorithm 1 An algorithm for verifying signing certificate

```
Step 1:                                   ▷ Save hash value while creating the signing certificate
ExpectedAppKey ← signature of signing key pair
Step 2:                                                    ▷ Hard-coded validation procedure
procedure OnStart
    package_manager ← AndroidPackageManager
    package_info ← package_manager.GetPackageInfo()
    received_app_key ← package_info.GetSignature()
    if ExpectedAppKey <> received_app_key then return false
    end if
     Signature verified. return true
end procedure
```

- *Avoiding the use of default or publicly known certificates:* 49,323 apps (7.3%) in our set were signed with Android's default certificate and 19,639 apps (2.9%) were signed with publicly known certificates.
- *Use of reported malicious and compromised samples:* 1,993 benign apps in our set contain malicious files, and 332 apps were affected by the use of compromised platform certificates. Our analysis relied on publicly available information that is readily available to any developer.
- *Avoid using not-protected private keys:* We were able to extract 1,597 private keys from malicious and benign apps, and consequently 1,108 public keys and 563 certificates. In Android application development, it is advised not to package unencrypted private keys in APK files and refrain from including the signer certificate's private key within the APK. Securely storing private keys in trusted environments, such as servers or hardware security modules, with limited access during the signing process is essential to enhance app security and safeguard cryptographic assets.

Validate expectations:

- *Embedded validation procedures:* Apart from considering all the settings and configurations of keys and certificates, an APK certificate should be further verified. We propose to use a set of steps in Algorithm 1 to guide app certificate validation. This process serves as a strong protection, ensuring that the APK's integrity remains intact and aligns with the expected attributes.
- *Use of tools:* The "jarsigner" tool used to verify signing certificates can give the "security risk" warning due to the use of deprecated signature algorithms, weak key sizes, and self-signed entries. This tool also informs if the *Extended Key Usage* extension allows the certificate to be used for code signing. Using such tools is encouraged to evaluate the signing certificate in order to reduce the risk of malicious modifications during distribution.

8 Conclusion

Digital certificates play a crucial role in Android app security, but many developers prioritize convenience over security. While there are cases where it may be justified – the app may not contain any important or identifying information – in

numerous instances, it poses substantial risks to users and app owners. Our study reveals the extent of certificate reuse in Android apps and the widespread presence of compromised certificates. While reusing signing certificates in Android apps can simplify the app management process and maintain user trust, it also comes with significant security considerations. We hope that this research will urge developers to reassess the current security practices. Prioritizing certificate security is crucial for safeguarding both users and apps.

9 Appendix

Table 9: File formats containing cryptographic elements

Identified extensions	Total	Unique	Parsable	Benign APK			Malicious APK		
				Certificates	Public Keys	Private Keys	Certificates	Public Keys	Private Keys
aes	19,671	5,439	0	0	0	0	0	0	0
appkey	87,073	56,616	0	0	0	0	0	0	0
asc	6,008	5,169	1	0	0	1	0	0	0
bks	24,399	2,296	2,082	43,421	0	0	185,912	0	0
ca-bundle	3	2	2	6	0	0	0	0	0
cer	80,199	3,563	3,357	8,183	3	2	76,250	32	12
cert	2,338	150	87	78	0	0	95	0	0
crt	14,711	2,842	2,655	52,713	3	2	22,066	21	7
csr	431	274	0	0	0	0	0	0	0
der	18,137	763	715	1,654	603	27	9,079	282	47
dsa	6,076	6,040	0	0	0	0	0	0	0
ec	2	2	0	0	0	0	0	0	0
exe	980,685	241,893	0	0	0	0	0	0	0
gpg	91	89	0	0	0	0	0	0	0
jks	6,615	347	63	325,104	0	0	53,310	0	0
kdb	58	16	0	0	0	0	0	0	0
kdbx	57	20	0	0	0	0	0	0	0
key	9,912	827	206	7	158	95	6	253	227
keystore	1,102	385	17	0	3	0	0	37	0
ovpn	3,417	3,274	3,211	2,369	2	60	1,034	0	32
p12	2,310	717	4	0	0	0	0	1	4
p7b	89	22	14	244	0	0	104	0	0
p7m	13	10	0	0	0	0	0	0	0
p7s	1	1	0	0	0	0	0	0	0
pem	52,447	5,045	4,807	130,326	14,898	236	59,774	12,361	842
pfx	6,340	1,330	1	0	0	0	4	0	0
pgp	5	3	0	0	0	0	0	0	0
pkcs11	1	1	0	0	0	0	0	0	0
pkcs12	34	10	0	0	0	0	0	0	0
ppk	55	40	0	0	0	0	0	0	0
priv	3	1	1	0	0	0	0	0	3
private	34	4	0	0	0	0	0	0	0
pub	6,716	5,468	35	0	64	0	2	24	0
pubkey_pgp	139,184	89,700	0	0	0	0	0	0	0
public	52	8	4	18	2	0	5	0	0
rsa	666,428	632,089	631,852	256,874	1	0	409,837	0	0
sec	582	291	0	0	0	0	0	0	0
seckey_pgp	217,462	145,336	0	0	0	0	0	0	0
sig	23,395	6,416	3	0	5	0	1	0	0
sign	290	166	0	0	0	0	0	0	0
signature	281	68	0	0	0	0	0	0	0
spc	14	10	0	0	0	0	0	0	0
Total	2,376,721	1,216,354	649,117	820,997	15,742	423	817,479	13,011	1,174

References

1. Allix, K., Bissyandé, T.F., Klein, J., Traon, Y.L.: Androzoo: Collecting millions of android apps for the research community. In: 2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR). pp. 468–471 (2016)
2. Barker, E., Chen, L., Roginsky, A., Vassilev, A., Davis, R., Simon, S.: Recommendation for pair-wise key establishment using integer factorization cryptography. Tech. rep., National Institute of Standards and Technology, Gaithersburg, MD (mar 2019), https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-56Br2.pdf
3. Boeyen, S., Santesson, S., Polk, T., Housley, R., Farrell, S., Cooper, D.: Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. RFC 5280 (May 2008). https://doi.org/10.17487/RFC5280, https://www.rfc-editor.org/info/rfc5280
4. Egele, M., Brumley, D., Fratantonio, Y., Kruegel, C.: An empirical study of cryptographic misuse in android applications. In: Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security. p. 73–84. CCS '13, Association for Computing Machinery, New York, NY, USA (2013)
5. Fahl, S., Dechand, S., Perl, H., Fischer, F., Smrcek, J., Smith, M.: Hey, nsa: Stay away from my market! future proofing app markets against powerful attackers. In: Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security. p. 1143–1155. CCS '14, Association for Computing Machinery (2014)
6. Falliere, N., Murchu, L., Chien, E.: W32.Stuxnet Dossier (Nov 2010), https://www.wired.com/images_blogs/threatlevel/2010/11/w32_stuxnet_dossier.pdf
7. Gao, J., Kong, P., Li, L., Bissyandé, T.F., Klein, J.: Negative results on mining crypto-api usage rules in android apps. In: 2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR). pp. 388–398 (2019)
8. Kang, H., wook Jang, J., Mohaisen, A., Kim, H.K.: Androtracker : Creator information based android malware classification system (2014)
9. Kim, D., Kwon, B.J., Dumitraş, T.: Certified Malware: Measuring Breaches of Trust in the Windows Code-Signing PKI. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. p. 1435–1448. CCS '17, Association for Computing Machinery, New York, NY, USA (2017)
10. Li, J., Lin, Z., Caballero, J., Zhang, Y., Gu, D.: K-hunt: Pinpointing insecure cryptographic keys from execution traces. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. p. 412–425. CCS '18, ACM, New York, NY, USA (2018)
11. Li, X., Jiang, J., Benton, S., Xiong, Y., Zhang, L.: A large-scale study on api misuses in the wild. In: 2021 14th IEEE Conference on Software Testing, Verification and Validation (ICST). pp. 241–252. IEEE Computer Society, Los Alamitos, CA, USA (apr 2021)
12. Muslukhov, I., Boshmaf, Y., Beznosov, K.: Source attribution of cryptographic api misuse in android applications. In: Proceedings of the 2018 on Asia Conference on Computer and Communications Security. p. 133–146. ASIACCS '18, Association for Computing Machinery, New York, NY, USA (2018)
13. Piccolboni, L., Guglielmo, G.D., Carloni, L.P., Sethumadhavan, S.: CRYLOGGER: Detecting crypto misuses dynamically. In: 2021 IEEE Symposium on Security and Privacy (SP). IEEE (may 2021)

14. Rahaman, S., Xiao, Y., Afrose, S., Shaon, F., Tian, K., Frantz, M., Kantarcioglu, M., Yao, D.D.: Cryptoguard: High precision detection of cryptographic vulnerabilities in massive-sized java projects. In: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security. p. 2455–2472. CCS '19, Association for Computing Machinery, New York, NY, USA (2019)
15. Research, K.L.G., Team, A.: The duqu 2.0 persistence module (Jun 2015), https://securelist.com/blog/research/70641/the-duqu-2-0-persistence-module/
16. Wickert, A.K., Baumgärtner, L., Schlichtig, M., Narasimhan, K., Mezini, M.: To fix or not to fix: A critical study of crypto-misuses in the wild. In: 2022 IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom). pp. 315–322. IEEE Computer Society, Los Alamitos, CA, USA (dec 2022)
17. Zhang, L., Chen, J., Diao, W., Guo, S., Weng, J., Zhang, K.: CryptoREX: Large-scale analysis of cryptographic misuse in IoT devices. In: 22nd International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2019). pp. 151–164. USENIX Association, Chaoyang District, Beijing (Sep 2019)
18. Zhang, Z., Diao, W., Hu, C., Guo, S., Zuo, C., Li, L.: An empirical study of potentially malicious third-party libraries in android apps. In: Proceedings of the 13th ACM Conference on Security and Privacy in Wireless and Mobile Networks. p. 144–154. WiSec '20, Association for Computing Machinery, New York, NY, USA (2020)