Measuring and characterizing weak RSA keys across PKI ecosystem

A thesis submitted to the
College of Graduate and Postdoctoral Studies
in partial fulfillment of the requirements
for the degree of Master of Science
in the Department of Computer Science
University of Saskatchewan
Saskatoon

By
Fatemeh Nezhadian

Permission to Use

In presenting this thesis in partial fulfillment of the requirements for a Postgraduate degree from the University of Saskatchewan, I agree that the Libraries of this University may make it freely available for inspection. I further agree that permission for copying of this thesis in any manner, in whole or in part, for scholarly purposes may be granted by the professor or professors who supervised my thesis work or, in their absence, by the Head of the Department or the Dean of the College in which my thesis work was done. It is understood that any copying or publication or use of this thesis or parts thereof for financial gain shall not be allowed without my written permission. It is also understood that due recognition shall be given to me and to the University of Saskatchewan in any scholarly use which may be made of any material in my thesis.

Disclaimer

Reference in this thesis to any specific commercial products, process, or service by trade name, trademark, manufacturer, or otherwise, does not constitute or imply its endorsement, recommendation, or favoring by the University of Saskatchewan. The views and opinions of the author expressed herein do not state or reflect those of the University of Saskatchewan, and shall not be used for advertising or product endorsement purposes.

Requests for permission to copy or to make other uses of materials in this thesis in whole or part should be addressed to:

Head of the Department of Computer Science
176 Thorvaldson Building, 110 Science Place
University of Saskatchewan
Saskatoon, Saskatchewan S7N 5C9 Canada

OR

Dean
College of Graduate and Postdoctoral Studies
University of Saskatchewan
116 Thorvaldson Building, 110 Science Place
Saskatoon, Saskatchewan S7N 5C9 Canada

Abstract

The insecurities of public-key infrastructure on the Internet have been the focus of research for over a decade. The extensive presence of broken, weak, and vulnerable cryptographic keys has been repeatedly emphasized by many studies. Analyzing the security implications of cryptographic keys' vulnerabilities, several studies noted the presence of public key reuse. While the phenomenon of private key sharing was extensively studied, the prevalence of public key sharing on the Internet remains largely unknown. This work performs a large-scale analysis of public key reuse within the PKI ecosystem. This study investigates the presence and distribution of duplicate X.509 certificates and reused RSA public keys across a large collection containing over 315 million certificates and over 13 million SSH keys collected over several years. This work analyzes the cryptographic weaknesses of duplicate certificates and reused keys and investigates the reasons and sources of reuse. The results reveal that certificate and key sharing are common and persistent. The findings show over 10 million certificates and 17 million public keys are reused across time and shared between the collections. Observations show keys with non-compliant cryptographic elements stay available for an extended period of time.

The widespread adoption of Android apps has led to increasing concerns about 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. This study performs a detailed analysis of code-signing 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. The results reveal that despite the growing nature of the Android ecosystem, the misuse of cryptographic elements is common and persistent. The findings uncover several issues and enable us to provide a series of applicable solutions to the seen security flaws.

Acknowledgements

I would like to express my gratitude to my advisor, Dr. Natalia Stakhanova. Her sincere support and encouragement propelled me through every phase of crafting this thesis. I will forever cherish the inspiration gained from her mentorship.

I am immensely grateful to my incredible family, whose unwavering love and support have been my steadfast motivation and source of strength throughout every step of my life's journey. Everything I have accomplished is a testament to their belief and support, and I owe all my achievements to them.

Dedicated to those with a passion for making the world a safer place.

Contents

List of Tables

List of Figures

List of Abbreviations

API	Application Programming Interface
APK	Android Application Package
app	application
BKS	Bouncy castle KeyStore
CA	Certification Authority
CRL	Certificate Revocation List
CT	Certificate Transparency
CVE	Common Vulnerabilities and Exposures
DNS	Domain Name System
DROWN	Decrypting RSA with Obsolete and Weakened eNcryption
GCD	Greatest Common Divisor
GPG	GNU Privacy Guard
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
JKS	Java KeyStore
OAuth	Open Authorization
OID	Object Identifier
OS	Operating System
PE	Portable Executable
PGP	Pretty Good Privacy
PKI	Public Key Infrastructure
RNG	Random Number Generator
ROCA	Return of Coppersmith's Attack
RSA	Rivest-Shamir-Adleman
SCADA	Supervisory Control and Data Acquisition
SSH	Secure SHell
SSL	Secure Sockets Layer
TLS	Transport Layer Security

1 Introduction

Cryptography plays a crucial role in modern communications and applications, ensuring the secure and confidential exchange of sensitive information. The underlying system that enables these functionalities 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 Certificates Authority (CA) plays a vital role in this infrastructure issuing certificates to software vendors and attesting to their identity.

In 2021, Github revoked RSA keys, generated by the vulnerable GitKraken client's library that created duplicate SSH authentication keys [29]. In 2018, Infineon released a security patch to update the vulnerable Trusted Platform Module in its microcontrollers responsible for generating vulnerable RSA keys [51].

The sad state of public key infrastructure on the Internet has been the focus of security research for over a decade. In 2011 Holz *et al.* [33] investigating the deployed X.509 certificates for TLS/SSL certification pointed out that infrastructure is broken, i.e., only one out of five certificates can be counted as valid, and many include cryptographically weak keys.

The presence of broken, weak, and vulnerable cryptographic keys on the Internet has been extensively investigated by a number of studies from different points of view. Some studies traced back the problem to weak random number generators and the lack of entropy [15, 21, 31], while others claimed a simple misuse of keys [20], and the improper implementation of cryptographic libraries as the main reasons [7, 18, 43, 50, 66].

Network appliances and services rely on certificates and keys to facilitate secure communication, code signing, authentication, and other security-related functionality. However, mishandling or misuse of certificates or keys poses a significant threat to the overall security of the PKI ecosystem.

Weaknesses in RSA keys allow for faster factorization, i.e., one can efficiently compute the corresponding private keys undermining the security of communication [27, 31]. When keys are reused across different devices and organizations, it exposes the corresponding private key, which can be exploited by attackers. This enables attackers to leverage benign certificates to sign malicious software or impersonate legitimate companies. Consequently, an attacker who gains access to the private key or is able to regenerate it (e.g., in case of vulnerable and weak public keys) can access encrypted content and/or intercept secured network traffic.

Misuse of cryptographic algorithms and a combination of implementation decisions made in software libraries can lead to distinguishable patterns and consequently can be leveraged in identifying a probable origin of a key, e.g., an originating library, its specific version, and operating system [10, 37, 51, 66]. The

public key reuse was initially observed by Heninger *et al.* [31] in a small study of 6.2 million SSH keys and 5.8 million TLS certificates, and later confirmed by Cangialosi [11]. Although the phenomenon of private key sharing with web hosting providers was extensively explored by Cangialosi *et al.* [11] and Liu *et al.* [45], the prevalence of public key reuse across domains remains largely unknown.

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 [23, 60]. The study by Kim *et al.* [41] demonstrated that malware uses valid certificates to evade anti-virus programs and bypass system protection mechanisms. Kang *et al.* [39] 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.

Digital certificates and keys are essential components that enable secure communication, code signing, authentication, secure storage of sensitive information, and other security-related features within hosts and applications. However, the improper use or mishandling of these certificates and keys poses a substantial risk to the overall security of the ecosystem.

1.1 Motivation

This study represents the most extensive measurement analysis of public RSA key reuse on the Internet to date. The collection comprises 84 million unique RSA keys, which is 15 times greater than the set used previously in a seminal study conducted by Heninger *et al.* [31]. This study conducts the lifespan analysis of the use of certificates over time and determines the reused certificates and keys across different devices, organizations, and domains. The analysis shows that 6.5% (10,110,361) of TLS certificates are shared across sets. This work finds a considerable amount of certificates that are reused to sign malware while still being served in communication by different hosts. The observations show that 28% of the certificates used in malicious binaries are also used by Android apps. This emphasizes the existing reuse of keys for different purposes and across domains.

This study investigates the cryptographic characteristics of these reused certificates. Through the analysis, a multitude of security issues is uncovered. These problems include an alarming number of weak and factorable keys, keys susceptible to the ROCA attack [51], and the use of deprecated and non-compliant signature algorithms. To delve deeper into the underlying causes of these vulnerabilities, IPv4 addresses that serve the reused TLS certificates have been scanned. The findings indicate that the majority of reused certificates are associated with embedded network devices running operating systems derived from Linux and BSD distributions. These certificates are primarily utilized by devices manufactured by a small number of companies and have been in active use for an extensive period. For instance, it is observed that 48,794 IP addresses, identified as routers, have been serving 950 distinct certificates for over two decades. Alongside

duplicate certificates, it is discovered 14,862,767 identical RSA public keys that appear in multiple distinct TLS certificates. 44% of these shared keys can be considered weak.

The current study traces the reused keys found in distinct certificates back to their origins. The investigation reveals that these keys are primarily generated by only a couple of versions of OpenSSL and GnuTLS libraries associated with a history of issues related to random number generation implementation.

As a complementary investigation for the observations, a systematic study is conducted to measure and characterize the use of compromised digital certificates by focusing 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, a complementary set of Android apps is collected to extract their corresponding digital certificates. This study analyzes the reuse of these deemed to be compromised certificates among the apps.

The findings show that certificate reuse is more pervasive and widespread than previously observed, 48% of the 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 apps is generally discouraged by Google, it is observed that this practice is commonly ignored by both benign and malicious apps. For example, 59% benign apps in the collections were signed with duplicate certificates. At the same time, it was discovered that 9,931 unique certificates were employed to sign 142,579 malicious and 84,922 benign apps.

While the study highlights the persistent weaknesses and vulnerabilities in certificates and RSA keys within the PKI ecosystem, we have also provided practical solutions to mitigate these issues and enhance the overall security of PKI.

1.2 Contribution

This research study offers the following contributions:

- This study conducts the most comprehensive and the largest Internet-wide scan and analysis of TLS/SSL certificates and RSA keys across domains.

- This study conducts the most comprehensive and largest analysis of cryptographic elements across Android applications.

- This study measures and characterizes the extent of TLS/SSL certificates and RSA public key sharing across domains on a diverse set of over 314 million valid certificates and 13 million SSH keys collected from multiple sources over a period of several years (up to nine years in some cases).

- This study measures and characterizes the extent of code-signing 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).
- The findings emphasize the need for robust mechanisms for the detection and analysis of duplicate and weak certificates and keys. To facilitate this analysis, a correlation platform is implemented to determine duplicate keys and detect TLS certificates and RSA keys' weaknesses. This platform and the resulting duplicate keys are publicly available: `https://key-explorer.com/`.
- To facilitate analysis of cryptographic reuse, the set of reused certificates is made publicly available.
- Based on the analysis, a set of recommendations is provided to help security practitioners improve overall cryptographic security.

1.3 Thesis Structure

This thesis is organized as follows:

- Chapter 1 (Introduction) presents an overview of the research, emphasizing its significance, the driving motivation behind it, and the specific contributions it makes to the scientific field.
- Chapter 2 (Background) offers a concise introduction to fundamental cryptography concepts. Additionally, it provides brief explanations of the protocols, executable files, and Android applications that are central to the discussions within this study.
- Chapter 3 (Related Work) provides a brief literature review that examines several studies focused on network scanning, vulnerabilities in RSA keys, the certificate reuse phenomenon, and misuse of cryptographic APIs.
- Chapter 4 (Methodology) outlines the process of data collection, tools used for parsing, systematic reuse detection, and analysis of reuse as the key steps of the proposed approach.
- Chapter 5 (Measuring the propagation of RSA keys) extensively describes the findings regarding the examination of certificates and RSA keys, along with an analysis of the prevalence of sharing these cryptographic components across various elements of the PKI ecosystem.
- Chapter 6 (Measuring malicious cryptographic reuse in Android applications) provides complementary results regarding the cryptographic reuse phenomenon across malicious binaries and Android apps.
- Chapter 7 (Discussion and Recommendations) delves into the implications of the findings, outlines the observations, and provides recommendations based on the research. Additionally, novel and captivating research prospects are proposed that can utilize the approach to broaden the scope of the study.

2 Background

This chapter presents a brief introduction to fundamental cryptography concepts and provides concise explanations of the communication protocols, Android applications, and executable files crucial to the discussions in this study. This knowledge is crucial for understanding the discussions in this study.

2.1 Overview

The presence of vulnerable and weak cryptographic elements has a number of critical security implications. If a certificate or the corresponding cryptographic key is vulnerable, the encrypted content and/or secured network traffic can be accessible by an insider attacker (or an external attacker with access to a compromised machine inside the network) who has access to the private key or is able to regenerate it. In spite of the strong mathematical foundations of cryptographic algorithms, several studies highlighted weaknesses in practical implementations of cryptographic protocols which rarely relate to the fundamental aspects of the algorithm's theoretic design.

This study focuses on the analysis and measurement of the presence and distribution of cryptographic materials in different host-based sources, malware executable files, and Android apps. This work explores the weak keys, the cryptographic reuse phenomenon, hosting environments of shared certificates and keys, along with malicious cryptographic reuse in Android apps, to this end, first, it is needed to take a look at some general terms.

2.2 Cryptographic Infrastructure

2.2.1 Public Key Infrastructure (PKI)

PKI is a framework of technologies, policies, and procedures that manages digital certificates and cryptographic keys. It is used for securing communication, verifying the identities of users and entities, and ensuring the integrity and confidentiality of data in a networked environment. PKI involves the use of asymmetric cryptography, which utilizes pairs of keys: a public key and a private key. The public key is widely distributed and used for encryption and verifying digital signatures, while the private key is kept secret and is used for decryption and creating digital signatures. Some of the related concepts and functionalities of PKI are:

- *Key pair generation*: A user or entity generates a pair of cryptographic keys – a public key and a

corresponding private key. The public key can be shared openly, while the private key is kept secret.

- *CA*: A trusted third-party organization known as a CA issues digital certificates. These certificates contain the public key and information about the owner, and they are digitally signed by the CA to verify their authenticity.

- *Certificate Distribution*: The digital certificates are distributed to users or entities. When someone wants to communicate securely or verify the identity of another party, they use the recipient's public key from the certificate.

- *Encryption and digital signatures*: Public keys are used for encrypting sensitive information and verifying digital signatures. Data encrypted with a recipient's public key can only be decrypted with their private key, ensuring confidentiality. Digital signatures are created with the sender's private key and can be verified using their public key, ensuring the integrity and authenticity of the sender's message.

- *Revocation and renewal*: Certificates have a limited validity period. If a private key is compromised or lost, the certificate must be revoked. Certificate Revocation Lists (CRLs) or Online Certificate Status Protocol (OCSP) services are used to check the validity status of certificates. Certificates can also be renewed when they expire.

2.2.2 Digital certificates

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 [8] is one of the most widely used standards for digital certificates that, in addition to the public key and owner's identity, contains a period during which the certificate is considered valid, and a digital signature of the issuing CA. In order to verify a certificate, a client needs to obtain a chain of certificates including a presented leaf certificate, intermediate certificates, and finally, a root certificate. In a web's PKI, when a server presents a leaf certificate, it is expected to include the certificate chain as well. A client then can verify each certificate along the chain using the included CAs' digital signatures. In general, certificates can be used for specific purposes listed in their extension fields.

Extensions: The usage or purpose of a certificate can be optionally listed in *Key Usage* or *Extended Key Usage* extension fields in a certificate such as digital signature, key encipherment, data encipherment, key agreement, TLS web server authentication, code signing, email protection, etc. Meanwhile, *Basic Constraints* provides structural information, including whether a certificate can act as a CA. These extensions enable certificate verifiers to assess suitability for cryptographic operations and enforce robust security measures. Starting from 2008, certificate extensions have been categorized as either critical or non-critical. If a certificate-using system encounters critical extensions 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 [8].

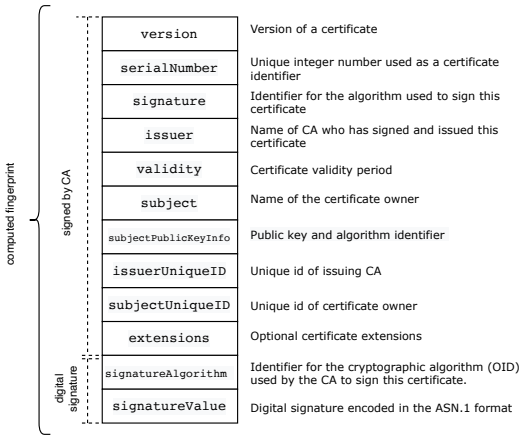

Figure 2.1: X.509 certificate layout.

Signature algorithm: is a cryptographic method used to sign a piece of data and to calculate its hash with a certain hash function. Generally, each public key certificate requires a default signature algorithm from a list of allowed algorithms such as sha256WithRSAEncryption.

An example of an X.509 certificate is given in Figure 2.1. In addition to the certificate layout, it also shows a fingerprint, which is a hash of a certificate in binary format. Typically, digital certificates are compared based on these fingerprints.

Digital certificates can be categorized based on the environment they are used for:

- *TLS/SSL Certificates*: Used to secure communications between web browsers and servers, ensuring data encryption and verifying the authenticity of websites.
- *Code Signing*: Used to verify the authenticity and integrity of software and code.
- *Email Security*: Used for digitally signing and encrypting email messages.
- *Digital Signatures*: Used for signing digital documents to prove their authenticity and integrity.
- *User Authentication*: Used in systems like VPNs or remote access to verify user identities.

2.2.3 RSA algorithm

This work focuses on the RSA keys [38] as this is arguably the most popular cryptographic system utilized on the Internet today. RSA is an asymmetric cryptographic algorithm that leverages the fact that while the multiplication of large prime numbers may not be computationally intensive, the factorization of large prime numbers is significantly more complex. An RSA public key is a pair of values (n, e). It is generated based on two prime numbers p and q used to calculate the modulus n, i.e., $n = p * q$. Here $\phi(n)$ is equal to $(p-1)(q-1)$ and the public key's exponent e is selected at random but it should be between 1 and $\phi(n)$ so that $e \in 1, 2, ..., \phi(n) - 1$ and the e and $\phi(n)$ should be co-prime numbers, therefore, $GCD(e, \phi(n))$ is equal

to 1 so that e and n are relatively prime. As a result, an RSA public key $Pub_k = (e, n)$ is represented by an exponent e and a modulus n. RSA public key size is measured by the length of the key's modulus in bits [53].

2.2.4 Cryptogrphic libraries

These libraries are software packages that include features and methods for secure data protection and communication. They are frequently employed to carry out cryptographic processes such as certificate and key generation, encryption, and decryption. Many different cryptographic methods, key management systems, and protocols are frequently included in these libraries. OpenSSL, GnuTLS, Bouncy Castle, Libsodium, and Crypto++ are a few well-known cryptography libraries.

2.2.5 Keystore

Android provides a hardware keystore facility for managing cryptographic keys. Keystores are utilized for encryption, decryption, TLS/SSL communication, and digital signing within the app, ensuring data integrity and user authentication. These keystores are typically stored in the app's assets or res/raw folder. Different cryptographic frameworks may use formats such as JKS and BKS, with standard extensions including keystore, jks, and bks.

Java KeyStore (JKS) and Bouncy Castle KeyStore (BKS)

A JKS is a standard format for securely storing cryptographic keys and certificates in Java applications. Protected with a password, JKS files hold private keys and their corresponding public key certificates. BKS is an alternative format offered by the Bouncy Castle library, providing a wider range of algorithms and cryptographic capabilities for Java-based applications.

Pretty Good Privacy (PGP) and GNU Privacy Guard (GPG)

PGP is a 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.

2.3 Communication protocols

2.3.1 TLS/SSL

Transport Layer Security (TLS) [59] and its predecessor Secure Sockets Layer (SSL) are cryptographic protocols that secure point-to-point communication over networks to a number of application layer protocols such as HTTPS, DNS, SMTPS, IMAPS, etc. A TLS/SSL connection is initiated with a TLS handshake when a

client requests a secure connection. At this stage, a server presents its digital certificate that allows it to authenticate it to the client. Further verification of the server's identity is performed by a client.

2.3.2 Secure Shell (SSH)

SSH protocol [46] is a cryptographic communication protocol that establishes secure remote access to computers and servers over a network. The SSH host runs an SSH server, which listens for incoming SSH connection requests from clients to start a secure session by exchanging the supported cryptographic algorithms available to both parties and the server then sends its public key to the client to authenticate its identity.

2.4 Android Package Kit (APK)

An APK 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:

1. *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.[1] 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.

   To achieve this, Android supports different APK signature schemes, such as v1 (JAR signing), v2 (introduced in Android 7.0), v3 (introduced in Android 9), and v4 (introduced in Android 11). Android signature mechanism is backward-compatible, and a v4 signature requires a complementary v2 or v3 signature (Figure 2.2).

2. *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.

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

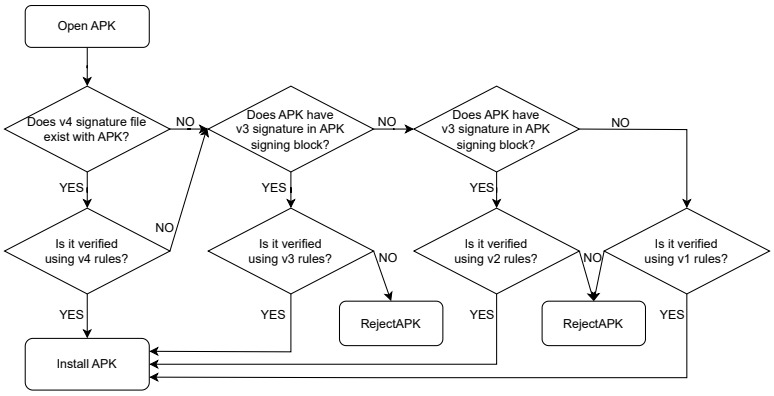

Figure 2.2: APK validation process [56]

3. *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.

2.5 Portable Executable (PE)

Like APKs, other software applications, including executable files, utilize digital certificates and cryptographic keys to ensure 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. Authenticode identifies the publisher of Authenticode-signed software and verifies that the software has not been tampered with since it was signed and published. SignTool, on the other hand, allows developers to sign their applications with a digital certificate to ensure their authenticity and integrity. Then, this digital signature can be verified by Windows during installation or execution, providing users with details of file origins.

3 Related Work

This chapter provides a concise overview of previous related works, which have been categorized into network scanning, weak keys, reused certificates and keys, and misuse of cryptographic APIs subsections. Examining the breadth of research in these areas makes this study gain valuable insights into the magnitude of the problem and the potential solutions that can be employed to safeguard the PKI ecosystem.

3.1 Overview

The study centers around digital certificates and keys derived from communication protocols and file sources. Its focus is on TLS/SSL and SSH protocols while the file collection is based on malicious PE binaries and Android application packages that use RSA keys. Cryptographic components, such as certificates and keys, are essential for ensuring the confidentiality, integrity, and authenticity of data and communication. However, these components are not immune to compromises. The vulnerabilities arise due to various factors, including errors in implementation, misconfigurations, mishandling, or improper parameter choices during their deployment.

To gather comprehensive data, this study utilized large-scale network scanning. This approach allowed the study not only to collect the necessary information but also to gain valuable insights into the hosting environments of these certificates and keys.

3.2 Network scanning

Over the past years, Internet-wide scanning has become a widely adopted practice, extensively utilized by the security, networking, and Internet measurement communities. Its effectiveness in uncovering vulnerabilities in cryptographic protocols like TLS, SSH, SMTP, Web PKI, and IoT devices has been demonstrated in several academic works. This type of scanning has become a common practice, providing valuable data that aids in understanding the prevalence and distribution of cryptographic components in diverse hosting environments. Within the scope of the project, it also explores prior research that leveraged large-scale network scanning techniques to identify and collect data pertaining to digital certificates and keys. Reviewing these works aims to gain a comprehensive insight into the significance of network scanning and its role in the research context.

Several works attempted to introduce network scanning tools for Internet-wide studies among which ZMap [19] is one the most popular optimized to perform fast network scans on entire IPv4 addresses. In

2016, Khattak *et al.* [40] used a network scanning method to investigate censorship issues. They analyzed 1,727,138 pairs of HTTP requests and responses. These pairs consist of requests from users who make use of Tor, a traffic-routing browser bundle for enhanced privacy, alongside non-Tor users. They assessed Tor filtering by employing ZMap probing, conducting tests from both Tor exit nodes and control (non-Tor) nodes to observe variations in their access to remote addresses. The researchers measured that at least 1.3 million addresses in the IPv4 address space along with 3.67% of the Alexa top 1,000 websites either block or offer degraded service to Tor users.

Mirian *et al.* [49] analyzed the security of industrial control systems (IDS) by examining devices exposed on the public Internet by implementing network scans on the IPv4 address space. They found over 60,000 vulnerable devices using the SCADA protocols, which are a set of communication standards used to monitor and control the communication between hardware and software components in IDS. These devices were originally intended for closed and serial systems but then opened to support long-distance communication.

Aviram *et al.* [3] presented DROWN [1] (Decrypting RSA using Obsolete and Weakened Encryption) which is a cross-protocol attack on TLS that uses a server supporting SSLv2 to decrypt TLS connections. They showed DROWN attack can decrypt a TLS 1.2 handshake using 2048-bit RSA. By performing Internet-wide IPv4 scans using ZMap on eight different ports (HTTPS, SMTP, POP3, IMAP, SMTPS, SMTP, IMAPS, POP3S), they investigated the security level of TLS connection which uses a server supporting SSLv2. Attempting three complete handshakes with different versions of TLS/SSL protocol in each connection, they found that 33% of all HTTPS servers and 22% of those with browser-trusted certificates are vulnerable to the protocol-level attack due to widespread key and certificate reuse for different protocol versions.

Amann *et al.* [2] investigated technologies applied to strengthen and improve the TLS and the web PKI such as Certificate Transparency (CT) which makes the CA system auditable; HTTP-based extensions that harden the HTTPS posture of a domain; SCSV downgrade prevention which protects protocol downgrade; and DNS-based extensions which control the certificate issuance and pinning. They combined the active and passive measurements over the Internet. In the case of active measurement, they targeted 193 million domain names instead of active IP address scanning in order to reduce bias from accidentally connected embedded devices hosting web servers. Among a total of 193 million registered domain names, merely 28.4 million domains (14.7%) yielded successful HTTP responses with a status code of 200. They then were able to extract around 12 million certificates corresponding to these domains. Their study revealed a correlation between the ease of deployment, risks to website availability, and the implementation status of technologies. Easily deployable and low-risk technologies like CT and SCSV showed higher adoption rates, whereas technologies demanding more effort or carrying higher risks tended to have lower adoption rates.

[1] https://cve.mitre.org/cgi-bin/cvename.cgi?name=cve-2016-0800

3.3 Weak keys

In real-world scenarios, the reliability of an encryption algorithm relies on the strength of the cryptographic key. However, vulnerabilities in the generation or management of RSA keys can result in the presence of weak RSA keys, posing a significant risk to the overall security of the encryption system. Weak RSA keys are keys that are susceptible to various cryptographic vulnerabilities. To address this issue, researchers have dedicated efforts to understand, identify, and mitigate the impact of weak RSA keys. This subsection reviews a collection of related works and studies conducted in this domain, exploring the techniques utilized to detect and analyze weak RSA keys. Focusing on studies that have concentrated on identifying cryptographic keys with inherent vulnerabilities aims to assess the security risks associated with specific cryptographic algorithms and pave the way for the development of stronger alternatives to enhance encryption system security effectively.

In 2012, Heninger *et al.* [31] studied the presence of vulnerable keys across the Internet by analyzing 6.2 million SSH keys and 5.8 million TLS certificates collected in the wild. Their results showed at least 5.57% of TLS hosts and 9.6% of SSH hosts used vulnerable duplicate keys, underscoring the critical need to address these widespread security risks. Similar results were obtained by Lenstra *et al.* [42] on the analysis of 11.7 million public RSA keys. The later study performed by Gasser *et al.* [27] on 56.4 million SSH keys confirmed that the amount of vulnerable keys is declining. While Heninger *et al.* [31] were able to factor keys for 0.03% of SSH hosts, Gasser *et al.* [27] found that 0.013%-0.016% SSH hosts use factorable keys.

Later in 2013, Durumeric *et al.* [19] evaluated the HTTPS ecosystem against vulnerabilities of RSA keys by performing 110 Internet-wide scans over 14 months. They specifically focused on Debian weak key vulnerability [6] and weak and shared public keys. They found that 44,600 unique certificates corresponding to factorable RSA keys served on 51,000 hosts, which showed a 20% decrease compared to previously reported statistics by Heninger *et al.* [31] but still is more than 25% of HTTPS servers supporting these weak keys.

Several other studies examined the weaknesses of TLS certificates. In 2016, Chung *et al.* [14] showed that on average, 65% of SSL certificates advertised in each IPv4 scan are invalid. Using a set of over 80 million certificates, they found that most invalid certificates originated from a few types of end-user devices associated with a few Autonomous Systems. A concurrent study by Samarasinghe *et al.* [61], albeit on a smaller scale, analyzed certificates obtained from 299,858 devices. Similar to others, the researchers found a presence of small keys (4% were 512-bit and 768-bit keys) and a use of deprecated RC4 stream cipher (37%).

In 2021, Hue et al. [35] assessed TLS authentication on 3,637 domains in a WPA2-Enterprise ecosystem with X.509 certificates. They found security issues like weak algorithms, expired certificates, and possible key reuse. They reported that 5.9% of leaf certificates had short RSA moduli, and 12% used the SHA1-RSA algorithm.

Several studies showed vulnerabilities of cryptographic keys in both protocols and files are due to improper use of libraries or inherited from their weaknesses, for example, random number generators (RNGs). Heninger

et al. [31] confirmed that limited sources for generating appropriate randomness in memory-constrained devices (such as routers, and smart cards). In 2008, a bug in the OpenSSL library made predictable generated random numbers. A follow-up by Yilek *et al.* [69] confirmed the spread of keys affected by the bug even after six months of disclosure. Slow industry response to the cryptographic bugs was also noted by Hastings *et al.* [30] which suggests that one of the main causes behind the majority of factored RSA keys may be related to RNG issues.

The various weaknesses in cryptographic keys were later used to attribute keys to the corresponding libraries that generated them. Svenda *et al.* [66] performed the technical analysis of over 60 million newly generated keys from 22 open and closed-source libraries and from 16 distinct smart card vendors, revealing that various security lapses allow attributing keys to the libraries that generated them based solely on the properties of RSA public keys.

Branca *et al.* [10] took a step further showing that it is feasible to accurately (with 95% accuracy) attribute RSA keys to individual library versions based on their moduli. By analyzing over 6.5 million keys generated by 43 cryptographic library versions they found that with just a few moduli characteristics, an individual key can be attributed to the specific library that created it. They also explored the attribution of SSH keys from publicly facing IPv4 addresses and by 95% accuracy distinguished corresponding libraries of RSA keys.

3.4 Reused certificates and keys

In the realm of web PKI, the phenomenon of certificate and key reuse has emerged as a legitimate scenario, but it also raises concerns about potential vulnerabilities that could be exploited by attackers to compromise security measures. This paper focuses on exploring the technical aspects related to reused certificates and keys. To gain deeper insights into the proposed strategies and detection mechanisms, existing literature and research are reviewed in this subsection. Investigating research efforts that have explored the reuse of cryptographic components across various systems or services reveals the severe security implications, as compromising one instance could lead to unauthorized access to multiple entities. Examining these findings aims to enhance the understanding of the risks associated with certificate and key reuse and identify potential solutions to mitigate such security threats effectively.

Host-based environments

In parallel to their other study, Durumeric *et al.* [17] reported an unexpected observation using their previously collected data. They attempted to collect the corresponding certificate chain as well as the leaf X.509 certificate, after completion of each TLS handshake. They showed out of 1,832 browser-trusted signing certificates, 380 certificates shared their public key with another browser-trusted certificate, forming 136 groups of "sibling" CA certificates.

In 2015, the Shodan company [54], which is a search engine for Internet-connected devices, identified

250,000 routers and 150,000 networking equipment devices, primarily located in Spain, Taiwan, and China, that possessed identical SSH keys. These keys appear to be pre-configured with a single operating system image before deployment. This phenomenon makes these devices open to attackers who discover the corresponding SSH private key.

Felsch *et al.* [24] demonstrated that the key reuse in IPSec protocol can lead to cross-protocol authentication bypasses since in practice just a single RSA key pair together with an encryption and signing certificate is used to configure different versions of the Internet Key Exchange (IKE) protocol. They observed this case in network equipment such as Clavister and ZyXEL devices. They also showed that their Huawei test device reuses its RSA key pair even for SSH host identification, which further exposes such key pair.

Moreover, Holz *et al.* [32] investigated the security of standard Internet messaging infrastructure on protocols of electronic mail (SMTP, IMAP, POP3) and instant chat (XMPP, IRC). They explored the key and certificate reuse in messaging ecosystem and compared it to their similar prior work on the web PKI [33]. They showed that out of encountered 6,398 certificates, 1,096 (17%) were seen on more than one IP address. They also investigated the reuse of self-signed certificates and explained if these certificates are created purposefully for a single server or service, they should not occur on too many hosts. Hence, they assumed that these were default certificates shipped with the software.

Further looking at the shared TLS certificates, Costin *et al.* [15] analyzing firmware images were able to extract 109 private RSA keys from 428 firmware images enabling them to identify approximately 35,000 active online devices on the Internet that were utilizing the same self-signed certificate. They also employed the fingerprint matching of SSL certificates and found that embedded certificates of devices were reused interchangeably and even among different vendors.

Springall *et al.* [64] conducted an analysis of the security of the FTPS protocol and revealed that only 793,000 unique certificates were utilized among the 3.4 million servers supporting FTPS. This pattern was attributed to hosting providers using shared SSL certificates and embedded device manufacturers deploying identical certificates across all their devices.

The study by Frustaci *et al.* [25] investigated the "trust ecosystem" of social IoT devices with the aim of highlighting critical security and privacy concerns. The researchers provided a taxonomic analysis based on the three key layers of the IoT system model: perception, transportation, and application levels. Within the perception layer, they identified shared cryptographic keys as the primary security risk.

Izhikevich *et al.* [36] identified server certificates belonging to IoT interfaces on unexpected ports and showed TLS services on unassigned ports are 1.17 times more likely to have a certificate with a known private key than on assigned ports, highlighting a concerning security issue. Their scan of unassigned ports showed that over twice as many certificates have a known private key compared to the previously reported spread of cryptographic elements investigated by Hastings *et al.* [30] and Heninger *et al.* [31]. They found that on 23% of scanned ports, public keys are more commonly shared on ports other than the expected port 443. In the case of TLS hosts using wireless devices, they observed that 40% of these hosts on port 8081 were

using the same OpenSSL Test Certificate and 39% of them on port 58000 were utilizing the same self-signed certificate, all with a known private key. Their result demonstrated that a significant fraction of services are located on non-standard ports which frequently exhibit weaker security measures compared to those on standard ports.

Files-based environments

The implication of shared certificates in digitally signed malware has been studied by Kim *et al.* [41]. Their introduced threat model centers on the vulnerabilities found within the code-signing PKI. These techniques allowed them to study threats that breach the trust encoded in the Windows code-signing PKI.

They showed that to evade anti-virus programs and bypass system protection mechanisms malware is often signed with valid certificates. They found that 88.8% of the signed malware families rely on a single certificate. Additionally, their analysis revealed that out of 153,853 signed malware samples over 40% were found to contain malformed digital signatures and reused certificates. This enables adversaries to employ the Microsoft Authenticode, which is the prevalent code signing standard on Windows, a signature from a legitimate sample to bypass antivirus detection.

Kang *et al.* [39] introduced AndroTracker, a tool that makes use of serial numbers of certificates to detect possible Android malware. Generally, for distributing an application, the creator signs it with their private key and its corresponding certificate. In this study, they tracked the reuse of certificates among 55,000 malicious and benign applications. They found that 70% of the malicious applications are using 4% of the total number of signing certificates used for the overall applications.

In 2023, Hageman *et al.* [28] conducted a study by crawling various markets, including Google Play Store, APKMonk, APKMirror, Baidu, and Tencent. They discovered that Android applications developed by different authors often share signing certificates. This phenomenon was attributed to the increasing use of app-building frameworks and software developers, which likely contributed to the generation of shared certificates among developers. Their analysis of certificate reuse revealed that a small portion of certificates (ranging from 1.2% on Google Play to 4.9% on Tencent) are used by multiple developer names. Despite their limited number, these certificates account for a significant proportion of market entries, ranging from 15.2% on Google Play to 22.6% on Tencent.

3.5 Misuse of cryptographic APIs

In the context of software development, cryptographic Application Programming Interfaces (APIs) play a key role in ensuring secure communication, data protection, and authentication. However, the improper usage or implementation of these APIs can pose a significant threat to system integrity and lead to security implications. Over the past decade, researchers have conducted several studies to explore the misuse of cryptographic APIs in Android applications, shedding light on potential vulnerabilities and challenges in

this domain. By analyzing their findings and methodologies, valuable insights are gained into the risks associated with the improper utilization or misconfiguration of cryptographic APIs. Understanding these cases is essential to analyze the overall security of the PKI ecosystem and to develop best practices for secure API implementation in order to safeguard sensitive data effectively.

The vast majority of the prior approaches use verification-based analysis that offers assurances for the correct implementation of cryptographic primitives. For example, the static analysis frameworks CryptoLint [20] and BinSight [50] perform analysis of cryptographic misuse at scale.

By focusing on the Google Play marketplace, CryptoLint [20] found that a significant number of Android applications, specifically 88% out of 11,748 analyzed, had at least one mistake in their usage of cryptographic APIs. The researchers specifically explored the utilization of cryptographic primitives through static analysis of symmetric encryption and its associated securing rules. The findings outlined that many applications did not adhere to best practices and guidelines when employing cryptographic APIs, which consequently compromised the overall security of these applications.

BinSight [50] 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 at least one call-site to Java cryptographic API guidelines. Researchers found that the use of deprecated cipher modes decreased over the years, but libraries increasingly relied on static encryption keys. However, applications did not show much interest in adopting static keys.

Similarly, Gao *et al.* [26] found that 96% of the analyzed 8 million APKs from the AndroZoo [1] dataset exhibited some crypto-API misuses. They presented the distribution of cryptographic misuse in APKs, indicating that most applications typically have between 2 to 9 instances of misuse cases. They also investigated the update rate at the crypto-API usage in newer versions of 600,000 applications and discovered that over 95% of the misused cases are not touched by app developers during the evolution of Android apps.

K-Hunt [43] focused on weak and insecure cryptographic keys by analyzing executable binaries and leveraging the properties of crypto operations. It identifies the memory buffers where crypto keys are stored, tracks their origin and propagation, and identifies insecure keys, including deterministically generated keys, insecurely negotiated keys, and recoverable keys.

Following the same idea, CRYLOGGER [55] 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, and weak passwords). The analysis of 1,780 Android apps from the Google Play Store showed that crypto misuses can be detected dynamically by supporting a large number of rules.

Wickert *et al.* [68] focused on the severity of crypto misuses especially in 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 and nearly half the misuses (42.78%) are of high severity. For example, 29.05% of the applications continue to utilize SSL, TLS 1.0, or

TLS 1.1, which could potentially make them vulnerable to remote exploitation.

Numerous studies explored more generic detection of cryptographic misuses. Li *et al.* [44] conducted a large-scale analysis of API misuse based on bug-fixing commits from GitHub projects, 51.7% of which were related to API misuses. Zhang *et al.* [71] proposed LibExtractor to detect potentially malicious libraries and identify malware families based on their digital certificates. They observed 99.65% of the 206,572 certificates collected from malware samples are self-signed.

CryptoGuard [57] detected cryptographic misuse in Java programs including 6,181 Android apps. Similar to other studies, CryptoGuard discovered that around 95% of discovered vulnerabilities, such as hardcoded and predictable passwords, originate from libraries that are packaged with the application code.

Zhang *et al.* [70] have proposed CryptoREX, a framework to identify cryptographic misuse of IoT devices. Analyzing 521 firmware images with 165 pre-defined cryptographic APIs, CryptoREX showed that 24.2% of firmware images violate at least one misuse rule.

Braga *et al.* [9] conducted an analysis of cryptography misuse in cryptography-based security and cryptographic programming using contributions from software developers on online forums. Their study employed a data mining technique to identify the impact of complex architectures on developers, leading to confusion and the persistence of recurring errors in cryptographic programming.

Chatzikonstantinou *et al.* [12] conducted an assessment of the cryptographic security of 49 Android applications. They categorized cryptographic weaknesses into four groups: use of weak cryptography, weak implementations, use of weak keys, and use of weak cryptographic parameters. The study revealed that nearly 88% of the applications exhibited some form of misuse, while the remaining applications did not utilize cryptography during the analysis.

3.6 The perspective

Similar to previous research, this study also examines weaknesses in RSA public keys. However, unlike studies that concentrate on a specific area of RSA key usage, the study investigates the prevalence of shared certificates and key usage across diverse domains on a much larger scale.

The approach goes beyond only relying on real-time data collection and active network scanning; additionally, a substantial dataset gathered over the years is leveraged, covering more than 314 million certificates and 13 million SSH keys. This extensive dataset offers a more comprehensive and macroscopic view of key usage on the Internet, allowing the approach to highlight the persistence of specific vulnerabilities and implementation errors over time. By adopting this approach, this work can provide valuable insights into the long-term implications of cryptographic practices and better understand the security landscape of cryptographic components.

This study undertakes a comprehensive investigation into the properties of certificates and RSA keys, along with the environment where shared cryptographic elements are hosted. Additionally, exploring the

libraries responsible for generating vulnerable keys is performed to understand the origins and implications of potential security weaknesses.

Moreover, the existing approaches within the Android ecosystem primarily concentrate on examining whether cryptographic functionality implemented by the cryptographic libraries embedded in Android is correct and how it has evolved over time, with their primary focus on benign applications. In contrast, the investigation focuses on the reuse of certificates, including malicious certificates.

Building on the pioneering work of Kim *et al.* [41], which explored the presence of PE files signed by malicious certificates, this study extends this analysis further. The research delves into the scope and attributes of malicious certificates across domains encompassing Windows binaries and Android apps, including both malicious and benign applications. By broadening the scope of the analysis, this work aims to gain a comprehensive understanding of the prevalence and implications of certificate reuse in different contexts, shedding light on potential security risks and challenges in cryptographic practices across various platforms.

4 Methodology

This chapter outlines the planned approach for carrying out this research. Our focus lies in gathering data from various sources, extracting certificates and keys, thoroughly analyzing how they are reused in the PKI ecosystem, and specifically investigating instances of malicious reuse within the context of Android.

4.1 Overview

The flow of our study is illustrated in Figure 4.1, comprising four key steps: data collection, parsing, reuse detection, and analysis of reuse.

Initially, we gather data from various sources, ensuring a diverse and comprehensive dataset. Next, the collected data undergoes a meticulous parsing process, extracting critical cryptographic information, primarily certificates, and keys. Subsequently, we conduct a systematic reuse detection process to identify instances of cryptographic component reuse within the PKI ecosystem and pinpoint potential security vulnerabilities. Lastly, our analysis extends to in-depth investigations of malicious reuse within the Android context, providing valuable insights into the security implications of cryptographic component reuse in this specific domain.

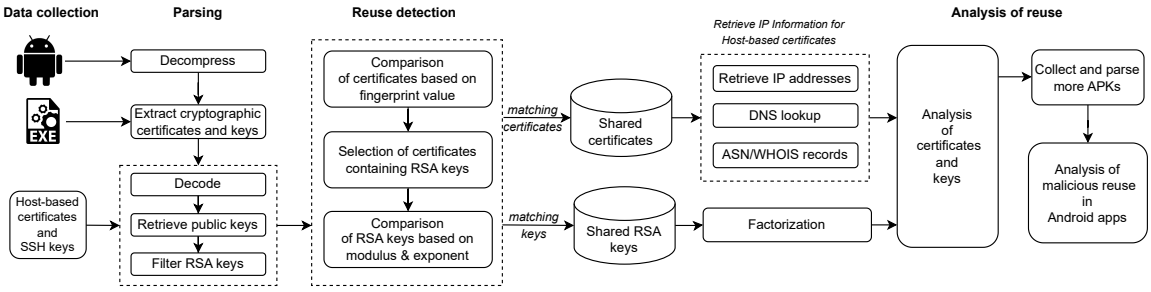

Figure 4.1: The flow of the analysis.

4.2 Data Collection

To ensure the completeness of our measurement study, digital certificates and RSA keys were collected from six different sources, covering a period of several years. It is included host-based repositories namely TLS/SSL certificates, the Rapid7 dataset, the SBA set, and SSH keys among these sources. Additionally, we incorporated file-based collections, namely certificates and keys from malware binaries and Android applications.

During the data collection process, there was no overlap between collected TLS/SSL, Rapid7, and SBA datasets in terms of IP addresses and protocols tested per port.

4.2.1 TLS/SSL certificates

To collect TLS/SSL certificates, 10 million series of pseudo-random integers are generated and interpreted as IPv4 addresses. This ensured a uniform distribution of hosts. These IP addresses were contacted on ports 21 (FTP), 443 (HTTPS), 465 (SMTPS), 993 (IMAPS), and 995 (POP3) to determine the listening hosts. If a host replied, acknowledging our connection (syn-ack packet), we further connected to it using TLS/SSL protocol to obtain a certificate. Several scans in 2013 have been made for collecting certificates using SSL2, SSL3, SSL23[1], TLS 1, TLS 1.1 protocols. For each successful connection of TLS/SSL, we collected a main certificate and discarded the chain of certificates.

4.2.2 Rapid7 certificates

An additional collection of SSL certificates is obtained from Rapid7.[2] It includes weekly scans collected by Rapid7 from October 2013 to September 2015, during August 2019, and during the period from September 2020 to July 2021. In addition to X.509 certificates, this retrieved set contains a collection of metadata, e.g., time of collection, IP address, protocol and port, and the X.509 certificate's fingerprint. Among all certificates only the leaf certificate were kept and the chain of certificates were discarded. The certificates from 2013 to 2015 were retrieved from Rapid7 in the past without the corresponding IP addresses. Note that this set is no longer available from the Rapid7 set in its entirety.

4.2.3 SBA certificates

In this analysis, the subset of certificates offered for analysis by Mayer *et al.* [47] is also used. The email ecosystem makes wide use of certificates for the secure transmission of messages. The SBA set contains the certificates retrieved over StartTLS protocol from SMTP servers on port 25 during a period of April 2015 - August 2015.[3]

4.2.4 SSH keys

To collect SSH keys, the public IPv4 space is scanned during May-September, 2021 by utilizing ZMap to perform a single-packet host discovery. Each host discovered by ZMap is contacted again on ports 22, 23, 2222, 4444, 5000, and 10001 using the ssh-keyscan tool[4] collecting SSH banner and their public SSH RSA

[1]The OpenSSL header file dealing with the combined use of the SSLv2 and SSLv3 protocols
[2]https://opendata.rapid7.com/sonar.ssl/
[3]The original set is now only available through Wayback machine: https://web.archive.org/web/20160307162719/https://scans.io/study/sba-email. Due to a site problem, we were never able to download the whole set.
[4]https://man.openbsd.org/ssh-keyscan.1

key. In addition to the cryptographic material from RSA keys, key collection time, an IP address, and a port are recorded. Using ssh-keyscan, both the public key and server header from the connection were collected.

4.2.5 Malware certificates

To explore certificates from binaries, a set of 40,270,387 malicious files in PE (Microsoft Windows Portable Executable) format were collected from VX underground's APT collection[5] and VirusShare repository[6] from 2012 to 2021.

4.2.6 APKs certificates

- *Original set:* The data collection efforts were extended by crawling several legitimate Android markets, namely Google Play Store, 1Mobile Market, SlideME, Xiaomi, Nduo.cn, Mob.org, Anzhi, F-Droid, and ApkGalaxy, to collect APK files. This extensive data collection process spanned from September 2020 to October 2020.

- *Complementary set:* The main analysis results encouraged the approach to collect more APKs for further investigation on malicious reuse of cryptographic elements. To ensure a robust dataset, Android APK files from several Android app distribution platforms were collected including Google's official market called Google Play Store, and Chinese app stores primarily focused on Chinese users such as Xiaomi and Anzhi; free open-source repositories for Android apps such as F-Droid and AndroidAPKs-Free; 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. Also apps from malware repositories VirusShare and VirusTotal were collected. By utilizing a combination of these sources, this study aimed to gather a representative sample of Android APK files, encompassing benign used by different categories of users and malicious apps. The complementary set included benign applications collected between 2017 and 2023 and thus presumably following the most recent standards and malicious apps from 2012 to 2022.

4.3 Parsing

The Android apps were unpacked to obtain and verify 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.[7] Then, we analyzed for the presence of cryptographic keys, i.e., files with standard extensions such as rsa, pem, crt, and cer. These certificate files are typically used to sign APK files,

[5]https://vx-underground.org/apts.html
[6]https://virusshare.com
[7]https://github.com/itomsu/apk_parse3

i.e., to identify the owner of an Android app. We were primarily interested in keys used for code signing, hence, we did not decompile .dex files to extract hard-coded keys.

Malware binaries were parsed using GoLang's sigtool[8], which is a PE package designed to extract digital signatures from signed PE files. The certificates used for signing code are contained in the "Attribute certificate" section of PE files in DER format.

To maintain consistency, we converted the extracted certificates and keys from APKs and binaries into the PEM format. Subsequently, along with the host-based certificates and keys, we parsed and extracted the cryptographic information from all these sources using Python cryptographic libraries namely PyOpenSSL, Cryptography, Pyasn1, PyJKS, and Paramiko; keytool[9] and CERT Keyfinder[10].

While parsing data from our sets, we found that a significant number of IP addresses presented invalid or incomplete certificates. Furthermore, not all the certificates and keys collected from files are parsable due to corrupted formats, password protections, or outdated (no longer supported by libraries) standards.

We thus discarded all invalid, incomplete, or unparsable certificates and keys. Among the collected keys, approximately 97.8% were RSA keys. As a result, we further filtered non-RSA keys. For our analysis, we retained RSA public keys, i.e., their modulus and exponent, the information extracted from their corresponding certificates, and the IP addresses (when available).

4.4 Reuse detection

To determine the potential key reuse across our sets, we proceed to perform a pairwise comparison of RSA certificates based on their fingerprints (also referred to as thumbprints), i.e., the SHA-1 hash of the certificate in DER binary format. In addition to matching certificates, we perform a pairwise comparison of valid RSA keys from all sets irrespective of whether their corresponding certificates are shared or not.

For shared certificates, we retrieve available IP addresses. Since the Rapid7 scans from 2013 to 2015 had no associated IP address or port information, further analysis of the certificates was performed based on the IP information for the matching certificates retrieved from the other sets. For each of these IP addresses, we performed a DNS query to retrieve the corresponding PTR records, which provide us with a domain name associated with an IP address. We also map an IP to ASN and retrieve WHOIS contacts using the Team Cymru service.[11]

4.5 Analysis of reuse

In the next step of the analysis, our focus shifts exclusively to shared certificates and RSA public keys. We thoroughly examined shared certificates from multiple perspectives, including the reuse of certificates

[8] https://github.com/doowon/sigtool
[9] https://docs.oracle.com/javase/8/docs/technotes/tools/unix/keytool.html
[10] https://github.com/CERTCC/keyfinder
[11] https://team-cymru.com/community-services/ip-asn-mapping/

over time and for different purposes. Additionally, we investigated signature algorithms and explored the environments where these certificates were utilized. Furthermore, we conducted an in-depth investigation of key reuse, particularly vulnerable keys.

The observations during shared certificates analysis encouraged us to expand our approach and collect a larger set for APKs, we refer to them as *complementary* set, to measure and characterize the extent of compromised certificates and key reuse across Android apps and malware binaries. To this end, we analyzed a broader collection of Android apps and for each of the apps, we extracted all cryptographic elements such as digital certificates, and public and private keys that were included in the Android application packages.

During the process of unpacking the APKs, 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.

In order to assess the reuse within the complementary data, we conducted similar reuse detection with 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.

In the context of the experimental analysis, it is essential to highlight that handling and processing the extensive dataset pose significant challenges. Parsing such large volumes of data is a time-consuming operation that demands substantial memory resources, making it resource-intensive. To address this, we developed custom parsers to efficiently parse the certificates and RSA keys. Once parsed, all the data, which amounts to over 4 terabytes, was loaded into a PostgreSQL 14.9[12] database.

Querying such a substantial amount of data comes with a high computational cost. PostgreSQL uses an arbitrary unit to measure the cost of queries, and for instance, one of our primary queries, which involves pairwise comparisons to identify reused certificates, is estimated to execute a sequence of 14 functions. These functions include grouping, aggregation, filtering, gathering of merged results, various sorting operations, and sequential scans, all of which incur significant computational expenses.

All our parsers are implemented in Python and are publicly available in our GitHub repository: `https://github.com/thecyberlab/RSA-keys-analysis`.

4.6 Summary

In the methodology section, our research approach involved a series of well-defined steps. Firstly, we collected certificates and keys from six diverse sources, ensuring a comprehensive and representative dataset for analysis. Next, the collected data, comprising Android applications, malware binaries, and host-based certificates, underwent thorough parsing and processing to extract essential cryptographic information, laying

[12]PostgreSQL 14.9 (Ubuntu 14.9-0ubuntu0.22.04.1) on x86_64-pc-linux-gnu, compiled by gcc (Ubuntu 11.4.0-1ubuntu1~22.04) 11.4.0, 64-bit

the groundwork for subsequent analyses.

Subsequently, we employed a systematic reuse detection process to identify instances of shared certificates and RSA keys, allowing us to gain insights into the prevalence and implications of cryptographic component reuse. The identified shared certificates and RSA keys were then subjected to in-depth analysis, enabling us to make observations and uncover practices that informed our approach's direction. These observations further motivated us to explore the phenomenon of malicious cryptographic reuse within the Android ecosystem, leading to a deeper understanding of potential security risks and vulnerabilities in cryptographic practices on the platform.

5 Measuring the propagation of RSA keys

This chapter conducts a comprehensive examination of certificates and RSA keys along with an analysis of the prevalence of sharing of these cryptographic components across various components of the PKI ecosystem, including hosting environments, executable files, and Android APKs.

5.1 Overview

The propagation of certificates and RSA keys across the PKI ecosystem is a less-explored phenomenon raising concerns about the efficacy of existing security protocols. We endeavor to identify the reasons behind this duplication, and regardless of the cause, the presence of duplicates raises red flags, casting doubt on the overall security of the PKI ecosystem.

Section 5.2 gives details of our collected data from various sources. In section 5.3 the general findings about weak and vulnerable keys along with the presence of legacy certificates are discussed. Section 5.4 reviews the presence of the shared certificates by taking a deeper look into their characteristics as well as hosting environments. Section 5.5 takes a deeper look at the phenomenon of shared RSA keys in distinct certificates by measuring their weakness and vulnerabilities along with exploring the libraries that are responsible for generating such keys. Finally, section 5.6 summarizes our findings in this section.

5.2 Collected Data

The details of the collected data are provided in Table 5.1. The dataset comprises data from six various sources, including 863,872 TLS/SSL certificates from a set of generated specific IP addresses, 295,063,780 certificates from the Rapid7 dataset, and 202,381 certificates from the SBA dataset, resulting in 296,130,033 certificates from host-based sources along with an additional 13,681,145 SSH keys gathered from 10,435,118 IP addresses.

Additionally, we gathered a substantial dataset of 18,081,501 certificates, as digital signatures, from 40,270,387 malware binaries.

In the case of Android APKs, we initially collected 80,000 APK files, out of which 79,652 were successfully decompressed. Generally, Google installation requirements enforce apps to be signed for distribution through the official Google Play application.[1] Therefore, we expected all Android developers to follow this security

[1] https://developer.android.com/google/play/requirements/target-sdk

Table 5.1: The summary of collected certificates and keys

Datasets	Collection period	Responsive hosts/ collected files	Collected certificates and keys	All valid certificates	Valid RSA certificates & keys	Unique RSA keys
Collected TLS/SSL	8/2013-11/2013	1,675,040 IPs	863,872 certs	863,871 (100%)	863,500 certs	863,500 (100%)
Collected SSH	5/2021-9/2021	10,435,118 IPs	13,681,145 keys	n/a	13,681,145 keys	9,747,793 (71.25%)
Rapid7	10/2013-09/2015, 8/2019, 9/2020-7/2021	99,910,085 IPs	295,063,780 certs	234,865,702 (79.6%)	135,823,576 certs	104,904,586 (77.24%)
Malware	2012-2021	40,270,387 files	18,081,501 certs	18,081,501 (100%)	18,081,489 certs	41,282 (0.23%)
Android APKs	9/2020-10/2020	72,508 APKs	118,743 certs	29,247 (24.63%)	29,247 certs	29,247 (100%)
SBA	4/2015 - 9/2015	n/a	202,381 certs	202,381 (100%)	202,378 certs	121,990 (60.28%)
Total	-	-	**314,330,277 certs 13,681,145 keys**	**254,042,702 (77.45%)**	**168,681,335 keys 155,000,190 certs**	115,708,398* (68.60%)

* not deduplicated across sets

practice, but only 72,508 APKs had at least one cryptographic content. We were able to extract 118,743 certificates from these files which formed our *original* set for Android APKs.

The combination of certificates collected from malware binaries and APKs resulted in a total of 18,200,244 certificates from file-based sources.

In total, 314,330,277 certificates and 13,681,145 keys were collected for analysis. However, not all the certificates and keys are parsable due to corrupted formats, password protections, or outdated standards. Also, considering the fact that our focus in this study is on RSA certificates, after the filtering process, out of 314,330,277 certificates, we extracted 155,000,190 valid certificates containing RSA keys. Overall, we were left with 168,681,335 valid RSA keys for our analysis.

5.3 Initial Analysis

We observed that there is a significant amount of duplication across certificates and keys used on the Internet. The highest amount of duplication is present in our Malware collection where almost all keys (99%) found in binaries were also seen in other sets. Around 29% of keys served by SSH hosts and 43% of Rapid7 certificates are duplicates.

Some aspects of the key or certificate reuse phenomenon were noted by previous studies. For example, Heninger *et al's* [31] study conducted in 2011-2012 on a smaller scale (12.8 million TLS hosts and 10.2 million SSH hosts) showed that 61% of TLS hosts and 65% of SSH hosts serve the same key as other hosts. It appears that the usage of repeated keys has decreased since that time, yet remains a considerable issue. To understand the current state of the PKI ecosystem, we investigate the strength of collected keys and certificates individually.

Table 5.2: RSA public key size

Key size range (bits)	Total	Frequency						
		Total Unique	SSL/TSL	SSH	Rapid7	Malware	Android APKs	SBA
0-1023	**4,203,348 (2.49%)**	582,823 (0.69%)	9,539 (1.12%)	0 (0.00%)	581,173 (0.78%)	1,126 (3.05%)	10 (0.03%)	240 (0.20%)
1024-2047	**47,050,256 (27.89%)**	26,223,521 (30.98%)	385,044 (45.30%)	225,357 (2.31%)	25,958,127 (34.65%)	10,554 (28.61%)	9,544 (32.68%)	26,356 (21.69%)
2048-4095	111,732,314 (66.24%)	55,309,352 (65.33%)	444,259 (52.26%)	9,468,431 (97.13%)	45,892,172 (61.26%)	24,405 (66.15%)	18,767 (64.25%)	89,481 (73.64%)
4096-8191	5,684,453 (3.37%)	2,536,655 (3.00%)	11,111 (1.31%)	53,539 (0.55%)	2,480,086 (3.31%)	802 (2.17%)	885 (3.03%)	5,395 (4.44%)
8192-up	10,964 (0.01%)	6,603 (0.01%)	64 (0.01%)	466 (0.005%)	6,122 (0.01%)	8 (0.02%)	2 (0.01%)	38 (0.03%)
Total	**168,681,335**	**84,658,954 (50.19%)**	850,017	9,747,793	74,917,680	36,895	29,208	121,510

5.3.1 Weak key size

We analyzed the strength of the collected RSA keys based on their modulus length. Table 5.2 shows the wide presence of weak keys across collections.

Among the collected keys, 30% (51,253,604) of keys 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 [4, 5].

NIST also recommended deprecating signing certificates that contained RSA keys shorter than 1024 bits by the end of 2013. However, across all our scans, 2.49% (4,203,348) of keys are less than 1024 bits in length and thus have deprecated status. While some of these keys are found in the TLS/SSL set collected in 2013 and the Rapid7 set partially covering 2013, many come from sets collected more recently (in 2019-2021) indicating that legacy keys are still widely in use.

Among these weak keys, 445,900 (0.26%) are 512-bit RSA keys. Only a portion of these keys (119,612) in our sets is associated with an IP address, so we can see that 119,612 TLS hosts serve these vulnerable RSA keys. As a comparison, in 2012 Heninger *et al.* [31] showed that 123,038 TLS hosts were using 512-bit keys, which reveals that the numbers have not decreased over time. Since we do not have the corresponding host information for 326,288 keys, it is likely that the actual number of TLS hosts using 512-bit keys is significantly underestimated.

5.3.2 Breakable RSA keys

Besides the weaknesses examined related to the modulus of keys, we explore other vulnerabilities that lead to breaking an RSA key such as weak exponents, the ROCA vulnerability, and factorable GCD. The summary of these experiments is given in Table 5.3.

Weak exponent

Out of all collected keys, 28 RSA keys have an exponent equal to one, and 42 keys have an even exponent. The exponent is supposed to be a large coprime number, preferably equal to 65537 as recommended by NIST.

In the cases when $e = 1$, deriving a private key is trivial, hence, the corresponding public key is considered weak. When an even exponent is used, by calculating the square root of the ciphertext, it is potentially possible to retrieve the original plaintext without possessing the private key.

ROCA

We have also tested collected keys for the Return of Coppersmith's Attack (ROCA), a security vulnerability that affects the cryptographic keys generated by a specific type of hardware RNG called Infineon RSA library [51]. The cryptographic key originated from the Infineon RSA library allows an attacker to exploit the weak prime numbers generated by the faulty implementation using Coppersmith's algorithm. To test the gathered keys for this vulnerability, we have used the ROCA detection tool.[2] Results show that 231 keys in our all collected data were vulnerable to ROCA.

GCD-Factorable

One of the threats to the RSA cryptosystem is the possibility of factorizing modulus to decompose it to p and q values, and consequently to compute the private key. Theoretically, such factorization is computationally intensive and should be unfeasible for sufficiently large p and q numbers. Yet in practice, the occurrence of weak keys is more common which makes the factorization possible in some cases. Studies by Heninger *et al.* [31] and Gasser *et al.* [27] measured the spread of weak factorable keys in 2012 and 2014.

To determine the presence of weak factorable keys, we used the *Fastgcd*[3] tool, which was originally developed by Heninger *et al.* [31]. The tool performs a pairwise computation of GCDs of all moduli to determine if any of them share a prime number with any other modulus, in which case, a computation of the corresponding private key is straightforward. In our study, the GCD's computation was performed on 141,098,520 unique moduli extracted from the collected certificates and keys. We were able to find divisors for 185,731 (0.13%) moduli that were present in 793,694 keys in our sets, i.e., we could factor the moduli corresponding to 793,694 keys (Table 5.3). This is considerably higher (80 times) than the numbers reported by the previous studies, e.g., Heninger *et al.* [31] reported finding divisors for 2,314 out of 11,170,883 moduli. Since our coverage is significantly larger than the previous studies, we believe our results are more representative of the security state of the RSA keys.

5.3.3 Use of certificates over time

The presence of legacy devices with legacy certificates on the Internet can partially provide an explanation of key weaknesses. We therefore also analyzed the lifetime of our collected certificates. To explore the use of certificates across time, we grouped all certificates based on the time of their collection. Within each time interval, we discarded duplicates based on their fingerprints and checked the presence of the remaining unique

[2] https://github.com/crocs-muni/roca
[3] https://factorable.net/resources.html

Table 5.3: Factorable RSA keys

Heninger *et al.* [31]	
SSH hosts using factorable RSA keys	0.03%
TLS hosts using factorable RSA keys	0.5%
Gasser *et al.* [27]	
SSH hosts using factorable RSA keys	0.013%-0.016%
Our results:	
Factorization analysis:	
GCD factorable moduli	185,731 (0.13%)
Total impacted RSA keys	793,694 (0.47%)
Total impacted certificates	792,246 (0.31%)
Total impacted hosts	35,700
Key dataset:	
TLS/SSL	3,923
SSH	1,448
Rapid7	786,293
Malware	2,030
Android APKs	0
SBA	0
Key size	
0-1023	14,279
1024-2047	769,986
2048-4095	9,327
4096-8191	98
8192-up	4
ROCA vulnerability (keys)	231
Exponent = 1	28
Exponent = *even number*	42

certificates in other time intervals. Figure 5.1 shows the frequency of distinct certificates seen across different time ranges.

The certificates appearing in earlier time ranges (2013 - 2015) are rarely seen at later times. This is generally expected as some of the older certificates will become obsolete due to evolving cryptographic standards and would require organizations to generate new keys and the corresponding certificates to comply with the new rules and recommendations. For example, the introduction of SHA-3 in 2015, and consequently, the requirement to phase out certificate chains with SHA-1 by 2017 naturally led to the certificates being reissued.

The drastic shift is clearly visible in recent years. The overwhelming majority (77-90%) of certificates collected in 2020 and 2021 were encountered across multiple time ranges between 2013 to 2015. This fact suggests that organizations are not replacing their certificates as frequently as expected. Many of these certificates are shared between our sets, we thus focus specifically on the shared RSA certificates and keys.

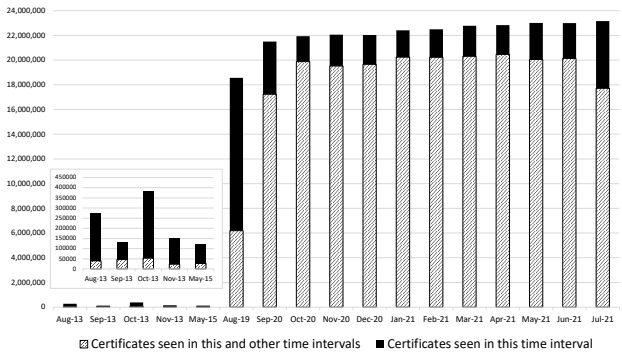

Figure 5.1: The use of distinct certificates across time.

5.4 The shared RSA certificates

Out of 155,000,190 valid RSA certificates, 10,110,361 (6.5%) are shared across the collected datasets (Table 5.4).

Table 5.4: The shared RSA certificates

Datasets	**Total shared**	**Unique shared in each set****	**Certificates shared with other sets**				
			Collected TLS/SSL	**Rapid7**	**Malware**	**SBA**	**Android APKs**
Collected TLS/SSL	752,029 (7.44%)	752,029 (100%)	*	751,993	0	3,948	1
Rapid7	863,792 (8.54%)	863,284 (99.94%)	752,309	*	491	114,733	227
Malware	8,302,431 (82.12%)	377 (<0.01%)	0	**8,299,100**	*	0	**2,338,672**
SBA	191,919 (1.90%)	114,734 (59.78%)	17,496	191,879	0	*	0
Android APKs	190 (<0.01%)	190 (100%)	1	**187**	58	0	*
Total	**10,110,361**	1,730,614	**863,323 (8.54%)** are distinct across sets				

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

There are common scenarios where certificates can be potentially shared across multiple hosts and domains. A certificate may belong to an organization that serves this certificate across any IP addresses that belong to it. Conversely, a certificate may come from a third-party hosting provider supporting multiple clients. Historically, an SSL certificate was issued to a host/domain name indicated in the "Common Name" field within the "Subject" field of the certificate, establishing a one-to-one mapping between a certificate and a host. With the growing prevalence of website hosting, in 2000 [58], certificates were permitted to encompass more than one domain name, enabling the utilization of a single certificate across multiple hosts through the use of the "Subject Alternative Name" extension. This development led to providers commonly employing custom certificates that encompass multiple domains or organizations within a single certificate. These certificates could be served from a single IP address belonging to a hosting provider, by using wildcard certificates.[4]

[4]In RFC 2818, section 3.1 allows for wildcard character * which is considered to match any single domain name.

In the following subsections, we look at the various aspects of the certificate reuse phenomenon.

5.4.1 Reuse of certificates over time

The vast majority of all shared certificates are shared between the Rapid7 set and other sets. A large cluster of SSL certificates is seen in our collected TLS/SSL and Rapid7 set. The presence of these shared certificates is not surprising. Both sets overlap in time (the year 2013), hence, they may potentially include certificates obtained from the same hosts. However, further timeline analysis of these shared certificates shows a significant time gap, i.e., the fingerprint of certificates found in TLS/SSL set match certificates collected by Rapid7 during 2019-2021 (Figure 5.1). This implies that many certificates have been reused by different hosts for 7-8 years.

For example, out of the certificates that were analyzed, 58 have been consistently used since 2013 and have appeared a total of 6,873 times across our datasets. Interestingly, 31 of them were self-signed with the sha1WithRSAEncryption signature algorithm and were issued by GlobalSign, one of the largest CAs, which is a part of the CA/B (Certification Authority/Browser) Forum, an industry organization that establishes the rules and governs the issuance and management of digital certificates. An analysis of key usage extension fields indicates that these 58 certificates are intended for various purposes including internal CA use (29 certificates), browser use (6 certificates), device authentication use (16 certificates), and authentication for firewall and security gateways (7 certificates).

Over time, the validity period of certificates varied according to their usage. For example, in cases when renewing certificates was not feasible, RFC 5280 [8] released in 2008 allowed CAs to issue certificates with no expiration date. However, the validity of certificates has since been significantly shortened.

In 2017, the CA/B Forum established the maximum validity of certificates to two years (825 days). Prior to this, the maximum validity was three years for most certificates. In 2022, the CA/B Forum set the maximum validity period of a TLS certificate to 398 days.[5] In March 2023, Google announced that the maximum certificate validity will be soon reduced to 90 days for all publicly trusted TLS certificates. While recommendations leading to shorter certificate validity emphasize the idea of regularly rotating keys to enhance security and mitigate the potential impact of key compromise, the CA/B Forum guidelines do not affect already issued certificates, effectively facilitating the reuse of legacy certificates.

Further investigation revealed that one of these consistently used certificates has been shared 5,989 times, 4 times in the Rapid7 set and 5,985 times in the Malware set. This is a CA signing certificate distributed by GlobalSign, i.e., this certificate can be only used by a CA to sign other certificates and CRLs. The presence of these non-compliant certificates within the PKI ecosystem is puzzling and dangerous, considering that CA certificates serve as the foundation of trust in the PKI ecosystem. Such incidents underscore the growing concerns to investigate the situation of shared certificates and keys deeper.

[5] https://cabforum.org/wp-content/uploads/CA-Browser-Forum-BR-1.8.4.pdf

5.4.2 Reuse of certificates for different purposes

While the overlap between the collected TLS/SSL set and Rapid7 is expected, the other cases reveal worrisome patterns. For example, we observed that 28% (2,338,672) of certificates used in malicious binaries can also be found in Android apps (Table 5.4). There have been reports of malicious binaries signed with valid certificates [13, 41] in the past. The reappearance of this problem in Android apps that were collected in 2020, and are supposed to be legitimate, indicates that this phenomenon may be more widespread than initially believed.

We also identified a considerable amount of certificates shared between malware and the Rapid7 sets, i.e., with almost all certificates collected from malware binaries (99.9%) found in the Rapid7 set containing TLS/SSL certificates.

Although both types of certificates are integral to the PKI ecosystem, they have different purposes. The code signing mechanism allows authentication software publisher, while an SSL certificate serves to verify the identity of a server. Typically, the purpose of the certificate can be specified in the "Key Usage" and "Extended Key Usage" fields of the certificate. Certificates issued for one purpose should not be used for different usage (e.g., a TLS certificate cannot be used to sign code). However, the overlap shows that at some point these malicious certificates were used for both purposes.

5.4.3 Weak signatures algorithms

Valid digital certificates must be signed by the certification authority that issued them. While RFC 3279 [34] and its subsequent versions permit the use of any public key signature algorithm in conjunction with a one-way hash function, NIST recommends appropriate hash functions based on the algorithm's strengths.

To examine the cryptographic algorithms utilized by the certificate issuer to produce their digital signatures, we extracted the object identifiers (OID) of the cipher algorithms from the digital signature section of the certificates. For ease of interpretation, we resolved the OIDs into their corresponding algorithm names. For instance, the OID 1.2.840.113549.1.1.5 was translated to sha1WithRSAEncryption.

The list of the observed signature algorithms in shared certificates is shown in Table 5.5. The table lists a total of 10,110,361 certificates that were associated with 863,323 unique fingerprints (Table 5.4).

We discovered that the use of obsolete algorithms among certificates is overwhelmingly high.

The vast majority of the shared certificates ($>$ 9 million) are signed using the RSA algorithm with SHA-1 and MD5 hashing. Both have not been recommended for use. NIST deprecated the use of SHA-1 in 2011 and disallowed its use for digital signatures in 2013 [52]. Although our TLS/SSL set was collected at the end of 2013, the total number of these certificates (863,871) is several times smaller than the observed number of certificates with the use of SHA-1 algorithm (e.g., dsaWithSHA1, ecdsa-with-SHA1, sha1WithRSA, sha1WithRSAEncryption).

Similarly, the MD5 hashing algorithm has been considered broken, and unacceptable for use in digital

Table 5.5: Signature algorithms seen in shared certificates

Signature algorithm	Total shared	Unique certificate
sha1WithRSAEncryption	8,986,133	770291 (8.57%)
md5WithRSAEncryption	536,080	62830 (11.72%)
sha256WithRSAEncryption	275,416	26451 (9.60%)
sha384WithRSAEncryption	272,627	27 (0.01%)
md2WithRSAEncryption	32,644	15 (0.05%)
sha1WithRSA	5,646	2818 (49.91%)
sha512WithRSAEncryption	1,635	801 (48.99%)
rsaEncryption	116	58 (50.00%)
dsaWithSHA1	20	10 (50.00%)
rsassaPss	20	10 (50.00%)
ecdsa-with-SHA256	8	4 (50.00%)
shaWithRSAEncryption	4	2 (50.00%)
ecdsa-with-SHA1	4	2 (50.00%)
ecdsa-with-SHA512	4	2 (50.00%)
GOST R 34.11-94 with GOST R 34.10-2001	2	1 (50.00%)
sha224WithRSAEncryption	2	1 (50.00%)
Total	**10,110,361**	**863,323 (8.54%)**

signatures since 2008 [16, 65]. Yet, over 500,000 shared certificates using MD5 hashing are still in use. Out of these certificates, the Malware set accounted for the largest proportion, i.e., 75.87% of certificates, followed by Rapid7 (11.73%), TLS/SSL (10.90%), and the SBA dataset (1.50%).

To our surprise, we noticed that roughly 32,000 shared certificates were signed with the MD2 hashing algorithm, which was deprecated in 2011 [67], and thus should not have been present in any of our collections.

Unexpectedly, we also found a certificate, shared two times, that uses GOST hashing functions as the signature algorithm which based on RFC 4491 [62] was a valid algorithm for PKI and CRL profile but has been removed from the OpenSSL family of libraries in 2016.[6]

Our numbers indicate that the use of stronger signature algorithms (e.g., SHA-256, SHA-512) is barely noticeable (∼3%) among the shared certificates. We note that in 2015, NIST recommended the minimum use of SHA-256 for any application of hash functions requiring interoperability[7], and in 2020, NIST made it a requirement for all certificates to be signed with an approved signature algorithm and hash algorithm, such as SHA-256) [63].

5.4.4 Who uses shared certificates

We identified 863,323 unique certificates across all sets (Table 5.4), linked to 29,937,166 reachable IP addresses. While not every shared certificate is linked to an IP address in our sets, all corresponding IP addresses are unique.

The majority of these hosts are mainly linked to a single distinct shared certificate, meaning that most hosts in our sets use the same certificate as another host. Out of the total, 17,785,693 addresses (59.41%) utilize a single certificate, while 12,151,473 IP addresses (40.59%) are connected to multiple distinct shared certificates, as illustrated in Figure 5.2.

By focusing our attention on the IP addresses associated with 13 or more certificates, we found a total of

[6] https://www.openssl.org/news/changelog.txt
[7] https://csrc.nist.gov/Projects/Hash-Functions/NIST-Policy-on-Hash-Functions

228 distinct certificates duplicated 4,122,199 times and issued by a variety of organizations.

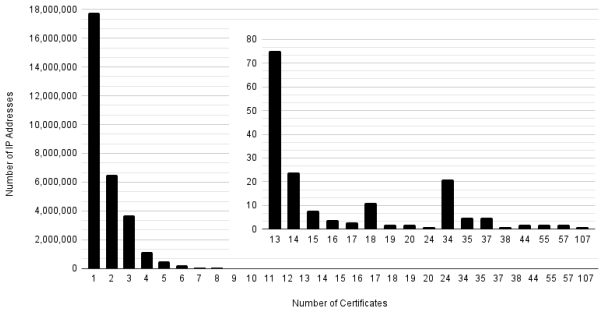

Figure 5.2: The distribution of shared certificates among IP addresses.

To further understand what devices and applications are associated with shared certificates, we scanned IP addresses that share certificates using the nmap application between December 2021 to February 2022. Out of 29,937,166 IP addresses that share certificates, 1,557,199 IP addresses replied with a valid response to the nmap scan.

Due to the time gap between collection and analysis, we took a step to validate certificates. To unambiguously match an IP address to the corresponding certificate, we initiated a TLS/SSL connection with the IP addresses of the shared certificates. We requested their current certificate and matched their fingerprint with the one we have stored in our records. This step ensured that the host/organization using the IP address is the same and our further analysis is relevant. 51.32% (5,188,895) of shared certificates were validated. Our further analysis focuses on IP addresses with validated certificates.

For the majority of these 1,557,199 hosts, nmap was able to identify the operating system (OS) (89.62% hosts) and the device type (88.24% hosts). The remaining cases were categorized as "None" since nmap could not provide any information on the hosts' software or hardware. The summary of the nmap scan is given in Table 5.6.

Table 5.6: The summary of nmap scan

Total shared certificate	10,110,361
Validated shared certificates	5,188,895 (51.32%)
Total IP addresses	29,937,166
Hosts with valid nmap response	1,557,199
Fingerprinted OS	
Hosts with validated certificates	1,395,662 (89.62%)
None	161,537
Fingerprinted device	
Hosts with validated certificates	1,374,178 (88.24%)
None	183,021

Our scan discovered 169 distinct operating systems within 1,395,662 hosts that served shared certificates. We investigated the top 20 OSs based on the frequency of responded IP addresses (Table 5.7).

The certificates are predominantly shared by embedded network devices including routers, mobile phones,

Table 5.7: The top 20 operating systems seen in hosts sharing certificates

OS Name	Released Year	IP frequency	Unique shared certs	Key Size		Signature	
				≥2048 bits	<2048 bits	Strong*	Weak
Linux 3.18	2014	721,818 (51.72%)	7,970	5,847 (73.36%)	2,123 (26.64%)	502 (6.30%)	7,468 (93.70%)
ZyXEL ZyWALL 70 firewall (ZyNOS 3.65)	2008	173,036 (12.40%)	4,232	2,532 (59.83%)	1,700 (40.17%)	150 (3.54%)	4,082 (96.46%)
Aerohive HiveOS 6.8	2018	136,509 (9.78%)	2,053	1,278 (62.25%)	775 (37.75%)	115 (5.60%)	1,938 (94.40%)
Linux 3.13	2014	59,597 (4.27%)	1,107	737 (66.58%)	370 (33.42%)	136 (12.29%)	971 (87.71%)
iPXE 1.0.0+	2010	56,355 (4.04%)	965	629 (65.18%)	336 (34.82%)	64 (6.63%)	901 (93.37%)
Linux 2.6.32	2009	55,700 (3.99%)	1,309	976 (74.56%)	333 (25.44%)	72 (5.50%)	1,237 (94.50%)
Efficient Networks 5930 ADSL router	2002	43,924 (3.15%)	868	495 (57.03%)	373 (42.97%)	64 (7.37%)	804 (92.63%)
D-Link DWL-624+ or DWL-2000AP or TRENDnet TEW-432BRP WAP	2005-2007	19,926 (1.43%)	750	429 (57.20%)	321 (42.80%)	62 (8.27%)	688 (91.73%)
Panasonic BL-C210A webcam	2009	13,463 (0.96%)	140	104 (74.29%)	36 (25.71%)	35 (25.00%)	105 (75.00%)
Juniper Networks SSG 20 firewall	2006	10,397 (0.74%)	146	94 (64.38%)	52 (35.62%)	35 (23.97%)	111 (76.03%)
Apple iOS 8.0 - 8.1 (Darwin 14.0.0)	2014	9,070 (0.65%)	387	264 (68.22%)	123 (31.78%)	49 (12.66%)	338 (87.34%)
Canon i-SENSYS MF5490dn printer	2008	7,235 (0.52%)	109	78 (71.56%)	31 (28.44%)	33 (30.28%)	76 (69.72%)
Linux 2.6.18 - 2.6.22	2006-2007	7,048 (0.50%)	264	171 (64.77%)	93 (35.23%)	49 (18.56%)	215 (81.44%)
Vivint alarm panel (Linux 2.6.21)	ukn.	4,889 (0.35%)	174	106 (60.92%)	68 (39.08%)	35 (20.11%)	139 (79.89%)
Cisco 7200 router (IOS 12.4)	2005	4,870 (0.35%)	82	73 (89.02%)	9 (10.98%)	38 (46.34%)	44 (53.66%)
Panasonic WV-SP300 or WV-SF330 webcam	2010-2011	4,868 (0.35%)	72	63 (87.50%)	9 (12.50%)	32 (44.44%)	40 (55.56%)
Microsoft Windows Server 2012 R2	2012	4,868 (0.35%)	168	110 (65.48%)	58 (34.52%)	42 (25.00%)	126 (75.00%)
Epson UB-E02 print server	ukn.	4,756 (0.34%)	334	168 (50.30%)	166 (49.70%)	37 (11.08%)	297 (88.92%)
Moxa NPort 5610 terminal server	ukn.	4,540 (0.33%)	154	102 (66.23%)	52 (33.77%)	34 (22.08%)	120 (77.92%)
Linux 4.9	2016	4,040 (0.29%)	82	69 (84.15%)	13 (15.85%)	37 (45.12%)	45 (54.88%)
Total	-	1,346,909 (96.51%)	21,366	14,325 (67.05%)	**7,041 (32.95%)**	1,621 (7.59%)	**19,745 (92.41%)**

"*" Strong: signature algorithm is either sha256WithRSAEncryption, sha384WithRSAEncryption, or sha512WithRSAEncryption

printers, and firewalls with the majority of them using OS derived from Linux and BSD distributions.

Linux 3.18 OS, released in 2014, is found to be the most common OS serving shared certificates by the majority of hosts. Google and other vendors seem to be utilizing Linux 3.18 on numerous Android-based devices. Some Chromebooks are also operating on the same kernel version as part of Chrome OS. We further investigated the device types of IP addresses associated with this OS and not surprisingly all matched devices are categorized as general-purpose computing systems.

Apart from Linux kernels, the rest of the operating systems indicate network-connected devices, such as HiveOS, SSG firewall, wireless and ADSL routers, webcams, printers, printer servers, terminal servers, and home security controllers, which have embedded computing capabilities and can be remotely managed and monitored. Most of these devices come from a handful of manufacturers: ZyXEL Communications Corporation, Aerohive Networks, IPEX Group, Siemens, D-Link Corporation, TRENDnet, Panasonic Corporation, Juniper Networks, Apple, Canon, Vivint Smart Home, Cisco Systems, Microsoft Corporation, Epson Corporation, and Moxa.

We have also seen a large number of certificates shared over an extended period of time. Devices identified

as the "Efficient Networks 5930 ADSL router" released in 2002 and the "Cisco 7200 router" released in 2005 are among the oldest seen in our scan, pointing that these shared certificates have been actively used for over two decades.

To validate these observations, we delved deeper into the certificates' validity periods. Our investigation revealed certificates with notably extended validity periods, starting from their manufacturing date and spanning up even to an impressive 50 years. There are a total number of 48,794 IP addresses and 950 distinct certificates associated with these older systems. It appears these instances are indicative of factory-default certificates being in use.

We also analyzed our shared certificates for the presence of certificates generated with default parameters by looking at certificates generated by Linux and BSD systems that still use so-called "Sneak Oil" certificate generation scripts, for automatically generating self-signed certificates. We found that 381 certificates were associated with systems that used default configuration options to establish encrypted TLS/SSL communications.

With the legacy status of the devices, we anticipated finding outdated cryptographic settings and discovered that around 33% (7,041) of these certificates relied on weak RSA keys with key lengths less than 2048. However, the vast majority of certificates (92.41%) were signed with weak signature algorithms that were introduced before sha256WithRSAEncryption, including md5WithRSAEncryption and sha1WithRSAEncryption present in our set (Table 5.7).

We conducted a similar analysis to identify the hardware hosting the shared certificates which confirmed our conclusions (Table 5.8).

5.5 The shared RSA keys

In the previous section, we delved deeper into the phenomenon of shared certificates, yet, we were surprised to discover that multiple distinct certificates were serving identical public keys, i.e., identical modulus and/or exponent. Therefore, we decided to not only match certificates but also conduct a pairwise comparison of valid RSA keys across all sets.

We found that 10.16% (17,141,441) of all keys are shared across sets, and 87% (14,862,767) of these duplicate keys (116,868 distinct) appear in distinct certificates (Table 5.9).

5.5.1 Vulnerable keys

In our initial analysis (Section 5.3), we discussed several weaknesses associated with all RSA keys, here we specifically examine weaknesses of the shared keys.

Table 5.8: Key properties of devices seen in shared certificates

Device Name	IP frequency	Unique shared certs	Key Size		Signature	
			≥2048 bits	<2048 bits	Strong*	Weak
general purpose	868,432 (63.20%)	9,703	7,001 (72.15%)	2,702 (27.85%)	578 (5.96%)	9,125 (94.04%)
firewall	189,588 (13.80%)	4,398	2,631 (59.82%)	1,767 (40.18%)	159 (3.62%)	4,239 (96.38%)
WAP	140,013 (10.19%)	2,123	1,299 (61.19%)	824 (38.81%)	118 (5.56%)	2,005 (94.44%)
specialized	61670 (4.49%)	1,057	671 (63.48%)	386 (36.52%)	66 (6.24%)	991 (93.76%)
broadband router	46349 (3.37%)	887	501 (56.48%)	386 (43.52%)	65 (7.33%)	822 (92.67%)
webcam	18339 (1.33%)	156	118 (75.64%)	38 (24.36%)	38 (24.36%)	118 (75.64%)
phone	9321 (0.68%)	398	270 (67.84%)	128 (32.16%)	50 (12.56%)	348 (87.44%)
printer	7618 (0.55%)	135	91 (67.41%)	44 (32.59%)	36 (26.67%)	99 (73.33%)
router	5731 (0.42%)	104	78 (75.00%)	26 (25.00%)	38 (36.54%)	66 (63.46%)
print server	4756 (0.35%)	334	168 (50.30%)	166 (49.70%)	37 (11.08%)	297 (88.92%)
terminal server	4545 (0.33%)	156	103 (66.03%)	53 (33.97%)	35 (22.44%)	121 (77.56%)
load balancer	3549 (0.26%)	117	80 (68.38%)	37 (31.62%)	40 (34.19%)	77 (65.81%)
VoIP adapter	3325 (0.24%)	185	117 (63.24%)	68 (36.76%)	43 (23.24%)	142 (76.76%)
switch	3062 (0.22%)	91	66 (72.53%)	25 (27.47%)	24 (26.37%)	67 (73.63%)
proxy server	2933 (0.21%)	118	82 (69.49%)	36 (30.51%)	36 (30.51%)	82 (69.49%)
media device	1565 (0.11%)	157	98 (62.42%)	59 (37.58%)	35 (22.29%)	122 (77.71%)
storage-misc	1496 (0.11%)	330	190 (57.58%)	140 (42.42%)	30 (9.09%)	300 (90.91%)
terminal	1111 (0.08%)	79	65 (82.28%)	14 (17.72%)	30 (37.97%)	49 (62.03%)
VoIP phone	413 (0.03%)	105	51 (48.57%)	54 (51.43%)	27 (25.71%)	78 (74.29%)
remote management	349 (0.03%)	48	41 (85.42%)	7 (14.58%)	24 (50.00%)	24 (50.00%)
bridge	13 (<0.01%)	5	5 (100.00%)	0 (0.00%)	5 (100.00%)	0 (0.00%)
Total	1,374,178	20,686	13,726 (66.35%)	**6,960 (33.65%)**	1,514 (7.32%)	**19,172 (92.68%)**

"*" Strong: signature algorithm is either sha256WithRSAEncryption, sha384WithRSAEncryption, or sha512WithRSAEncryption

Weak key size

Our analysis found that 44% (6,595,972) of shared keys can be considered weak, i.e., with key length < 2048 bits (Table 5.11). This can be interpreted as a strong indicator of the association of duplicate keys with their perceived vulnerability level.

ROCA

We identified one shared key vulnerable to ROCA vulnerability. The key was duplicated 16 times in the Rapid7 dataset with 3 different signature algorithms, i.e., md5WithRSAEncryption, sha1WithRSAEncryption, and sha256WithRSAEncryption. However, our cursory check revealed different issuers and owners corresponding to these certificates.

GCD-Factorable

Among the 793,694 factored RSA keys, 191,236 are duplicate keys that appear in distinct certificates. As expected, almost all of these factored keys are less than 2048 bits in length (Table 5.10). Interestingly, duplicate keys retrieved from SSH hosts, malware executables, and Android apps, appear to be noticeably absent from GCD factorable keys, i.e., we were able to factor only 12 SSH keys and none of the Android or malware keys.

Table 5.9: The shared RSA keys

Datasets	Total number of shared keys	Unique shared in each set**	Shared keys with distinct certificates	Unique shared keys with distinct certificates**	Shared keys found in distinct certificates					
					Collected TLS/SSL	Collected SSH	Rapid7	Malware	SBA	Android APKs
Collected TLS/SSL	767,709	754,324 (98.26%)	123,582	110,197 (89.17%)	*	48	123,581	11	805	0
Collected SSH	195,107	169,851 (87.06%)	3,961	2,502 (63.17%)	1,039	*	3,961	0	2	0
Rapid7	7,619,478	1,034,226 (13.57%)	6,556,755	116,867 (1.78%)	6,528,028	8,519	*	898	38,279	594
Malware	8,366,724	320 (<0.01%)	8,165,826	145 (<0.01%)	133	0	8,157,286	*	0	2,899,636
SBA	192,210	114,492 (59.57%)	12,524	4,593 (36.67%)	4,657	3	12,523	0	*	0
Android APKs	213	196 (92.02%)	119	102 (85.71%)	0	0	117	58	0	*
Total	**17,141,441**	1,034,263 (6.03%)***	**14,862,767 (86.71%)**	234,406 (1.58%)	**116,868 keys (0.79%)** are distinct across sets					

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

Table 5.10: GCD-factorable shared RSA keys

Impacted distinct shared keys	3,182
Impacted shared keys	191,236
Dataset	
SSL/TSL	3,856
SSH	12
Rapid7	187,368
Malware	0
Android APKs	0
SBA	0
Key size	
0-1023	327
1024-2047	190,884
2048-4095	25

5.5.2 Sources of duplicate keys

The majority of the keys are shared between Collected TLS/SSL and Rapid7 sets. To limit the impact of a potential key compromise, organizations use different keys for distinct purposes when they need more than one certificate. However, in some cases, for example, for consistency in single sign-on mechanisms using the SAML[8], load-balanced environments, multi-domain SSL certificates, and shared hosting situations, organizations may prefer or need to use the same key for different certificates, provided that the certificates are related in terms of the owner.

To understand the reasons behind duplicate keys in our collections, we analyzed the "Subject" field in selected certificates and subsequently parsed the "Organization" field of these shared keys. We observe that only 7,060 (6.04%) out of 116,868 distinct keys have the same or related organization information. Hence, hosting situations appear to be responsible for only a small amount of duplicate keys.

[8]Security Assertion Markup Language

Our observations in Table 5.11 also point out that around 99% of unique weak keys are associated with both Rapid7 (100%) and our own collected TLS/SSL (98.42%) datasets.

Table 5.11: Shared RSA public key size

Key size range(bits)	**Total**	**Frequency**						
		Total Unique	**SSL/TSL**	**SSH**	**Rapid7**	**Malware**	**SBA**	**Android APKs**
0-1023	832,184 (5.59%)	2,193 (0.26%)	2,182	0	2,193	4	9	0
1024-2047	5,763,788 (38.78%)	35,583 (0.62%)	35,000	208	35,583	36	408	15
2048-4095	8,110,887 (54.57%)	76,695 (0.95%)	71,097	2,245	76,694	92	3,733	80
4096-8191	155,860 (1.05%)	2,383 (1.53%)	1,907	49	2,383	13	440	7
8192-up	48 (<0.01%)	14 (29.17%)	11	0	14	0	3	0
Total	14,862,767	116,868 (0.79%)	110,197	2,502	116,867	145	4,593	102

Interestingly, duplicate keys retrieved from SSH hosts, malware executables, and Android apps, appear to be compliant with NIST regulations (Table 5.11). When it comes to SSH keys, it is common for clients to be updated, which means that the majority of SSH keys should adhere to current standards. Similarly, keys associated with software compilation, such as malware executables and Android APKs, require compatibility with current standards due to compiler prerequisites. Consequently, these keys are more likely to be stronger.

The most commonly shared key that we could identify in our sets has been seen 3,454,586 times in the Rapid7 set. This key is associated with 3,454,586 distinct TLS certificates, mostly belonging to the same organization, Lancom Systems. We have different records for these certificates. They were served by 11 distinct IPs, and classified once by nmap as a "general purpose" device served by Microsoft Windows Server 2008 SP2. This key is weak, its key modulus length is 1024 bits. The associated certificates were signed with the sha1WithRSAEncryption algorithm, but our TLS scan showed this key still is being served in certificates for an extended period of time despite its obvious weaknesses.

Another source of concern is the presence of shared keys between web (Rapid7 and Collected TLS/SSL) and file (Malware and Android APKs) datasets. For example, most RSA keys found in malware samples are served by TLS hosts, and 49% (2,899,636) of keys used by Android apps are used to sign malware. This suggests that corresponding private keys could be within the reach of malware authors or attackers. This presence of shared keys points towards a broader issue within the PKI ecosystem.

5.5.3 Key generators

Duplicated keys, where the same cryptographic key is generated more than once, serve as a warning sign and raise concerns about the security of the key generation process. Consequently, any similar keys produced using the same library should be viewed as questionable. It suggests a lack of proper security measures during key generation, making the generated keys unreliable and vulnerable to exploitation.

To identify the origin of the shared RSA keys, we have adopted an approach for a fine-grained RSA key origin attribution proposed by Branca *et al.* [10]. The approach is based on spatial characteristics of RSA moduli associated with different library implementations, consequently, allowing for accurate origin attribution.

We have retrained the Random Forest model using their generated set of 6.5 million RSA keys which contains details on the type and version of potential libraries used to generate each key. We utilized this model to deduce the libraries and their versions responsible for generating the shared RSA keys. The resulting set of 17,141,441 public keys contains 1,034,263 distinct RSA moduli. We retraced each modulus back to the original keys to confirm the corresponding dataset (see Table 5.12).

Table 5.12: Predicted libraries per dataset

Library	**Number of moduli**	**Affected keys**	**Frequency**					
			TLS/SSL	**SSH**	**Rapid7**	**Malware**	**SBA**	**Android APKs**
OpenSSL 1.1.x	1,002,727	17,004,035	744,543	189,405	7,534,379	8,350,100	185,408	200
GnuTLS 3.6.x	20,989	99,343	15,474	3,794	58,950	16,607	4,509	9
GnuTLS 2.2.x	10,451	37,440	7,623	1,891	25,632	17	2,273	4
OpenSSL 1.0.x	94	619	67	17	515	0	20	0
GnuTLS 3.1.x	1	2	1	0	1	0	0	0
GnuTLS 2.1.x	1	2	1	0	1	0	0	0
Total	**1,034,263**	17,141,441	767,709	195,107	7,619,478	8,366,724	192,210	213

.x stands for all minor versions

We found that the shared keys were predominantly generated by OpenSSL (97%) and GnuTLS (3%) libraries. This is not surprising as OpenSSL is the most widely used open-source cryptographic library installed by default in many Linux kernel-based systems.

The problems with low entropy pool affecting the generation of RSA key prime numbers in OpenSSL 1.0.0 on Linux-based systems were discovered by Heninger *et al.* [31]. This is again not surprising as OpenSSL library versions 1.0 and 1.1 have a history of issues related to random number generation implementations (e.g., CVE-2015-0285 [9], CVE-2015-3216 [10], CVE-2019-1549 [11]). However, only a few versions of OpenSSL and GnuTLS libraries appear to be responsible for duplicate keys (Table 5.13). For example, the OpenSSL 1.1.x library generated around 97% of duplicate keys. As expected (based on Tables 5.2 and 5.7), more than half of these keys are on the upper end of the key numerical range, i.e., 57% (9,735,163) of OpenSSL 1.1.x keys have a length 2048 bits or more. This appears to be common among all affected keys.

Although OpenSSL release notes of the 1.1.1 version state that the random number generator was completely rewritten to address this problem, the presence of weak moduli in a considerable percentage of our shared keys suggests that conditions may persist and be related to the issues of the aforementioned entropy pool. It is worth mentioning that there are 189,405 SSH keys that our analysis predicts were likely generated

[9] http://cve.mitre.org/cgi-bin/cvename.cgi?name=cve-2015-0285
[10] http://cve.mitre.org/cgi-bin/cvename.cgi?name=cve-2015-3216
[11] http://cve.mitre.org/cgi-bin/cvename.cgi?name=cve-2019-1549

Table 5.13: Predicted libraries per modulus

Library	Number of moduli	Affected keys	Modulus size range (bits)				
			0-1023	1024-2047	2048-4095	4096-8191	8192-up
OpenSSL 1.1.x	1,002,727	17,004,035	824,147	6,444,725	9,556,530	178,406	227
GnuTLS 3.6.x	20,989	99,343	14,987	34,218	42,968	7,170	0
GnuTLS 2.2.x	10,451	37,440	5,834	14,747	16,505	354	0
OpenSSL 1.0.x	94	619	294	182	135	8	0
GnuTLS 3.1.x	1	2	0	2	0	0	0
GnuTLS 2.1.x	1	2	0	0	2	0	0
Total	1,034,263	17,141,441	845,262	6,493,874	9,616,140	185,938	227

.x stands for all minor versions

using older versions of the OpenSSL library. This observation underscores not only the continued usage of outdated cryptographic libraries used in web protocols as well as libraries targeted at infrastructure services such as SSH (Table 5.12).

5.5.4 Certificates with shared keys

Furthermore, to identify legacy public key material, we started from 116,868 distinct public keys that we identified to be shared across all sets, this allowed us to detect 14,895,604 certificates that 14,444,022 (96.97%) certificates were linked to either SHA-1 (e.g. sha1WithRSAEncryption, sha1WithRSA) or MD5 hashing algorithms, both of which are no longer supposed to be used for general or browsing purposes, according to NIST recommendations [48].

Also, out of 14,895,604 certificates, a total of 35,571 certificates were found to be using non-standard OIDs as signature algorithms, as shown in table 5.14.

Table 5.14: Unresolvable signature algorithms seen in certificates with shared keys

Signature algorithm	Total certificates
1.2.840.113549.1.60.21	26,449
1.2.840.113549.1.60.20	8,613
1.2.840.113549.1.60.27	473
1.2.840.113549.1.60.26	29
1.2.840.113549.1.60.29	3
1.2.832.113549.1.1.4	1
1.2.840.113037.1.1.5	1
1.2.840.113548.1.1.5	1
1.2.840.114573.1.1.5	1
Total	35,571

We also observed that out of 35,571 certificates corresponding to 15,714 keys, 138 keys have been found to be related to 6,114 certificates, collected after 2017, related to hosting companies (OVH, GoDaddy.com, Inc., GANDI SAS), security devices (Fortinet Ltd., SonicWALL), wireless appliances (Ruckus Wireless, Inc., D-LINK), storage appliances (EMC Corporation), SSL libraries using custom formats (AlphaSSL), companies related to EV and DV certificate validation schemes (DigiCert Inc), and collaboration servers (Zimbra Collaboration Server).

5.6 Conclusion

This chapter highlighted various significant findings that emerged from our comprehensive analysis and investigation. By gathering a wide range of more than 254 million valid certificates and nearly 170 million RSA keys from various sources throughout several years, we were able to analyze the weak and vulnerable RSA keys.

We examined the cryptographic features of certificates and keys along with validating their persistence in network devices over an extended period regardless of their intended purpose of use. We measured the weakness of RSA keys by their key size along with known vulnerabilities leading to key factorization.

Upon observing the wide presence of duplicated certificates across different sources, we tracked back these certificates to the original hosts from which the certificates were collected. To gain further information on the corresponding network sources, we performed an application security scan using nmap, which enabled us to gather network stack and application stack details. This comprehensive approach allowed us to outline the technical characteristics associated with the services and devices hosting these certificates. We discovered the persistence of weak RSA keys and deprecated signature algorithms especially in legacy embedded network devices relying on shared certificates.

During our analysis, we noted multiple distinct certificates using the same public keys. We investigated not only the presence of vulnerable keys within these shared ones but also their generation process. Using machine learning techniques, we managed to uncover the foundational libraries utilized in producing these keys, thereby raising concerns about the security of the generation procedure of these keys.

These observations collectively contribute to a deeper understanding of cryptographic reuse issues within the study context. Although browser vendors tend to follow the latest standards of cryptographic elements, we showed that deprecated hashing algorithms and non-compliant key sizes are still widespread in embedded network devices. The RSA keys found in malware samples are currently served by TLS hosts and Android apps. Known vulnerabilities remain unpatched on still accessible devices and files.

6 Measuring malicious cryptographic reuse in Android applications

This chapter analyzes the cryptographic file structure and the characteristics of cryptographic components in Android apps along with measuring the compromised trust arising from the reuse of cryptographic components in malicious binaries across various Android apps.

6.1 Overview

While there might be some justifiable cases for inadequate cryptographic settings in certain apps, numerous instances present significant risks to both users and app owners. Our investigation highlights the extent of certificate reuse in Android apps, revealing the prevalent presence of compromised certificates, which necessitates prioritizing the enhancement of cryptographic structures within Android apps to bolster security measures.

Section 6.2 gives details of our collected data from various sources. In section 6.3 the initial analysis is discussed. Section 6.4 reviews the general findings about cryptographic files in Android applications and compares the presence of these elements between benign and malicious applications. Section 6.5 takes a deeper look at the phenomenon of malicious reuse of certificates by examining their characteristics and purpose of usage, along with shared RSA public keys and non-encrypted private keys. Finally, section 6.6 summarizes our findings in this section.

6.2 Collected Data

The insights gained from our analysis of shared certificates and keys (Sections 5.4 and 5.5) motivated us to broaden our approach and gather a larger dataset of APKs.

For our *complementary* set of APKs, we collected a total of 714,616 APK files summarized in Table 6.1. 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 due to Google installation requirements.

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

Table 6.1: 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**

Malshare and VirusShare hashes. We gathered 3,266,932 hash values of malicious binaries from the Malshare Daily Digest[1] (covering the period from September 2017 to July 2023) along with 40,894,458 hash values of the malware samples provided by VirusShare.[2] We further matched our benign set against these hashes. As a result, 247 APK files from benign sources were detected as malicious. As a result, we were left with a set of 259,677 benign apps and 411,596 malicious APK files.

6.3 Initial Analysis

In our complementary 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.

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 6.2). 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 6.3).

Table 6.2: 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

[1] https://malshare.com/daily/
[2] https://virusshare.com/hashes

Table 6.3: 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	194	421,175	*	166,844	254,331
Malicious APKs	789,117	146,329	11,234	5,224,399	4,629,047	*	595,352
Benign APKs	778,260	135,895	11,213	3,766,525	3,256,951	509,574	*
Total	19,648,866	323,506	22,641	9,412,099	11,251 (0.12%) are distinct across sets		

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

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*.

6.4 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 6.4).

Table 6.4: 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%)

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 have been found renamed. The summary of renaming instances is presented in Table 6.5, 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.

Table 6.5: 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

Among the 104,044 files without extensions, 87,072 were identified as being in the "appkey" format. While the remaining formats were distributed 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. 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 (see Table 6.6).

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

A clear difference between these two sets is in the use of *jks* and *bks* files. While both are keystore formats, the use of *bks* format is more popular in malicious apps due to the fact that this keystore format supports a wider range of algorithms and cryptographic capabilities compared to *jks*.

6.5 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 of them being unique instances. As the results in Table 6.3 show, there is significant duplication of certificates within and across sets. Even more interesting is the presence of 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.

Table 6.6: 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

6.5.1 Reuse of signing certificates from malware binaries

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.[3] The previous study by Kim *et al.* [41] 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.

6.5.2 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 certificates makes it challenging to determine the true source

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

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.[4] 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 certificates, 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 6.7. The most widely used certificate is the default certificate of Android Studio, used 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.* [22] back in 2014. Interestingly, after several years we still observe

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

Table 6.7: Use of known and default certificates in apps

Certificate 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	-

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 customers of the Qbiki Networks seems to persist over time.

We were able to detect the presence of a total of 68,962 apps signed with these known key pairs (Table 6.7).

6.5.3 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 define the purpose of the keys when issuing digital certificates through designated fields known as *Key Usage*, *Extended Key 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 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 6.8. The results indicate

Table 6.8: 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

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 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 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 [8]. 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 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 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.

6.5.4 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 6.9 presents the distribution of key size ranges for the public keys extracted from the reused certificates.

Table 6.9: 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

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 [5]. 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 decrypting of protected data or hijacking 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.

6.6 Conclusion

This chapter highlighted the significant prevalence of compromised certificates being reused across Android apps and malicious binaries. By collecting a wide range of more than 700,00 apps from official and alternative markets, we were able to analyze the compromised reuse of code-signing certificates.

Our investigation focused on the file structure of Android applications, particularly examining file formats that may contain cryptographic content. Intriguingly, we found about half of the files had extensions that were renamed. We compared malicious and benign apps in terms of their cryptographic elements and discovered that malicious apps tend to have fewer certificates helping them to evade detection.

Our analysis revealed that 48% (9,412,099) of certificates extracted from malware binaries and our *complementary* set of Android APKs are reused across these two domains. These certificates were reused not only for signing a benign or malicious app but also for purposes beyond code-signing which confirms our previous findings for shared certificates across multiple domains. We observed inadequate context-relevant extensions with unspecified usage can cause this extensive reuse phenomenon. Following the same idea from the previous chapter, we examined the strength of public keys corresponding to shared certificates.

Furthermore, we observed a significant number of signed apps (89%) are treated as unsigned by verification tools due to deprecated settings of their code-signing certificate. We encountered a wide number of apps signed with Android default keys and discovered leaked platform certificates. We were also able to spot benign apps containing known malicious samples. Furthermore, we identified unencrypted private keys embedded in apps, which enabled us to derive their public key and locate the corresponding certificates.

These observations, taken as a whole, serve to enhance our comprehension of the practice of reusing code-signing certificates in the context of file-based sources. While there are valid reasons to argue that certain applications might not require robust cryptographic settings, such as when they lack critical or identifiable information, the compromised use of an application's identity still gives rise to security concerns in the majority of scenarios.

7 Discussion and Recommendations

This chapter discusses our findings offering valuable insights into the security landscape surrounding cryptographic practices in both PKI and Android environments. In light of our research, we propose several recommendations to enhance the security of cryptographic components in these ecosystems to ensure the integrity of cryptographic key usage.

7.1 Discussion

The presence of duplicated cryptographic keys is an alarming indicator when it deviates from established security guidelines. It implies a potential absence of adequate security protocols. during the key generation and management, resulting in unreliable keys prone to exploitation. This duplication may stem from various reasons, e.g., errors in the key generation algorithm or weaknesses in the underlying library used for key generation. Regardless of the cause, the presence of duplicates raises red flags and casts doubt on the overall security of the PKI ecosystem. The results derived through our analysis shed light on:

The persistently weak state of RSA keys

Our analysis extends over a considerable length of time, spanning up to nine years in certain datasets (Rapid7 and Malware) which gives us a unique historical view of RSA key security. While numerous previous studies have suggested improvements in the quality of RSA keys, our analysis reveals a different outcome. For example, we factored 185,731 unique moduli corresponding to 793,694 RSA keys (181,784 unique). In 2012, Heninger *et al.* [31] reported finding divisors for 2,314 moduli for 16,717 distinct public keys, and in 2016, Hastings *et al.* [30] factored 313,000 RSA keys.

Despite conducting our analysis almost 10 years after the initial report, we still observed significantly higher numbers, which is a disheartening outcome. The comprehensive and large-scale nature of our study, however, gives a more realistic view security state of the PKI ecosystem.

Shared RSA keys are weak

We found that 87% (14,862,767) shared keys appear in distinct certificates, out of which we found that 44% (6,595,972) are weak in terms of key length and 1.3% were factored due to weak prime number selection. We also found that these shared keys were predominantly generated using outdated and vulnerable libraries, resulting in indicating inherently weak and vulnerable keys. We also noticed that 92% (19,172) of all shared certificates among devices were found to be non-compliant with the NIST standards.

TLS certificates are widely reused for different purposes

Aside from the presence of vulnerable cryptographic attributes, another factor contributing to the weaknesses in the PKI is the utilization of certificates beyond their intended scope. Certificates issued for a specific purpose should not be used for any other usage. In reality, the certificates are being used interchangeably. Almost all certificates found in malicious binaries, 8,299,100 (99.9%), were served by TLS hosts in the Rapid7 set. 2,338,672 (28%) of certificates used in malicious binaries are also used for signing Android apps.

Overwhelmingly high use of obsolete algorithms among shared certificates

95% (9,560,681) of the shared certificates are signed using deprecated hashing algorithms, which were already considered obsolete by the time of our certificate collection. Such a significant presence of non-compliant certificates points to the bigger problem within the PKI ecosystem.

Weak hashing algorithms can undermine the integrity and authenticity of TLS certificates and allow attackers to create fraudulent certificates leading to impersonation and unauthorized access. It becomes easier for attackers to tamper with the certificate data without detection. This compromises the trustworthiness of the certificates and opens the door to various security risks, such as man-in-the-middle attacks or the interception of sensitive information. Finally, weak hashing algorithms hinder the long-term security and validity of TLS certificates. As cryptographic attacks evolve and computational power increases, weak hashing algorithms become even more susceptible to brute-force and collision attacks.

The shared certificates are predominantly used by legacy embedded network devices

Many shared certificates appear to be served by legacy devices. Over half of validated shared certificates (55%) come from devices that pre-date our data collection period. The rest are served by devices released in 2014. All these certificates are currently in use and this fact emphasizes the lack of oversight regarding the certificates that have been already issued.

7.2 Observations and Recommendations

Our analysis highlights several observations that underline the existing problems and enables us to propose the following potential countermeasures:

Public key-relevant improvements

While the solution to the weak key situation appears to be simple—enforcing strong key generation—many cryptographic libraries, such as OpenSSL, still allow for the generation of weak keys essentially weakening the PKI environment. On a different note, even though the certificate community (CA/B Forum) appears to be moving towards reduced longevity of certificates, the lack of enforcement for revocation of legacy certificates and re-issuance for devices using them leads to continued use of these certificates and consequently weakens

the security of the PKI. We suggest the adoption of current cryptographic parameters as suggested by NIST would ensure wide compatibility.

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 (28.5%) certificates 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 (26.3%) certificates of malware binaries along with 63,285 (4.03%) 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. This practice can be followed in specific scenarios where a trusted organization requires the capability to generate new certificates autonomously, such as within secure corporate networks or controlled environments. However, caution should be exercised, especially in publicly distributed applications, where assigning such authority could lead to potential security vulnerabilities and exploitation by malicious actors. For instance, if a malicious actor gains control over a signing certificate with CA privileges, they could potentially create compromised certificates, enabling man-in-the-middle attacks and data interception. Careful consideration is essential to create a balance between operational convenience and maintaining a robust security posture.

- *Mandated purpose-related extensions:* Only 4,730,060 (24%) certificates mandate verifiers to process the purpose-related components of certificates. If a verifier lacks support for critical extensions, it can safely ignore such extensions 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

Implementing prevention mechanisms is a beneficial security practice that minimizes risks. It may serve as a simple solution to vet apps, such as:

- *Avoiding the use of default or publicly known certificates:* 49,323 apps in our set were signed with Android's default certificate and 19,639 apps 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

Algorithm 1 An algorithm for verifying signing certificate

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

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

It holds true that upon publishing an application through an official marketplace, the marketplace assumes responsibility for tasks such as identity verification and the update process. In the case of Android applications that are self-signed, the Android operating system takes charge of signature verification. When a user downloads an update for an application, the Android runtime compares the signature of the new version with that of the original. If the signatures align, the Android runtime proceeds with the installation of the update. However, it is always considered a safe approach to incorporate a more customized verification process to address potential unforeseen security issues.

- *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.

  To begin, developers should retrieve the hash value of the expected code-signing certificate. Subsequently, within the initial procedure of the application, incorporate a check that involves programmatically calculating the hash value of the current signing certificate. Finally, a successful match from comparing the calculated hash value with the expected hash value obtained earlier indicates that the app is signed with the expected certificate and can proceed to run. Otherwise, it suggests a potential security breach or tampering. This process enhances app security by validating certificate authenticity and thwarting unauthorized modifications. It also guards against impersonation and enables swift breach detection for timely mitigation.

  This process serves as a strong protection by providing an added layer of protection through the

validation of an APK's authenticity and integrity. It prevents unauthorized modifications to the app and reduces the risk of running a compromised version with a potentially harmful or altered certificate. This mechanism ensures early detection of any discrepancies in the signing certificate, maintaining its integrity and alignment 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.

Use of our proposed platform

The analysis conducted in our study highlights the extent of sharing certificates and keys across the PKI ecosystem. While individual CAs may not have a comprehensive view of the Internet, our developed platform can be utilized for identifying instances of certificate reuse.

Moreover, Owners and issuers of certificates and keys should give careful attention to the excessive use of duplicated keys. Our study provides a valuable tool for identifying vulnerabilities in cryptographic implementations, and we hope it will be utilized by the owners and issuers of the certificates and keys.

7.3 Conclusion

Digital certificates play a crucial role in the security of network communications and application developments, but there are hosting organizations and application developers who prioritize convenience over security. While there are legitimate scenarios for the reuse of cryptographic components, in numerous instances, it poses substantial risks to users and app owners.

Our contributions encompass a comprehensive and extensive endeavor in the context of Internet security. We have conducted a broad analysis of TLS/SSL certificates and RSA keys, spanning domains and platforms. In the realm of domains, we embarked on the most comprehensive scan, measuring and characterizing the distribution of TLS/SSL certificates and RSA public key sharing. This exhaustive investigation covered over 314 million valid certificates and 13 million SSH keys, carefully collected from diverse sources over an extended period, some spanning up to nine years. Similarly, in the realm of Android applications, our analysis extended to over 19 million certificates and 9 million reused keys found across APKs and malware binaries, gathered from various sources over as many as eleven years.

Our findings emphasize the critical necessity for robust mechanisms to identify and analyze duplicate and weak certificates and keys. To facilitate such essential analysis, we have developed a correlation platform capable of pinpointing duplicate keys and identifying vulnerabilities within TLS certificates and RSA keys. We have made this platform, openly accessible to the public through https://key-explorer.com/. Moreover,

to further enable the analysis of cryptographic reuse, we have made the set of reused certificates available to the public.

7.4 Future Work

Derived from our analysis, we offer a set of interesting research paths intended to empower security practitioners in elevating the overall cryptographic security landscape. Our work highlights how important it is to continue observing and keep working on making the digital world safer. We hope that this research will urge organizations and developers to reassess the current security practices. Prioritizing certificate security is crucial for safeguarding their data, hosting platform, apps, and users. As a result, the approach outlined here opens the door to potential research endeavors:

- **Assessing the context of all reused certificates:** This study revealed a noteworthy quantity of certificates and keys that had been reused, with potential associations to IP addresses, domain names, and organizations. As a next step, examining the context of each reused certificate could be a promising direction to explore.
- **Developing a policy engine:** This study has demonstrated that the absence of certificates with contextually relevant definitions creates opportunities to be compromised. To address this issue, the proposal to develop a policy engine capable of interpreting the specific needs of a given environment while adhering to the latest NIST recommendations and recommending optimal certificate settings offers a promising avenue for diminishing certificate misuse.
- **Enhancing certificate inventory management for Android apps:** Similar to how browser policies evaluate digital certificates for authenticity on websites, Android devices can improve the capability to authenticate the credibility and dependability of certificates presented by apps. This validation procedure guarantees that the certificates used to sign the apps align with essential minimum criteria and characteristics.
- **Analyzing the traits of compromised certificates:** This work showed extensive reuse of certificates within malware binaries. Consequently, due to the lack of an established point of reference for querying compromised certificates and keys, giving attention to looking closely at the characteristics of these certificates would be a new path to explore. This exploration can uncover potential recurring patterns, offering insights that can contribute to the development of a precise detection platform.

References

[1] Kevin Allix, Tegawendé F. Bissyandé, Jacques Klein, and Yves Le Traon. AndroZoo: Collecting Millions of Android Apps for the Research Community. In *Proceedings of the 13th International Conference on Mining Software Repositories*, page 468–471. Association for Computing Machinery, 2016.

[2] Johanna Amann, Oliver Gasser, Quirin Scheitle, Lexi Brent, Georg Carle, and Ralph Holz. Mission Accomplished? HTTPS Security after Diginotar. In *Proceedings of the 2017 Internet Measurement Conference*, page 325–340. Association for Computing Machinery, 2017.

[3] Nimrod Aviram, Sebastian Schinzel, Juraj Somorovsky, Nadia Heninger, Maik Dankel, Jens Steube, Luke Valenta, David Adrian, J. Alex Halderman, Viktor Dukhovni, Emilia Käsper, Shaanan Cohney, Susanne Engels, Christof Paar, and Yuval Shavitt. DROWN: Breaking TLS Using SSLv2. In *Proceedings of the 25th USENIX Conference on Security Symposium*, page 689–706. USENIX Association, 2016.

[4] Elaine Barker. Recommendation for Key Management: Part 1. Technical report, National Institute of Standards and Technology, 2020.

[5] Elaine Barker, Lily Chen, Allen Roginsky, Apostol Vassilev, Richard Davis, and Scott Simon. Recommendation for pair-wise key establishment using integer factorization cryptography. Technical report, National Institute of Standards and Technology, 2019.

[6] Luciano Bello. DSA-1571-1 OpenSSL—Predictable random number generator, 2008.

[7] Benjamin Beurdouche, Karthikeyan Bhargavan, Antoine Delignat-Lavaud, Cédric Fournet, Markulf Kohlweiss, Alfredo Pironti, Pierre-Yves Strub, and Jean Karim Zinzindohoue. A Messy State of the Union: Taming the Composite State Machines of TLS. In *Proceedings of the 2015 IEEE Symposium on Security and Privacy*, pages 535–552. Institute of Electrical and Electronics Engineers, 2015.

[8] Sharon Boeyen, Stefan Santesson, Tim Polk, Russ Housley, Stephen Farrell, and David Cooper. Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. RFC 5280, 2008.

[9] Alexandre Braga and Ricardo Dahab. Mining Cryptography Misuse in Online Forums. In *Proceedings of the 2016 IEEE International Conference on Software Quality, Reliability and Security Companion*, pages 143–150. Institute of Electrical and Electronics Engineers, 2016.

[10] Enrico Branca, Farzaneh Abazari, Ronald Rivera Carranza, and Natalia Stakhanova. Origin Attribution of RSA Public Keys. In *Proceedings of the Security and Privacy in Communication Networks*, pages 374–396. Springer International Publishing, 2021.

[11] Frank Cangialosi, Taejoong Chung, David Choffnes, Dave Levin, Bruce M. Maggs, Alan Mislove, and Christo Wilson. Measurement and Analysis of Private Key Sharing in the HTTPS Ecosystem. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, page 628–640. Association for Computing Machinery, 2016.

[12] Alexia Chatzikonstantinou, Christoforos Ntantogian, Georgios Karopoulos, and Christos Xenakis. Evaluation of Cryptography Usage in Android Applications. In *Proceedings of the 9th EAI International Conference on Bio-Inspired Information and Communications Technologies*, page 83–90. Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering, 2016.

[13] Chronicle. Abusing Code Signing for Profit, 2019.

[14] Taejoong Chung, Yabing Liu, David Choffnes, Dave Levin, Bruce MacDowell Maggs, Alan Mislove, and Christo Wilson. Measuring and Applying Invalid SSL Certificates: The Silent Majority. In *Proceedings of the 2016 Internet Measurement Conference*, page 527–541. Association for Computing Machinery, 2016.

[15] Andrei Costin, Jonas Zaddach, Aurélien Francillon, and Davide Balzarotti. A Large-Scale Analysis of the Security of Embedded Firmwares. In *Proceedings of the 23rd USENIX Conference on Security Symposium*, page 95–110. USENIX Association, 2014.

[16] Chad R. Dougherty. MD5 vulnerable to collision attacks, 2008.

[17] Zakir Durumeric, James Kasten, Michael Bailey, and J. Alex Halderman. Analysis of the HTTPS Certificate Ecosystem. In *Proceedings of the 2013 Conference on Internet Measurement Conference*, page 291–304. Association for Computing Machinery, 2013.

[18] Zakir Durumeric, Frank Li, James Kasten, Johanna Amann, Jethro Beekman, Mathias Payer, Nicolas Weaver, David Adrian, Vern Paxson, Michael Bailey, and J. Alex Halderman. The Matter of Heartbleed. In *Proceedings of the 2014 Conference on Internet Measurement Conference*, page 475–488. Association for Computing Machinery, 2014.

[19] Zakir Durumeric, Eric Wustrow, and J. Alex Halderman. ZMap: Fast Internet-Wide Scanning and Its Security Applications. In *Proceedings of the 22nd USENIX Conference on Security*, page 605–620. USENIX Association, 2013.

[20] Manuel Egele, David Brumley, Yanick Fratantonio, and Christopher Kruegel. An Empirical Study of Cryptographic Misuse in Android Applications. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer and Communications Security*, page 73–84. Association for Computing Machinery, 2013.

[21] Adam Everspaugh, Yan Zhai, Robert Jellinek, Thomas Ristenpart, and Michael Swift. Not-So-Random Numbers in Virtualized Linux and the Whirlwind RNG. In *Proceedings of the 2014 IEEE Symposium on Security and Privacy*, pages 559–574. Institute of Electrical and Electronics Engineers, 2014.

[22] Sascha Fahl, Sergej Dechand, Henning Perl, Felix Fischer, Jaromir Smrcek, and Matthew Smith. 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*, page 1143–1155. Association for Computing Machinery, 2014.

[23] Nicolas Falliere, Liam Murchu, and Eric Chien. W32.Stuxnet Dossier, 2010.

[24] Dennis Felsch, Martin Grothe, Jörg Schwenk, Adam Czubak, and Marcin Szymanek. The Dangers of Key Reuse: Practical Attacks on IPsec IKE. In *Proceedings of the 27th USENIX Conference on Security Symposium*, page 567–583. USENIX Association, 2018.

[25] Mario Frustaci, Pasquale Pace, Gianluca Aloi, and Giancarlo Fortino. Evaluating Critical Security Issues of the IoT World: Present and Future Challenges. In *Proceedings of the IEEE Internet of Things Journal*, pages 2483–2495. Institute of Electrical and Electronics Engineers, 2018.

[26] Jun Gao, Pingfan Kong, Li Li, Tegawendé F. Bissyandé, and Jacques Klein. Negative Results on Mining Crypto-API Usage Rules in Android Apps. In *Proceedings of the 2019 IEEE/ACM 16th International Conference on Mining Software Repositories*, pages 388–398. Institute of Electrical and Electronics Engineers, 2019.

[27] Oliver Gasser, Ralph Holz, and Georg Carle. A deeper understanding of SSH: Results from Internet-wide scans. In *Proceedings of the 2014 IEEE Network Operations and Management Symposium (NOMS)*, pages 1–9. Institute of Electrical and Electronics Engineers, 2014.

[28] Kaspar Hageman, Álvaro Feal, Julien Gamba, Aniketh Girish, Jakob Bleier, Martina Lindorfer, Juan Tapiador, and Narseo Vallina-Rodriguez. Mixed Signals: Analyzing Software Attribution Challenges in the Android Ecosystem. In *Proceedings of the IEEE Transactions on Software Engineering*, pages 2964–2979. Institute of Electrical and Electronics Engineers, 2023.

[29] Mike Hanley. GitHub security update: revoking weakly-generated SSH keys, 2021.

[30] Marcella Hastings, Joshua Fried, and Nadia Heninger. Weak Keys Remain Widespread in Network Devices. In *Proceedings of the 2016 Internet Measurement Conference*, pages 49–63. Association for Computing Machinery, 2016.

[31] Nadia Heninger, Zakir Durumeric, Eric Wustrow, and J. Alex Halderman. Mining Your Ps and Qs: Detection of Widespread Weak Keys in Network Devices. In *Proceedings of the 21st USENIX Security Symposium*, pages 205–220. USENIX Association, 2012.

[32] Ralph Holz, Johanna Amann, Olivier Mehani, Matthias Wachs, and Mohamed Ali Kaafar. TLS in the Wild: An Internet-wide Analysis of TLS-based Protocols for Electronic Communication. In *Proceedings of the 2016 Network and Distributed System Security Symposium*, pages 1–15. Internet Society, 2016.

[33] Ralph Holz, Lothar Braun, Nils Kammenhuber, and Georg Carle. The SSL Landscape: A Thorough Analysis of the x.509 PKI Using Active and Passive Measurements. In *Proceedings of the 2011 ACM SIGCOMM Conference on Internet Measurement Conference*, page 427–444. Association for Computing Machinery, 2011.

[34] Russ Housley, Tim Polk, and Lawrence E. Bassham III. Algorithms and Identifiers for the Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. RFC 3279, 2002.

[35] Man Hong Hue, Joyanta Debnath, Kin Man Leung, Li Li, Mohsen Minaei, M. Hammad Mazhar, Kailiang Xian, Endadul Hoque, Omar Chowdhury, and Sze Yiu Chau. All Your Credentials Are Belong to Us: On Insecure WPA2-Enterprise Configurations. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, page 1100–1117. Association for Computing Machinery, 2021.

[36] Liz Izhikevich, Renata Teixeira, and Zakir Durumeric. LZR: Identifying Unexpected Internet Services. In *Proceedings of the 30th USENIX Security Symposium*, pages 3111–3128. USENIX Association, 2021.

[37] Adam Janovsky, Matus Nemec, Petr Svenda, Peter Sekan, and Vashek Matyas. Biased RSA Private Keys: Origin Attribution of GCD-Factorable Keys. In *Proceedings of the Computer Security - ESORICS 2020*, pages 505–524. Springer International Publishing, 2020.

[38] Burt Kaliski and Jessica Staddon. PKCS #1: RSA Cryptography Specifications Version 2.0. RFC 2437, 1998.

[39] Hyunjae Kang, Jae wook Jang, Aziz Mohaisen, and Huy Kang Kim. AndroTracker: Creator Information based Android Malware Classification System. In *Proceedings of the Information Security Applications-15th International Workshop*, pages 1–12. Semantic Scholar, 2014.

[40] Sheharbano Khattak, David Fifield, Sadia Afroz, Mobin Javed, Srikanth Sundaresan, Vern Paxson, Steven Murdoch, and Damon Mccoy. Do You See What I See? Differential Treatment of Anonymous Users. pages 1–15. Internet Society, 2016.

[41] Doowon Kim, Bum Jun Kwon, and Tudor Dumitraş. 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*, page 1435–1448. Association for Computing Machinery, 2017.

[42] Arjen K. Lenstra, James P. Hughes, Maxime Augier, Joppe W. Bos, Thorsten Kleinjung, and Christophe Wachter. Ron was wrong, Whit is right. In *Proceedings of the IACR Cryptol ePrint Archive*, pages 64–81. Internet Society, 2012.

[43] Juanru Li, Zhiqiang Lin, Juan Caballero, Yuanyuan Zhang, and Dawu Gu. K-Hunt: Pinpointing Insecure Cryptographic Keys from Execution Traces. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, page 412–425. Association for Computing Machinery, 2018.

[44] Xia Li, Jiajun Jiang, Samuel Benton, Yingfei Xiong, and Lingming Zhang. A Large-scale Study on API Misuses in the Wild. In *Proceedings of the 2021 14th IEEE Conference on Software Testing, Verification and Validation*, pages 241–252. Institute of Electrical and Electronics Engineers, 2021.

[45] Yabing Liu, Will Tome, Liang Zhang, David Choffnes, Dave Levin, Bruce Maggs, Alan Mislove, Aaron Schulman, and Christo Wilson. An End-to-End Measurement of Certificate Revocation in the Web's PKI. In *Proceedings of the 2015 Internet Measurement Conference*, page 183–196. Association for Computing Machinery, 2015.

[46] Chris M. Lonvick and Tatu Ylonen. The Secure Shell (SSH) Transport Layer Protocol. RFC 4253, 2006.

[47] Wilfried Mayer, Aaron Zauner, Martin Schmiedecker, and Markus Huber. No Need for Black Chambers: Testing TLS in the E-mail Ecosystem at Large. In *Proceedings of the 2016 11th International Conference on Availability, Reliability and Security*, pages 10–20. Institute of Electrical and Electronics Engineers, 2016.

[48] Kerry A. McKay and David A. Cooper. Guidelines for the Selection, Configuration, and Use of Transport Layer Security (TLS) Implementations. Technical report, National Institute of Standards and Technology, 2019.

[49] Ariana Mirian, Zane Ma, David Adrian, Matthew Tischer, Thasphon Chuenchujit, Tim Yardley, Robin Berthier, Joshua Mason, Zakir Durumeric, J. Alex Halderman, and Michael Bailey. An Internet-wide view of ICS devices. In *Proceedings of the 2016 14th Annual Conference on Privacy, Security and Trust*, pages 96–103. Institute of Electrical and Electronics Engineers, 2016.

[50] Ildar Muslukhov, Yazan Boshmaf, and Konstantin Beznosov. Source Attribution of Cryptographic API Misuse in Android Applications. In *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*, page 133–146. Association for Computing Machinery, 2018.

[51] Matus Nemec, Marek Sys, Petr Svenda, Dusan Klinec, and Vashek Matyas. The Return of Coppersmith's Attack: Practical Factorization of Widely Used RSA Moduli. In *Proceedings of the 24th ACM Conference on Computer and Communications Security*, pages 1631–1648. Association for Computing Machinery, 2017.

[52] NIST. Research Results on SHA-1 Collisions . Technical report, National Institute of Standards and Technology, 2017.

[53] NIST. Digital Signature Standard (DSS). Technical report, National Institute of Standards and Technology, 2023.

[54] Darren Pauli. Shodan boss finds 250,000 routers have common keys, 2015.

[55] Luca Piccolboni, Giuseppe Di Guglielmo, Luca P. Carloni, and Simha Sethumadhavan. CRYLOGGER: Detecting Crypto Misuses Dynamically. In *Proceedings of the 2021 IEEE Symposium on Security and Privacy*, pages 1972–1989. Institute of Electrical and Electronics Engineers, 2021.

[56] Android Open Source Project. APK Signature Scheme v4, 2022.

[57] Sazzadur Rahaman, Ya Xiao, Sharmin Afrose, Fahad Shaon, Ke Tian, Miles Frantz, Murat Kantarcioglu, and Danfeng (Daphne) Yao. 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*, page 2455–2472. Association for Computing Machinery, 2019.

[58] Eric Rescorla. HTTP Over TLS. RFC 2818, 2000.

[59] Eric Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446, 2018.

[60] Kaspersky Lab's Global Research and Analysis Team. The Duqu 2.0 persistence module, 2015.

[61] Nayanamana Samarasinghe and Mohammad Mannan. Short Paper: TLS Ecosystems in Networked Devices vs. Web Servers. In *Proceedings of the Financial Cryptography and Data Security: 21st International Conference*, page 533–541. Springer-Verlag, 2023.

[62] Dennis Shefanovski and Serguei Leontiev. Using the GOST R 34.10-94, GOST R 34.10-2001, and GOST R 34.11-94 Algorithms with the Internet X.509 Public Key Infrastructure Certificate and CRL Profile. RFC 4491, 2006.

[63] Murugiah Souppaya, William Haag, Mehwish Akram, William Barker, Rob Clatterbuck, Brandon Everhart, Brian Johnson, Alexandros Kapasouris, Dung Lam, Brett Pleasant, Mary Raguso, Susan Symington, Paul Turner, Clint Wilson, and Donna Dodson. Securing Web Transactions TLS Server Certificate Management. Technical report, National Institute of Standards and Technology, 2020.

[64] Drew Springall, Zakir Durumeric, and J. Alex Halderman. FTP: The Forgotten Cloud. In *Proceedings of the 2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pages 503–513. Institute of Electrical and Electronics Engineers, 2016.

[65] Marc Stevens, Alexander Sotirov, Jacob Appelbaum, Arjen Lenstra, David Molnar, Dag Arne Osvik, and Benne de Weger. Short Chosen-Prefix Collisions for MD5 and the Creation of a Rogue CA Certificate. In *Proceedings of the Cryptology ePrint Archive*, pages 1–17. Cryptology ePrint Archive, 2009.

[66] Petr Svenda, Matus Nemec, Peter Sekan, Rudolf Kvasnovsky, David Formanek, David Komarek, and Vashek Matyas. The Million-Key Question—Investigating the Origins of RSA Public Keys. In *Proceedings of the 25th USENIX Security Symposium*, pages 893–910. USENIX Association, 2016.

[67] Sean Turner. MD2 to Historic Status. RFC 6149, 2011.

[68] Anna-Katharina Wickert, Lars Baumgärtner, Michael Schlichtig, Krishna Narasimhan, and Mira Mezini. To Fix or Not to Fix: A Critical Study of Crypto-misuses in the Wild. In *Proceedings of 2022 IEEE International Conference on Trust, Security and Privacy in Computing and Communications*, pages 1–8. Institute of Electrical and Electronics Engineers, 2022.

[69] Scott Yilek, Eric Rescorla, Hovav Shacham, Brandon Enright, and Stefan Savage. When Private Keys Are Public: Results from the 2008 Debian OpenSSL Vulnerability. In *Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement*, page 15–27. Association for Computing Machinery, 2009.

[70] Li Zhang, Jiongyi Chen, Wenrui Diao, Shanqing Guo, Jian Weng, and Kehuan Zhang. CryptoREX: Large-scale Analysis of Cryptographic Misuse in IoT Devices. In *Proceedings of the 22nd International Symposium on Research in Attacks, Intrusions and Defenses*, pages 151–164. USENIX Association, 2019.

[71] Zicheng Zhang, Wenrui Diao, Chengyu Hu, Shanqing Guo, Chaoshun Zuo, and Li Li. 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*, page 144–154. Association for Computing Machinery, 2020.