# Encryption workarounds for Android

*Skerdi Basha*

Thesis submitted in partial fulfillment of the requirements for the

*Masters' of Science degree in Computer Science and Engineering*

University of Crete
School of Sciences and Engineering
Computer Science Department
Voutes University Campus, 700 13 Heraklion, Crete, Greece

Thesis Supervisor: Prof. *Evangelos Markatos*

Thesis Co-Advisor: Dr. *Harry Manifavas*

This work has been performed at the University of Crete, School of Sciences and Engineering, Computer Science Department.

The work has been supported by the Foundation for Research and Technology - Hellas (FORTH), Institute of Computer Science (ICS).

This work has been performed as part of a project supported by the European Union's Internal Security Fund - Police (ISFP) programme under grant agreement No 101038738.

University of Crete
Computer Science Department

**Encryption workarounds for Android**

Thesis submitted by
**Skerdi Basha**
in partial fulfillment of the requirements for the
Masters' of Science degree in Computer Science

THESIS APPROVAL

Author: ______________________
Skerdi Basha

Committee approvals: ______________________
Evangelos Markatos
Professor, Thesis Supervisor

______________________
Harrys Manifavas
Thesis Co-Advisor

______________________
Yannis Tzitzikas
Professor, Committee Member

______________________
Kostas Magoutis
Associate Professor, Committee Member

Departmental approval: ______________________
Polyvios Pratikakis
Associate Professor, Director of Graduate Studies

Heraklion, July 2023

# Encryption workarounds for Android

## Abstract

Encryption on Android involves the process of encoding user data both on the device and during transit using either symmetric or asymmetric encryption keys. When a device is encrypted, all data created by the user is automatically encrypted before being stored on the disk, and decryption occurs automatically when the data is accessed by an authorized process. The purpose of encryption is to ensure that unauthorized parties cannot read the data even if they gain access to it.

Android offers two methods of device encryption: file-based encryption and full-disk encryption. File-based encryption, introduced in Android 7.0, allows different files to be encrypted with different keys that can be unlocked independently. On the other hand, full-disk encryption has been supported since Android 4.4, but as of Android 10, it is being phased out in favor of file-based encryption.

Over the years more and more encryption enabled Android devices and applications have been used by criminals to carry out illegal activities or to cover their traces, making it difficult for forensic investigations to search for digital evidence. As such, encryption has become an increasing concern for LEAs and a wide range of encryption bypassing tools are being used to perform investigations on evidence found in crime scenes.

According to some studies, encryption bypassing is categorized into six categories: find a copy of the key, guess the key, compel the key from someone who knows it, exploit a flaw, access plaintext when the device is in use, locate a plaintext copy of the encrypted artifact.

In this thesis we introduce a tool with Android encryption bypassing capabilities offering logical extraction analysis and decryption of multiple widely used Android applications such as Signal, Wickr and WeChat. Our tool also offers password brute-forcing capabilities and enables the orchestration of phishing attacks aimed at the lockscreen. Furthermore we document the capabilities and shortcomings of each of the techniques implemented in the tool, while also listing the improvements developers could use to fortify their applications and the habits users should follow to protect their data stored in Android devices and applications.

From the experience we gained while implementing the tool we conclude that while using encryption does not ensure the invincibility of the system, our techniques are also not always effective. Their success often depends on conditions related to the inner encryption implementation of the system and actions that where performed by the user prior to the acquisition of the evidence.

# Παράκαμψη κρυπτογράφησης για Android

## Περίληψη

Η κρυπτογράφηση στο Android περιλαμβάνει τη διαδικασία κωδικοποίησης των δεδομένων του χρήστη τόσο στη συσκευή όσο και κατά τη μεταφορά, χρησιμοποιώντας συμμετρικά ή ασύμμετρα κλειδιά κρυπτογράφησης. Όταν μια συσκευή είναι κρυπτογραφημένη, όλα τα δεδομένα που δημιουργούνται από τον χρήστη κρυπτογραφούνται αυτόματα πριν αποθηκευτούν στο δίσκο και η αποκρυπτογράφηση γίνεται αυτόματα όταν υπάρξει πρόσβαση στα δεδομένα απο μια εξουσιοδοτημένη διαδικασία. Ο σκοπός της κρυπτογράφησης είναι να διασφαλίσει ότι χωρίς εξουσιοδότηση κανείς δεν μπορεί να διαβάσει τα κρυπτογραφημένα δεδομένα ακόμη και αν αποκτήσει πρόσβαση σε αυτά.

Το Android προσφέρει δύο μεθόδους κρυπτογράφησης συσκευών: κρυπτογράφηση μεμονωμένων αρχείων και πλήρης κρυπτογράφηση δίσκου. Η κρυπτογράφηση μεμονωμένων αρχείων, που εισήχθη στο Android 7.0, επιτρέπει διαφορετικά αρχεία να κρυπτογραφούνται με διαφορετικά κλειδιά που μπορούν να ξεκλειδωθούν ανεξάρτητα. Από την άλλη πλευρά, η κρυπτογράφηση πλήρους δίσκου υποστηρίζεται από το Android 4.4, αλλά από το Android 10, καταργείται σταδιακά υπέρ της κρυπτογράφησης μεμονωμένων αρχείων.

Με τα χρόνια όλο και περισσότερες συσκευές και εφαρμογές Android με δυνατότητα κρυπτογράφησης έχουν αρχίσει να χρησιμοποιούνται από εγκληματίες για την άσκηση παράνομων δραστηριοτήτων ή για την κάλυψη των ιχνών τους, καθιστώντας δυσκολότερη την αναζήτηση ψηφιακών πειστηρίων κατά τη διάρκεια των ψηφιακών εγκληματολογικών ερευνών.

Ως εκ τούτου, η κρυπτογράφηση έχει γίνει μια αυξανόμενη ανησυχία για τις αστυνομικές αρχές και ένα ευρύ φάσμα εργαλείων παράκαμψης κρυπτογράφησης χρησιμοποιείται για την εκτέλεση ερευνών πάνω σε στοιχεία που βρέθηκαν σε σκηνές εγκλήματος.

Σύμφωνα με ορισμένες μελέτες, η παράκαμψη της κρυπτογράφησης κατηγοριοποιείται σε έξι γενικές κατηγορίες: βρείτε το κλειδί, μαντέψτε το κλειδί, αναγκάστε τη παράδοση του κλειδιού, εκμεταλλευτείτε ένα ελάττωμα στη κρυπτογράφηση, αποκτήστε πρόσβαση σε μη κρυπτογραφημένο μήνυμα όταν η συσκευή χρησιμοποιείται, εντοπίστε ένα μη κρυπτογραφημένο αντίγραφο του μηνύματος.

Σε αυτή την εργασία παρουσιάζουμε ένα εργαλείο με δυνατότητες παράκαμψης κρυπτογράφησης σε Android. Το εργαλείο επιτρέπει την αποκρυπτογράφηση πολλαπλών ευρέως χρησιμοποιούμενων εφαρμογών Android όπως Signal, Wickr και WeChat, τα οποία ανιχνεύονται σε ένα logical extraction. Το εργαλείο μας επίσης προσφέρει δυνατότητες brute-forcing κωδικών κλειδώματος οθόνης και επιτρέπει την ενορχήστρωση phishing επιθέσεων με στόχο την οθόνη κλειδώματος. Επιπλέον τεκμηριώνουμε τις δυνατότητες και τις ελλείψεις καθεμιάς από τις τεχνικές που εφαρμόζονται στο εργαλείο. Απαριθμούμε επίσης τις βελτιώσεις που θα μπορούσαν να χρησιμοποιήσουν οι προγραμματιστές για να ενισχύσουν τις εφαρμογές τους και τις συνήθειες που πρέπει να ακολουθούν οι χρήστες για να προστατεύουν τα αποθηκευμένα δεδομένα τους σε συσκευές και εφαρμογές Android.

Από την εμπειρία που αποκτήσαμε κατά την δημιουργία του εργαλείου συμπεραίνουμε ότι ενώ η χρήση κρυπτογράφησης δεν διασφαλίζει πάντα την ασφάλεια ενός συστήματος, οι τεχνικές παράκαμψης κρυπτογράφησης επίσης δεν είναι πάντα αποτελεσματικές. Η επιτυχία τους εξαρτάται συχνά από συνθήκες που σχετίζονται με των τρόπο κρυπτογράφησης του συστήματος και ενέργειες που πραγματοποιήθηκαν από τον χρήστη πριν από την απόκτηση των στοιχείων.

# Ευχαριστίες

*Στην οικογένεια μου, στους καθηγητές μου και στους φίλους μου*

# Contents

# List of Tables

# List of Figures

# Chapter 1

# Introduction

Encryption has become an essential component of our society. The encryption market and the technology surrounding it have flourished, protecting everything we use day to day, such as financial transactions, in-vehicle information, health data and private communications. Over the years almost all of our online communications and transactions have been more and more secure thanks to the implementation of secure encryption practices which enable us to privately communicate and distribute assets on the internet.

However, our personal data is not only at risk when it is traveling over the internet. Despite our best efforts, there is a chance that our devices will be stolen, lost or infected with malware. If this is the case, an unauthorized individual or organization may gain access to sensitive data on our device. File encryption is an important aspect of data security. By encrypting files on a device, sensitive data remains protected even if unauthorized access to the device occurs. File encryption ensures that the contents of the files are securely encrypted and can only be accessed by authorized individuals who possess the decryption key or password.

When files are encrypted, they are transformed into a scrambled format that is unintelligible without the correct decryption key. This adds an additional layer of security to sensitive information stored on devices, such as personal documents, financial data, or private communications. Even if someone gains physical access to the device or manages to bypass other security measures, they would still need the encryption key or password to decrypt and access the files.

Although this technology plays an integral role in our lives, it also has some bad applications. The use of encryption-enabled devices and applications by criminals presents challenges to law enforcement agencies when it comes to decrypting investigation-related information.

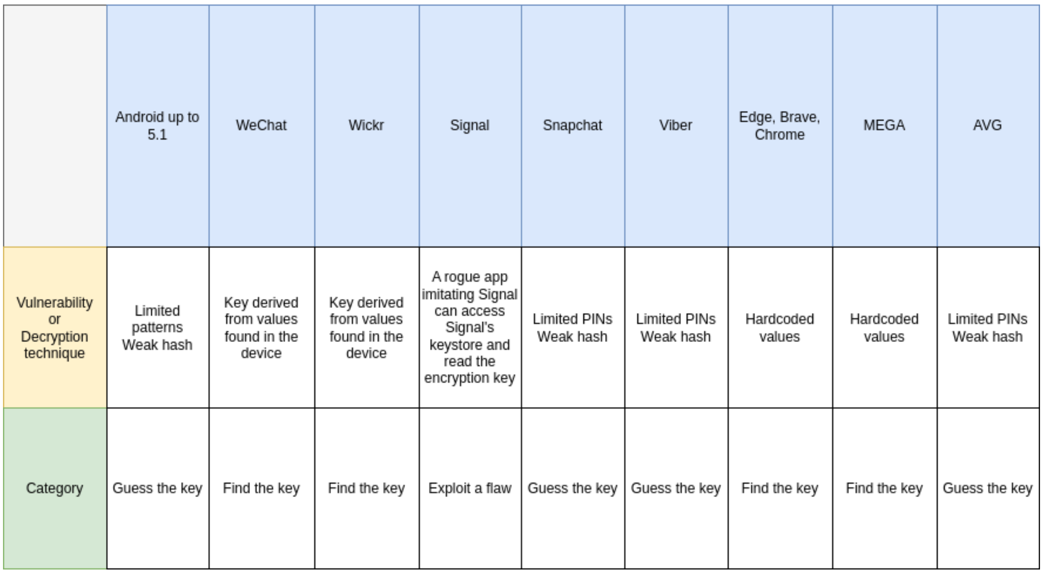
Secure messaging applications like Telegram and Signal have gained popularity due to their strong encryption and privacy features. These applications use end-to-end encryption, which ensures that only the communicating parties can access the content of the messages, making it difficult for anyone, including law enforcement, to intercept or decrypt the messages in transit.

Since in most countries it is unlawful for the state to compel a suspect to hand over the access key or the data in an unencrypted format [21], many law enforcement agencies proceed their investigations by using forensic tools or techniques that could help them bypass the encryption mechanism used by the perpetrators of the crimes under investigation. As a contribution to the work that is being done assisting law enforcement agencies in their investigations, we have gathered and documented a wide range of encryption bypassing techniques and assembled them in a tool which could be used for educational purposes while also being a helpful addition in the toolchain of a forensic investigator.

Accompanied by the created tool, in this theses we list different cases where encryption can be bypassed and we demonstrate the techniques that are used to do so. For each technique we show figures of the aforementioned tool and explain the usefulness of the technique in a forensic investigation, while also listing defense mechanisms against it.

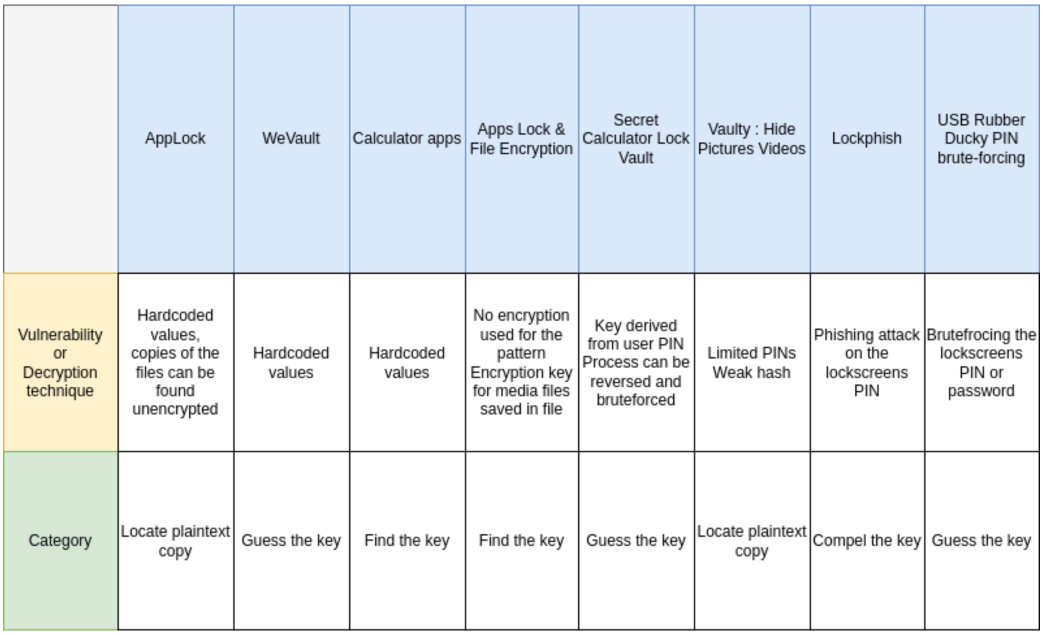
In the following chapter we explain some basic concepts of cryptographic primitives relative to Android security and Android forensics. In chapter 3 we introduce our tool and explain its multiple features and options. In the chapters 4 and 5 we explain the techniques we studied during our research and how we implemented them on our tool. In chapter 6 we present some other tools similar to ours and we compare the capabilities of those tools to the capabilities of our tool. In chapter 7 we list the features that could be introduced in future iterations of our tool and talk about the conclusions we made from our findings.

In Figure 1.1 shown below we list the decryption techniques we studied and the category each of them belongs to [74].



| | Android up to 5.1 | WeChat | Wickr | Signal | Snapchat | Viber | Edge, Brave, Chrome | MEGA | AVG |
|---|---|---|---|---|---|---|---|---|---|
| Vulnerability or Decryption technique | Limited patterns<br>Weak hash | Key derived from values found in the device | Key derived from values found in the device | A rogue app imitating Signal can access Signal's keystore and read the encryption key | Limited PINs<br>Weak hash | Limited PINs<br>Weak hash | Hardcoded values | Hardcoded values | Limited PINs<br>Weak hash |
| Category | Guess the key | Find the key | Find the key | Exploit a flaw | Guess the key | Guess the key | Find the key | Find the key | Guess the key |

(a) Table 1



| | AppLock | WeVault | Calculator apps | Apps Lock & File Encryption | Secret Calculator Lock Vault | Vaulty : Hide Pictures Videos | Lockphish | USB Rubber Ducky PIN brute-forcing |
|---|---|---|---|---|---|---|---|---|
| Vulnerability or Decryption technique | Hardcoded values, copies of the files can be found unencrypted | Hardcoded values | Hardcoded values | No encryption used for the pattern<br>Encryption key for media files saved in file | Key derived from user PIN<br>Process can be reversed and bruteforced | Limited PINs<br>Weak hash | Phishing attack on the lockscreens PIN | Brutefrocing the lockscreens PIN or password |
| Category | Locate plaintext copy | Guess the key | Find the key | Find the key | Guess the key | Locate plaintext copy | Compel the key | Guess the key |

(b) Table 2

Figure 1.1: Techniques studied

# Chapter 2

# Background

In this chapter we will present and analyze the basics of what someone needs to know related to Android security, cryptographic primitives and the processes followed during the a forensic investigation of mobile devices.

## 2.1 The Android Operating System

Android is primarily developed for touchscreen mobile devices like smartphones and tablets, serving as a mobile operating system. It utilizes a modified version of the Linux kernel and incorporates various open-source software. Initially developed by Android Inc., which was bought in 2005 by Google, it is now being developed by a developer consortium known as the Open Handset Alliance [2], in which Google is part of. It was first announced in November 2007 and since then there have been many major releases of the operating system with various features being added in over the years.

The fundamental foundation of the operating system is commonly referred to as the Android Open Source Project (AOSP) [36], which is freely available and falls under the category of open-source software (FOSS). It is predominantly licensed under the Apache License. However, the majority of devices utilize a customized, proprietary version of Android created by Google. These devices come pre-installed with additional proprietary closed-source software. Many of these devices feature customized user interfaces and software packages developed by vendors, such as Samsung's TouchWiz and later One UI, as well as HTC's Sense [43]. Other competing ecosystems and AOSP forks exist, including Amazon's Fire OS, OPPO's ColorOS, Vivo's OriginOS, Honor's MagicUI, and custom ROMs like LineageOS.

Android has maintained its position as the best-selling operating system for smartphones globally since 2012, as indicated by Figure 2.1 [80]. It has also been the leading operating system for tablets since 2013 [83]. Android is widely utilized in nearly all countries and holds the largest market share in many of them, like India, where it surpasses 95% [79]. As of January 2022, Android boasted more than three billion monthly active users [11]. Additionally, as of March 2023, the Google Play Store hosted an extensive collection of over 2.6 million applications [81].

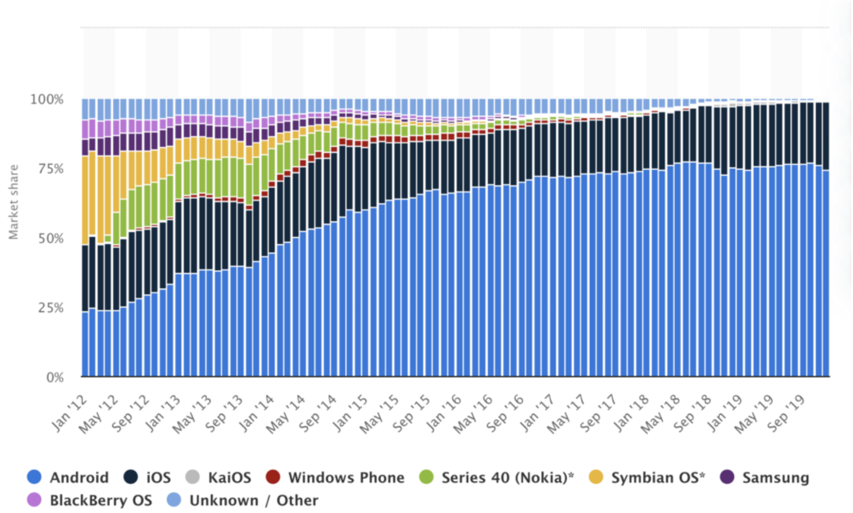


Figure 2.1: Android market share from 2012 to 2019

### 2.1.1 Filesystem Structure

The storage on Android devices is separated into several partitions, such as /system/ for the OS, and /data/ for installed applications and user data [69]. Installed applications specifically store their data inside their folder /data/data, where each application gets its own directory that is named using the application identifier (com.company.example). Application specific data can also be stored in the /sdcard/Android/data directory (like the /data/data directory).

Similar to the internal application-data directory, each application is allocated its exclusive directory that is accessible only to that specific application. Nevertheless, all the contents of the emulated SD card can be accessed and altered via USB. This encompasses the application data stored in /sdcard/Android/data as well.

Starting with Android 4.4 Kit Kat, the ability for user-installed applications to have shared writing access to the /sdcard/ directory was restricted. Instead, applications were limited to write data only in dedicated directories within Android/data/ that corresponded to their respective package names. However, with the release of Android 5 Lollipop, this restriction was lifted through the introduction of the Google Storage Access Framework interface, which was not backward-compatible [1].

In contrast to typical desktop Linux distributions, Android devices do not grant root access to the operating system by default. Sensitive partitions like /system/

are read-only, and other partitions like /data/ are inaccessible to users without root access. However, root access can be acquired by exploiting security vulnerabilities in Android. The open-source community often exploits these vulnerabilities to enhance device capabilities and customization options, but it also presents an opportunity for malicious actors to install viruses and malware. Certain devices, including most Google Pixel and OnePlus models, offer an OEM unlocking option that allows users to unlock the bootloader, thereby granting root access. It's important to note that the unlocking process resets the system to its factory state, erasing all user data [38].

### 2.1.2 Encryption

Android provides two methods for device encryption: file-based encryption [4] and full-disk encryption [5]. The choice of encryption method depends on the Android version and device capabilities.

Full-disk encryption was supported in Android versions 5.0 up to 9.0. It uses a single key, protected by the user's device password, to encrypt the entire user data partition. This means that the user needs to provide their credentials during boot before any part of the disk is accessible. While this provides strong security, it also means that certain core functionalities of the phone, such as alarms and accessibility services, are not immediately available after a device reboot.

Starting from Android 7.0, file-based encryption was introduced as an alternative method. File-based encryption allows different files to be encrypted with different keys, which can be unlocked independently. This allows for a more granular and flexible approach to encryption. Devices that support file-based encryption can also benefit from `Direct Boot`, which enables quick access to important device features, like accessibility services and alarms, straight from the lock screen. With file-based encryption, apps can operate within a limited context even before the user has provided their credentials, ensuring the protection of private user information.

In Android 9, metadata encryption was introduced where hardware support is available. Metadata encryption protects certain content that is not encrypted by file-based encryption, such as directory layouts, file sizes, permissions, and creation/modification times. The key used for metadata encryption is protected by `Keymaster`, which is responsible for managing cryptographic keys in Android. `Keymaster` itself is protected by `Verified Boot`, which ensures the integrity of the boot process and guards against tampering with the device's software.

These encryption methods and mechanisms aim to provide strong security for user data on Android devices while balancing usability and access to core functionalities.

## 2.2 Cryptography

Here we are going to give a short introduction about certain cryptographic primitives. Cryptography is the science of encryption. At least, that is how it began. It is now much broader, covering authentication, digital signatures, and many other security functions.

Modern cryptography is heavily rooted in mathematical theory and computer science. Cryptographic algorithms are designed based on computational hardness assumptions, which make them extremely difficult for adversaries to break in practical scenarios. While theoretically it is possible to breach a well-designed system, it is infeasible to do so in practice. Therefore, such schemes are referred to as "computationally secure." These designs require continuous reevaluation and potential adaptation due to theoretical advancements (e.g., improvements in integer factorization algorithms) and the development of faster computing technology.

The proliferation of cryptographic technology has given rise to various legal considerations. Governments, concerned about cryptography's potential for espionage and subversion, have classified it as a weapon and imposed limitations or outright bans on its use and export. Cryptography also plays a significant role in digital rights management and copyright infringement disputes concerning digital media.

### 2.2.1 Symmetric Cryptography

Symmetric cryptography relies on symmetric-key algorithms, which employ the same key for both the encryption of plaintext and the decryption of ciphertext, as depicted in Figure 2.2. The keys can either be identical or have a straightforward transformation to convert between them. In practice, these keys represent a shared secret between two or more parties, allowing them to maintain a private information channel. One significant limitation of symmetric-key encryption, compared to public-key encryption (also known as asymmetric-key encryption), is that both parties must have access to the same secret key. However, symmetric-key encryption algorithms are generally more efficient for bulk encryption tasks. They have smaller key sizes, requiring less storage space and facilitating faster transmission.

Symmetric-key encryption can use either stream ciphers or block ciphers.

- Stream ciphers encrypt the bits (typically in bytes) of a message one at a time.

- Block ciphers take a number of bits and encrypt them in a single unit, padding the plaintext to achieve a multiple of the block size.

In symmetric-key algorithms, the sender and recipient of a message must share the same secret key. In early cryptographic systems, this required physically secure channels to transmit the secret key between parties.

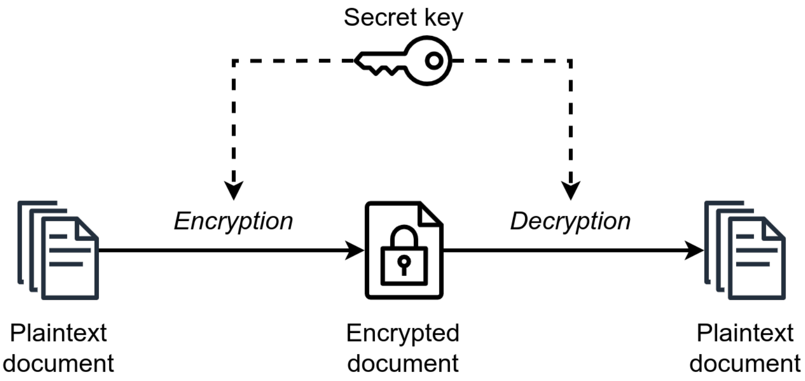


Figure 2.2: Symmetric-key encryption

Modern cryptographic systems still use symmetric-key algorithms for encrypting the majority of messages. However, they eliminate the need for physically secure channels by employing techniques like Diffie-Hellman key exchange [34] or other public-key protocols. These methods securely establish a fresh new secret key for each session or conversation, thus overcoming the key distribution challenge.

Pseudorandom key generators are commonly used with asymmetric ciphers for key transfer. It is crucial for the generators and their initialization vectors to have sufficient randomness, as their lack of randomness in the past has led to cryptanalytic vulnerabilities. Implementations should rely on high-entropy sources for initialization.

Symmetric ciphers are not limited to encryption, they can be used for other cryptographic purposes. To ensure the integrity of the ciphertext, a message authentication code is often added. Hash functions can also be constructed from block ciphers.

Historically, symmetric ciphers have been vulnerable to known-plaintext attacks, chosen-plaintext attacks, differential cryptanalysis, and linear cryptanalysis [19]. However, careful design of the cipher functions in each round can significantly reduce the chances of a successful attack. Increasing the key length or the number of rounds in the encryption process can enhance security but comes at the cost of increased processing power and decreased speed due to the additional computational operations required.

The most commonly used symmetric encryption algorithm nowadays is AES. AES was selected by the U.S. National Institute of Standards and Technology (NIST) in 2001 [64] as a replacement for the aging Data Encryption Standard (DES) [73].

AES operates on fixed-size blocks of data (128 bits), with three key sizes: 128 bits, 192 bits, and 256 bits. The algorithm consists of a series of transformations, including substitution, permutation, and mixing operations, applied in multiple rounds. These operations are designed to provide a high level of security and resistance against various cryptanalysis techniques.

### 2.2.2 Asymmetric Cryptography

Asymmetric cryptography, also called public-key cryptography, is a field of cryptography that utilizes pairs of interconnected keys. Each pair consists of a public key and a corresponding private key. These key pairs are created using cryptographic algorithms based on one-way functions. The security of public-key cryptography relies on maintaining the secrecy of the private key, while the public key can be freely shared without compromising security.

In a public-key encryption system, anyone possessing the public key can encrypt a message, to create a ciphertext. However, only individuals with the corresponding private key can decrypt the ciphertext and retrieve the original message, as shown in Figure 2.3.

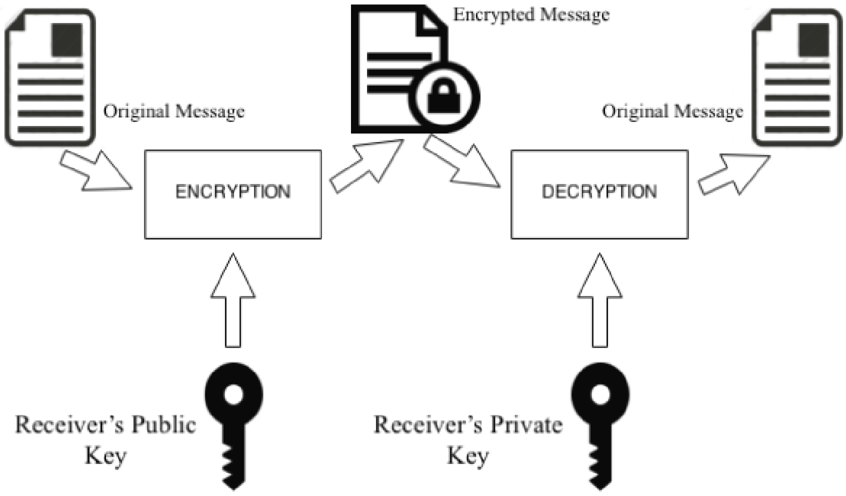


Figure 2.3: Asymmetric-key encryption

One of the key advantages of public key algorithms is their ability to provide key distribution and secrecy through protocols like Diffie-Hellman key exchange. This allows two parties to establish a shared secret key over an insecure channel without the need for a pre-shared secret. It enables secure communication between parties who have never communicated before.

Public key algorithms are also used in digital signatures, which are essential for ensuring the authenticity and integrity of electronic messages and documents. Digital signatures use the private key to sign a message, and the corresponding public key is used to verify the signature. This provides a way to prove the identity of the sender and detect any tampering or modifications of the message.

However, asymmetric encryption algorithms are generally slower than symmetric encryption algorithms [85]. Asymmetric encryption involves more complex mathematical operations, which require more computational resources. This makes asymmetric encryption less efficient for bulk encryption of large amounts of data. As a result, public key algorithms are often used in combination with symmetric encryption. For instance, a symmetric key can be exchanged using an asymmetric algorithm, and then the subsequent communication can be encrypted using a symmetric encryption algorithm.

Two of the best-known uses of public key cryptography are:

- Public key encryption: A message is encoded using the public key of the intended recipient. When suitable algorithms are employed and utilized correctly, it becomes practically impossible for anyone without the corresponding private key to decrypt the message. The possession of the private key assures ownership and association with the public key. This mechanism is employed to guarantee the confidentiality of a message.
- Digital signatures: A message is signed using the private key of the sender and can be validated by anyone who possesses the sender's public key. This verification serves as evidence that the sender had access to the private key, strongly suggesting their association with the corresponding public key. It also demonstrates that the signature was specifically generated for that particular message, as verification would fail for any other message that doesn't employ the private key.

Digital signature schemes serve as a means of sender authentication in cryptographic systems. They are used in non-repudiation systems to ensure that the originator of a document or communication cannot later deny their involvement. Digital signatures form the foundation for various applications, including digital cash, password-authenticated key agreement, time-stamping services, and non-repudiation protocols.

Like any security-related system, it is crucial to identify potential weaknesses. In addition to the risk of selecting an inadequate asymmetric key algorithm or using a key length that is too short, the primary security concern is the compromise of the private key in a key pair. If the private key becomes known to an unauthorized party, the security of messages, authentication, and other cryptographic operations will be compromised.

In theory, all public key schemes are vulnerable to brute-force key search attacks [68]. However, these attacks are impractical when the computational effort required exceeds the capabilities of potential attackers. Another potential security vulnerability is the man-in-the-middle attack [65], where an attacker intercepts the communication of public keys and modifies them to substitute different public keys. To execute this attack, encrypted messages and responses must be intercepted, decrypted, and re-encrypted by the attacker using the appropriate public keys for different communication segments to avoid detection.

One approach to mitigate such attacks is the use of a public key infrastructure (PKI). A PKI encompasses a set of roles, policies, and procedures necessary for the creation, management, distribution, usage, storage, and revocation of digital certificates. It also facilitates public-key encryption and provides a framework for secure and trusted communication [13].

### 2.2.3 Cryptographic Hash Functions

Cryptographic hash functions are mathematical algorithms used to transform an input message of any length into a fixed-length output called a hash or a message digest as shown in Figure 2.4. These functions are designed to be one-way and irreversible, meaning it is difficult or practically impossible to reconstruct the original message from its hash. They are primarily used to provide message integrity and authenticity, which are essential properties in secure communication protocols.

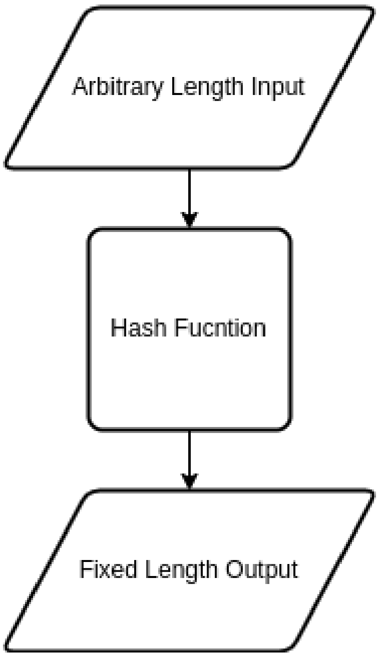


Figure 2.4: Cryptographic hash functions

A good cryptographic hash function should have two main properties: first, it should be deterministic, meaning that the same input always produces the same output hash and second, it should provide collision resistance, meaning that it should be computationally infeasible for an attacker to find two different messages that result in the same hash value.

The security of cryptographic hash functions is essential in applications such as digital signatures, password storage, and data authentication. In digital signature applications, a hash function is used to generate a unique message digest that can be signed by the sender using their private key. The recipient can then use the sender's public key to verify the signature and ensure the integrity and authenticity of the message.

Several cryptographic hash functions have been developed over the years, with varying degrees of security and practicality. Some popular examples include the SHA family [63] and the MD family [50]. As computing power and cryptanalysis techniques continue to advance, researchers constantly evaluate and improve hash function designs to enhance their security and practicality.

## 2.3 Mobile Device Forensics

Mobile device forensics refers to a branch of digital forensics that focuses on retrieving digital evidence or data from a mobile device while adhering to forensically sound practices, which means that a number of specific practices have to be followed during the collection, preservation, and analysis of digital evidence to ensure its reliability and admissibility in a court of law. Some of the key conditions that have to be considered during a investigation, for maintenance of forensic soundness, are listed below.

- Integrity: The integrity of the evidence must be preserved throughout the entire process, from collection to analysis. This includes ensuring that the evidence remains unaltered and protected against unauthorized access or tampering.

- Chain of Custody: A proper chain of custody must be established and maintained for all digital evidence. This involves documenting the location, possession, and movement of the evidence, along with the names of individuals who handled it, to establish its authenticity and prevent contamination.

- Documentation: Thorough documentation is crucial in maintaining forensic soundness. All actions taken during the investigation, including the collection, handling, and analysis of evidence, should be meticulously documented to provide a clear record of the processes followed.

- Authenticity: The authenticity of the evidence must be established and maintained. This involves using validated and reliable tools and techniques to collect and analyze the evidence, ensuring that the results obtained are accurate and can be verified.

- Reproducibility: The methods and procedures used to collect and analyze digital evidence should be reproducible. This means that another examiner should be able to replicate the same results using the same tools and techniques, which enhances the reliability and credibility of the evidence.

- Confidentiality: Sensitive information and evidence should be handled with utmost confidentiality and in compliance with relevant laws and regulations. Access to the evidence should be restricted to authorized personnel only, and measures should be in place to protect the privacy of individuals involved.

- Expertise: Digital forensic investigations should be conducted by qualified and experienced professionals who possess the necessary skills and knowledge to handle and analyze digital evidence accurately. Continuous training and staying updated with the latest techniques and tools are essential for maintaining forensic soundness.

The term "mobile device" typically refers to mobile phones, but it can encompass any digital device that possesses internal memory and communication capabilities. This includes devices such as PDAs, GPS devices, and tablet computers.

Mobile devices serve as repositories for various types of personal information, such as contacts, photos, calendars, notes, SMS and MMS messages. Smartphones, in particular, may also store video files, email messages, web browsing history, location data, and social networking communications and contacts.

With the advancement of mobile device technology, the volume and diversity of data stored on these devices continue to expand. Potential evidence that can be extracted from a mobile phone can originate from multiple sources, including the device's internal memory, SIM card, and attached memory cards like SD cards. Traditional mobile phone forensics primarily focused on retrieving SMS and MMS messages, call logs, contact lists, and phone identification information (IMEI/ESN). Also, nowadays smartphones store a broader array of data, encompassing web browsing history, wireless network configurations, geolocation details (e.g. geotags from image metadata), emails, and various forms of multimedia content derived from internet-based platforms.

There is growing need for mobile forensics due to several reasons and some of the prominent reasons are:

- Use of mobile phones to store and transmit personal and corporate information
- Use of mobile phones in online transactions
- Activities like hacking, identity theft, cyberbullying, and online fraud often involve the use of mobile devices

### 2.3.1 Challenges

Mobile device forensics poses several challenges, both in terms of evidence and technical aspects. These challenges arise due to the dynamic nature of the mobile device industry, where original equipment manufacturers (OEMs) frequently introduce changes to mobile phone form factors, operating system file structures, data storage methods and even pin connectors and cables. As a result, forensic examiners must adapt their forensic processes to address these variations, which differ from the procedures employed in computer forensics.

One challenge is keeping up with the rapid advancements in storage capacity. Mobile devices are increasingly becoming mini-computer-like devices with higher processing power and larger storage capabilities. This means that the amount of data that can be stored on a mobile device is constantly increasing, requiring forensic examiners to handle larger volumes of data during investigations.

Also nowadays, mobile devices often employ strong encryption and security measures to protect user data. These measures can make it difficult to access and extract data from locked or encrypted devices without the necessary credentials or technical expertise.

Many mobile applications store sensitive data and implement privacy mechanisms to protect user information. These privacy measures can make it challenging to access and extract relevant data, especially without the user's cooperation or access to app-specific credentials.

Mobile forensic investigations must adhere to legal and privacy regulations. The proper acquisition and handling of evidence, ensuring the preservation of the chain of custody, and respecting the privacy rights of individuals involved are critical challenges in mobile forensics.

### 2.3.2 Tools

The complexity of mobile device forensics, coupled with the diverse landscape of devices and operating systems, has prompted the development of numerous tools for extracting evidence from such devices. It is crucial to recognize that there is no universal tool or method capable of acquiring all evidence from every type of device. As a result, forensic examiners, particularly those aspiring to be recognized as expert witnesses in court, should undergo extensive training to understand the capabilities and limitations associated with each tool and method that is being used during a forensic investigation.

Forensic examiners need to be well-versed in how different tools and methods acquire evidence, maintain forensic soundness, and meet legal requirements, such as the Daubert [75] or Frye standards [76]. These standards establish the criteria for the admissibility of scientific evidence in court proceedings and ensure that the methods used are reliable and scientifically valid.

In the early stages of mobile device investigations, manual analysis was conducted live on the device, with examiners taking photographs or writing down relevant information as evidence. However, this approach had drawbacks as it risked modifying the device's content and limited access to proprietary parts of the operating system. Specialized forensic photography equipment, such as the Fernico ZRT [22], was sometimes used to overcome these limitations.

In recent years, a range of hardware and software tools have emerged to aid in the recovery of logical and physical evidence from mobile devices. These tools typically consist of both hardware components (such as cables for connecting the device to the acquisition machine) and software components for extracting and, in some cases, analyzing the evidence.

Currently the 3 main types of data extractions techniques for mobile devices are, logical extraction, physical extraction and chip-off extraction. Logical extraction, involves retrieving data that is readily accessible from the operating system of the mobile device. It typically includes data such as call logs, contacts, messages, emails, and app data. Physical extraction, involves acquiring a bit-by-bit copy of the entire storage of the mobile device, including system files, deleted data, and unallocated space. It provides a more comprehensive set of data, including deleted files and system-level information. Chip-off extraction, is utilized in cases where other extraction methods are not feasible. It involves physically removing the memory chip from the device's circuit board and reading its contents using specialized equipment.

Mobile forensics tools that perform extractions and analyze the extracted data, include Belkasoft X [9], Cellebrite UFED [12], Oxygen Forensic Detective [28], Elcomsoft Mobile Forensic Bundle [20], MOBILEdit Forensic Express [58], Magnet Axiom [27] and MSAB XAMN [59].

# Chapter 3

# ALPS: Android Lock Picking System

`ALPS` is a configurable and extendable tool for Android mobile forensics specifically designed for encrypted artifacts analysis and encryption bypassing techniques. It is developed for Linux using Python 3 and Flet [26], which is a Python library that enables us to develop Flutter applications using Python.

ALPS provides us the option to choose between 3 tasks from its starting screen as shown in Figure 3.1.

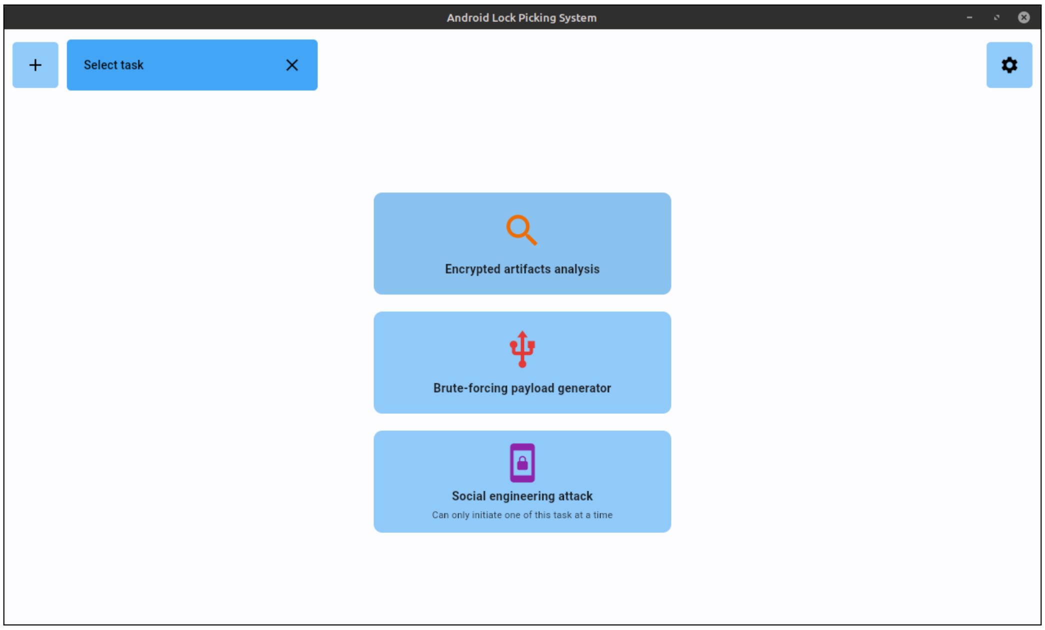

Android Lock Picking System
+
Select task
Encrypted artifacts analysis
Brute-forcing payload generator
Social engineering attack
Can only initiate one of this task at a time

Figure 3.1: ALPS home screen

- Encrypted artifacts analysis
- Brute-forcing payload generator
- Social engineering attack

## 3.1 Encrypted Artifacts Analysis

`Encrypted artifacts analysis` enables us to perform a logical extraction analysis on data received from a device. Through the `Encrypted artifacts analysis` panel shown in Figure 3.2, users can load a logical extraction and the tool will search for encrypted artifacts and list the artifacts that it could decrypt, as well as the keys and methods used for decrypting them.

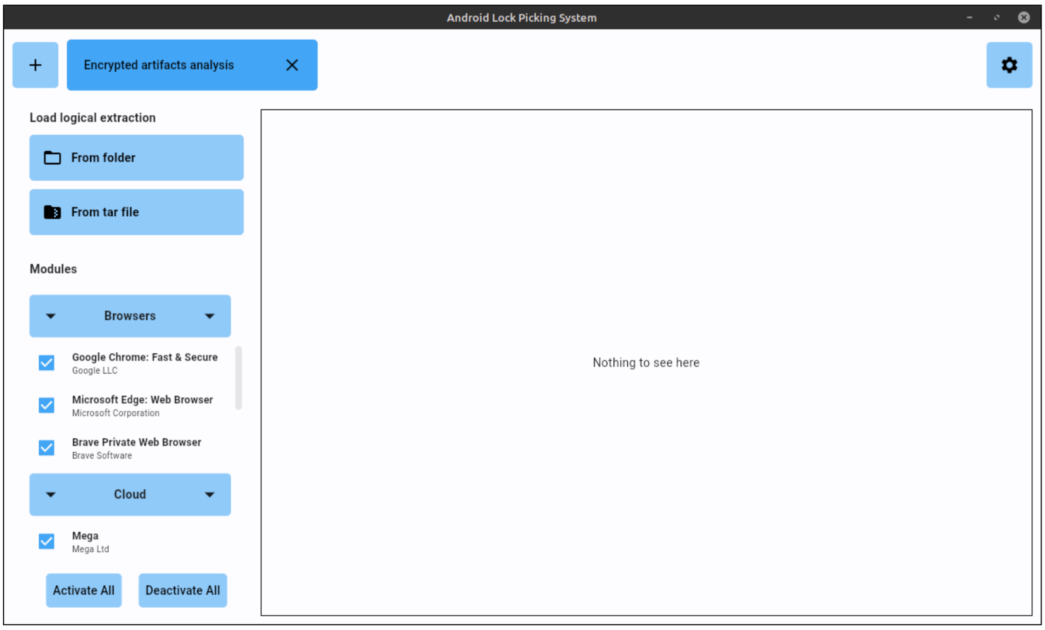
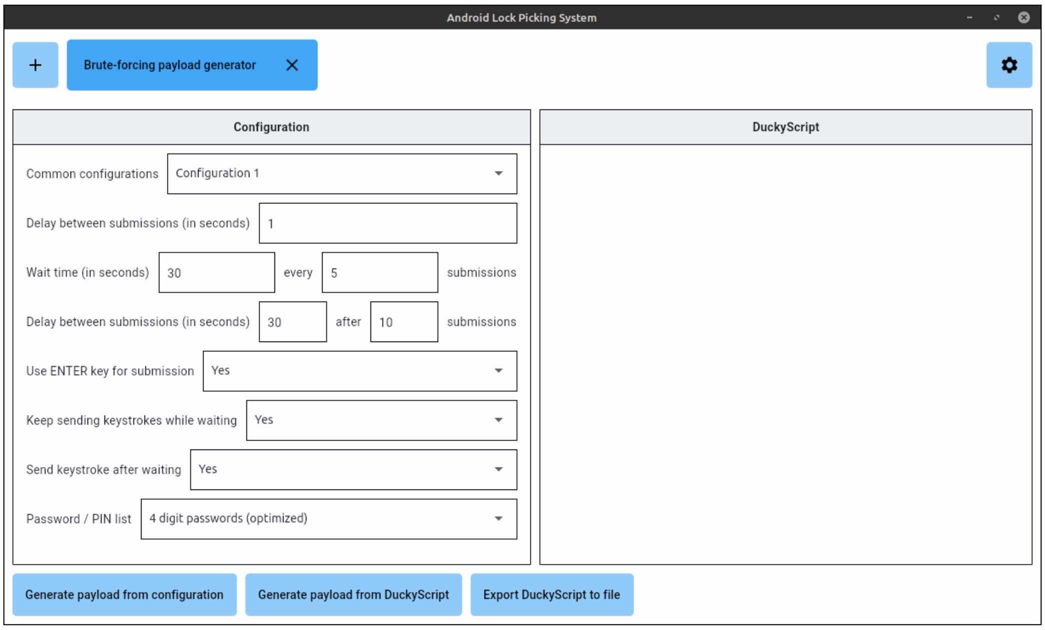


Figure 3.2: Encrypted artifacts analysis panel

## 3.2 Brute-forcing Payload Generator

`Brute-forcing payload generator` enables us to create brute-forcing payloads for a USB Rubber Ducky [41], which can be used to brute-force PINs or passwords. Through the `Brute-forcing payload generator` panel shown in Figure 3.3, users can generate a USB Rubber Ducky payload using predefined or custom entries for the delays or wordlist used, they can modify the generated payloads DuckyScript code and can also save it as text for future usage.


Android Lock Picking System
+
Brute-forcing payload generator
Configuration
DuckyScript
Generate payload from configuration
Generate payload from DuckyScript
Export DuckyScript to file

Common configurations: Configuration 1

Delay between submissions (in seconds): 1

Wait time (in seconds) 30 every 5 submissions

Delay between submissions (in seconds) 30 after 10 submissions

Use ENTER key for submission: Yes

Keep sending keystrokes while waiting: Yes

Send keystroke after waiting: Yes

Password / PIN list: 4 digit passwords (optimized)

Figure 3.3: Brute-forcing payload generator panel

## 3.3 Social Engineering Attack

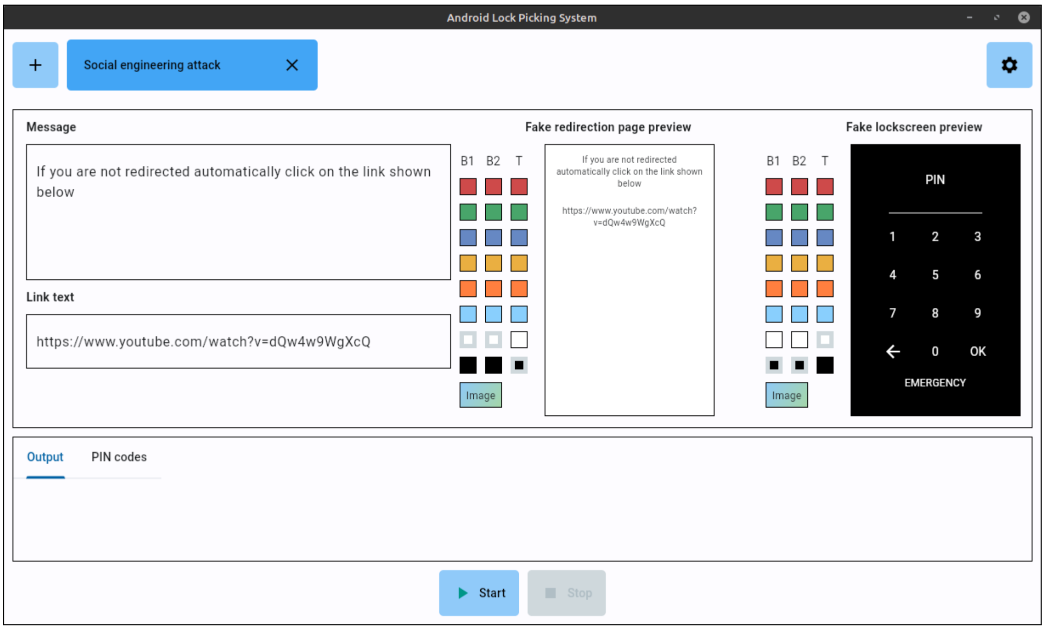
`Social engineering attack` enables us to orchestrate a social engineering attack specifically designed for acquiring mobile device PINs. Through the `Social engineering attack` panel shown in Figure 3.4 users can create a phishing page and generate a link pointing to it. Through the same panel they can view any data acquired using that attack.

## 3.4 Other Tool Features and Settings

ALPS uses a tabbed view for its panels, which means that users are able to have multiple tabs open and perform tasks parallelly, the only exception being the `Social engineering attack` task, which allows only one panel being open for that task at any time.

ALPS is configurable through a `config.json` configuration file and through its settings dedicated panel as shown in Figure 3.5. Any change is the configuration is displayed in any new tab opened during that session and future sessions, without the need to restart the tool. The values that can be configured through the configuration file or the settings panel are shown below.

- `ngrok_api_key: value used for the API key of the ngrok service [60]`
- `logging_enabled: defines if logging is enabled`



Message

If you are not redirected automatically click on the link shown below

Link text

https://www.youtube.com/watch?v=dQw4w9WgXcQ

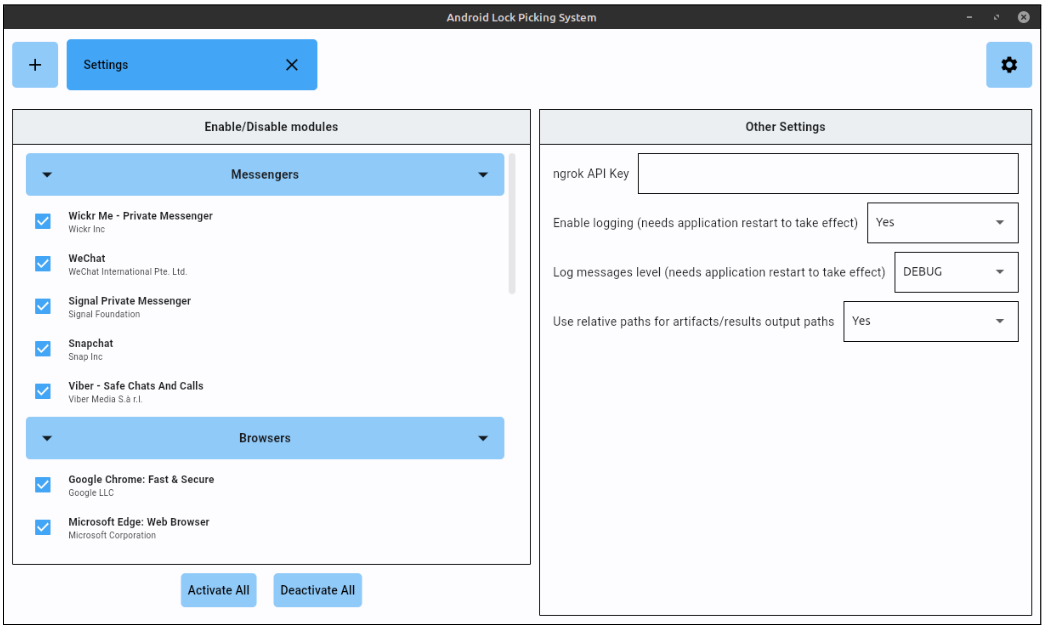
Figure 3.4: Social engineering attack panel

- `log_level`: defines the logging level, it can be `DEBUG`, `INFO` or `WARNING`
- `use_relative_paths`: defines if relative or absolute paths should be displayed in the encrypted artifacts analysis panel
- `modules`: contains the modules used for the encrypted artifacts analysis

Any output data or files created as a result of user actions using the tool, is saved in a folder named `out` which is created in the same directory as the one were we run our application from.

The tool also provides different levels of logging which can be configured, or disabled entirely through the setting panel or the configuration file. If logging is enabled log entries are output both in the console and in the session's log file. Log files are saved in the output folder of the tool. A log file is created for each session, the name of the log file is set as the time when the session started using the format `"day_month_year_hour_minutes_seconds"`. The log entries for both the console and the log files are using the following format `"time filename level message"`.

More information related to the installation and usage of the tool can be found in Appendix B.



Enable/Disable modules

Messengers

- ☑ Wickr Me - Private Messenger  
  Wickr Inc
- ☑ WeChat  
  WeChat International Pte. Ltd.
- ☑ Signal Private Messenger  
  Signal Foundation
- ☑ Snapchat  
  Snap Inc
- ☑ Viber - Safe Chats And Calls  
  Viber Media S.à r.l.

Browsers

- ☑ Google Chrome: Fast & Secure  
  Google LLC
- ☑ Microsoft Edge: Web Browser  
  Microsoft Corporation

Other Settings

ngrok API Key

Enable logging (needs application restart to take effect) Yes

Log messages level (needs application restart to take effect) DEBUG

Use relative paths for artifacts/results output paths Yes

Figure 3.5: ALPS settings panel

# Chapter 4

# Attacks on the Lock Screen

Encryption in Android is based on user-authentication-gated cryptographic keys [3] for both full-disk and file-based encryption. That means that a third party with knowledge of the authentication inputs provided for the encryption of contents stored on a mobile device, could completely decrypt and a have a look at the raw contents inside the device if given access to the device.

Android gives the option for PIN, pattern, password and biometric authentications such as fingerprint. Over the years biometric authentication has been widely adopted by the mobile market industry with over 80% of the mobile devices coming with fingerprint sensors or other biometric authentication mechanisms in 2020 [82]. Although biometric authentication is evolving to be the main form of authentication in mobile devices, Android enforces an additional option for authentication for enabling features of the OS that could otherwise compromise sensitive data. The second form of authentication can be either PIN, pattern or password.

## 4.1 A Social Engineering Attack

The problem with user-input based authentications is that they can easily be attacked using phishing techniques [15]. Phishing is a form of social engineering. Users trying to access content requiring input for authentication may be presented with a user interface pretending to be a legitimate-looking input prompt belonging to part of the system that the user is trying to access.

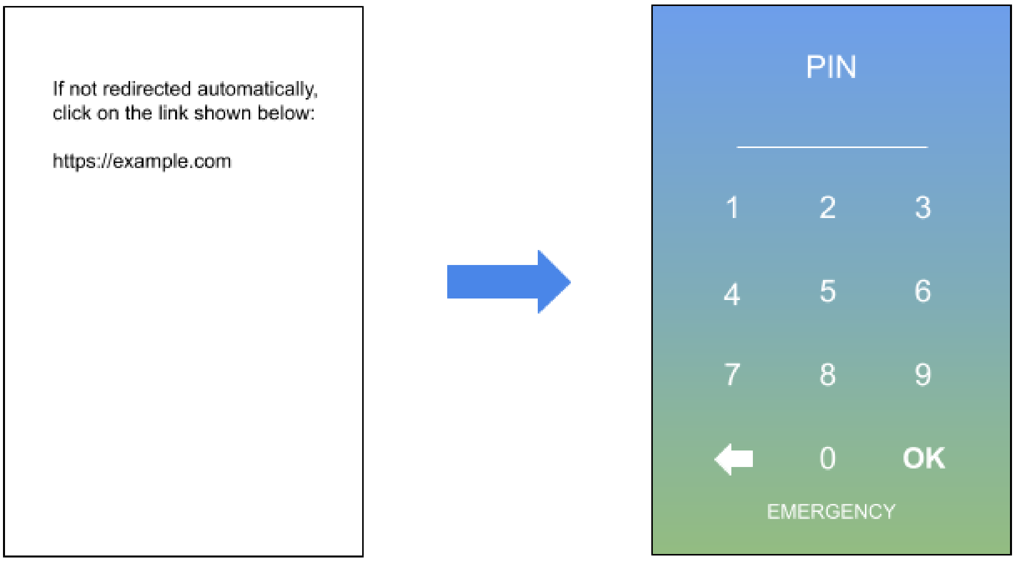
This technique has been used for stealing user passwords over the years, mostly from social web platforms such as Facebook or Twitter. Recently the same technique has been seen being applied for stealing lockscreen PINs [14] and we saw a chance to fully automate this technique and make it accessible for educational purposes.

### 4.1.1 Methodology

The way this technique works is by creating two webpage interfaces, one which depicts a redirection page to an legitimate website and one which depicts a legitimate looking lockscreen of an Android phone. A link that directs to the first webpage is then sent to a target user e.g. through a convincing phishing email. That page shows a message that tries to persuade the user clicking on an element on the screen e.g. a link. That webpage could present a message similar to the one shown below.

> If not redirected automatically, click on the link shown below:
>
> https://example.com

Then when the user tries to click on that link, instead of being redirected to the webpage mentioned on the shown message, the webpage activates fullscreen mode on the mobile browser. While on full screen mode the second webpage interface is being displayed. At this stage the webpage simulates a locked mobile phone screen asking for a PIN, password or pattern to be unlocked. If the user proceeds to enter any input to unlock his presumably locked phone, the browser exits full screen mode and the entered input is sent to the attacker, accompanied with information about the IP address and model of the mobile device the user used to visit the webpage. The user is then redirected to the previously shown link. Figure 4.1 shows the two stages of this technique.



If not redirected automatically,
click on the link shown below:

https://example.com

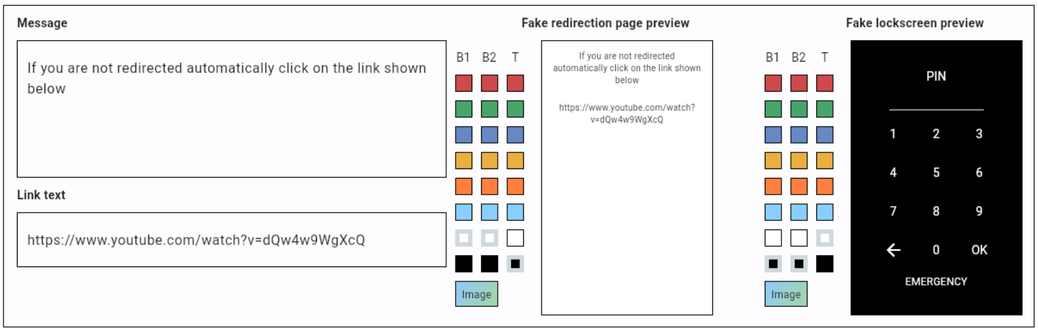
Figure 4.1: Example screens depicting the two stages of the lockscreen authentication acquiring technique

### 4.1.2 Implementation & Usage

This technique is implemented in as two components. The first one is the view and controller component as shown in Figure 3.4 and it handles user interactions with our tool.

As shown in Figure 4.2 the view panel of this component provides two text field inputs, one for the text message displayed during the first stage of the attack and one for the depicted link. Alongside those two text fields a user can see previews of the screen shown in both stages of the attack. Any change on the text fields is automatically depicted in the preview of the first stage.

Users are also provided with the option of changing the style of the webpages. Next to each preview, we can see 3 columns of hardcoded colors which users can pick from. The first two columns are responsible for the background colors of the pages using linear gradient coloring and the third column gives us the option to change the text color.



Message: If you are not redirected automatically click on the link shown below

Link text: https://www.youtube.com/watch?v=dQw4w9WgXcQ

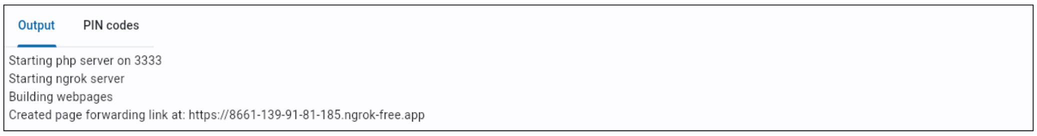
Message
Link text
Fake redirection page preview
Fake lockscreen preview
B1 B2 T
Image
If you are not redirected automatically click on the link shown below
https://www.youtube.com/watch?v=dQw4w9WgXcQ
B1 B2 T
Image
PIN
1 2 3
4 5 6
7 8 9
← 0 OK
EMERGENCY

Figure 4.2: Redirection and lockscreen page creation section

The default colors for the first stage are white for the background and black for the text. For the second stage the default colors are black for the background and white for the text. Users are also given the option to use backgrounds of theirs choosing, by inserting a custom image. That way one can more accurately simulate a real mobile device lockscreen. A collection of default background images shipped with popular Android devices is provided inside a folder together with our tool, when the user chooses to use a custom image for the a background he is sent to that directory first for picking an image file.

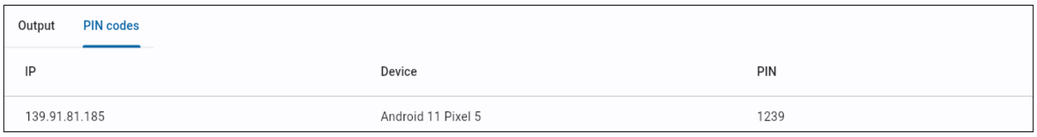
Under the page creation section users can see a tabbed view which consists of an `output` tab and a `PIN codes` tab. The `output` tab will display the messages coming from the second component as shown in Figure 4.3. The `PIN codes` tab will display any data acquired from a target mobile device e.g. PIN code, IP and device model as shown in Figure 4.4.

The second component of the implementation of this technique consists of the mechanisms which generate and forward the webpages based on the user inputs. This component contrary to the rest of the tool is not implemented in Python but



```
Starting php server on 3333
Starting ngrok server
Building webpages
Created page forwarding link at: https://8661-139-91-81-185.ngrok-free.app
```

Figure 4.3: Output tab



| IP | Device | PIN |
|---|---|---|
| 139.91.81.185 | Android 11 Pixel 5 | 1239 |

Figure 4.4: PIN codes tab

is instead implemented using a combination of PHP, HTML/CSS/Javascript and shell commands, which communicate with the rest of the tool through a Python script that is responsible for the interactions between the user interface and the shell script. A shell script generates the webpages for Android, iOS and PC Windows and starts the required PHP and Ngrok local servers. PHP is used for running a minimal backend server which will handle requests from the webpages. Ngrok is then used to forward the localhost created webpage and an ngrok domain is created which points to our frontend[1].

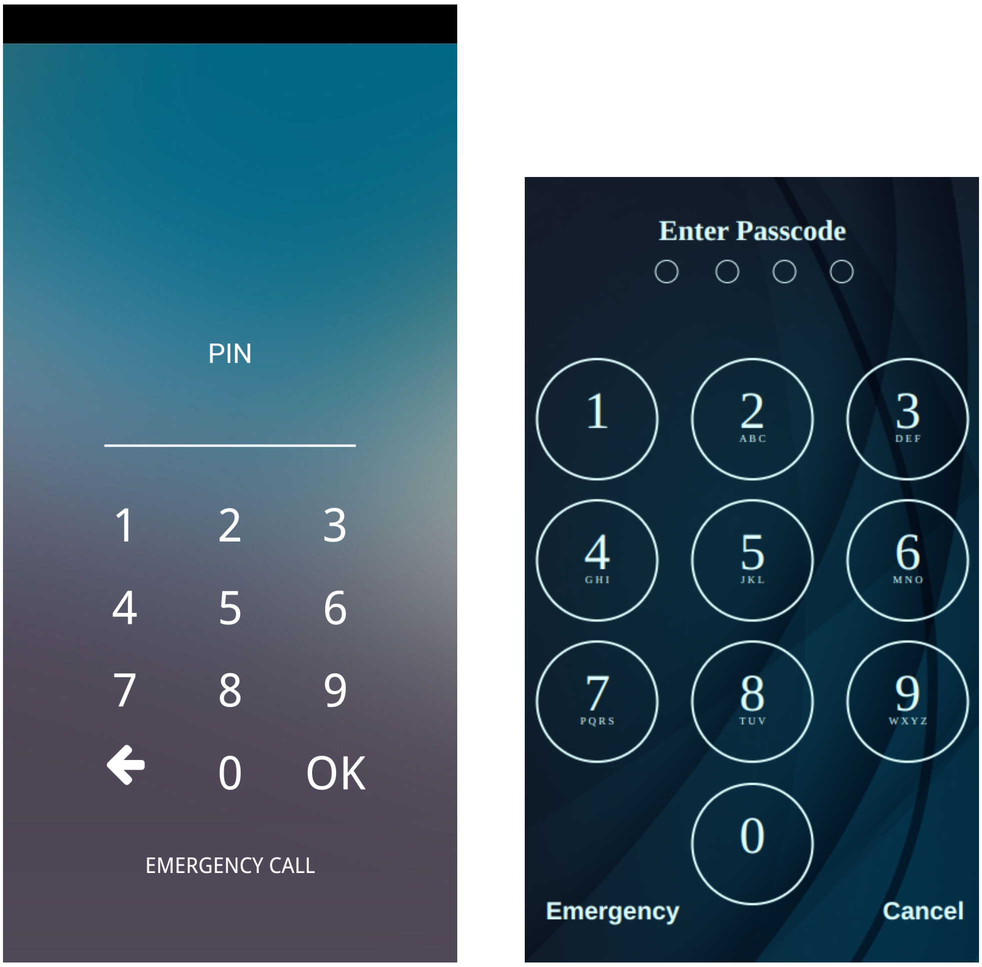
The created link is displayed in the view panel of the user interface alongside a copy icon so it can be easily transferred on the clipboard of the operating system. All the stages of the initialization of the link used in this technique are displayed on the output tab. Users can shutdown the link creation process at any point of the initialization.

Although this technique is targeted at Android mobile devices lockscreen, the same technique applies to interactions with the browser on other devices and operating systems. For completeness, we have included the features for the creation of legitimate looking lockscreens for PC Windows and iOS devices, although they are not configurable using the user interface of our tool and the acquired data from PC Windows and iOS devices will be visible in the `output` tab but not in the `PIN codes` tab of our tool.

### 4.1.3 Evaluation

We evaluated our technique by accessing the created links from a Xiaomi A2 Lite, a simulated view of a Pixel 3 device, a simulated view of an iPhone 12 Pro device and a device using the Windows operating system. The simulated view for the Pixel 3 and the iPhone 12 Pro devices was achieved through the Developer Tools of the Chrome browser [37]. Figure 4.5 depicts the lockscreen as shown on a Xiaomi

[1] An ngrok executable is included in our tool and is ready to be used, although the user has to specify an authentication token through the tool settings, to use the ngrok service.

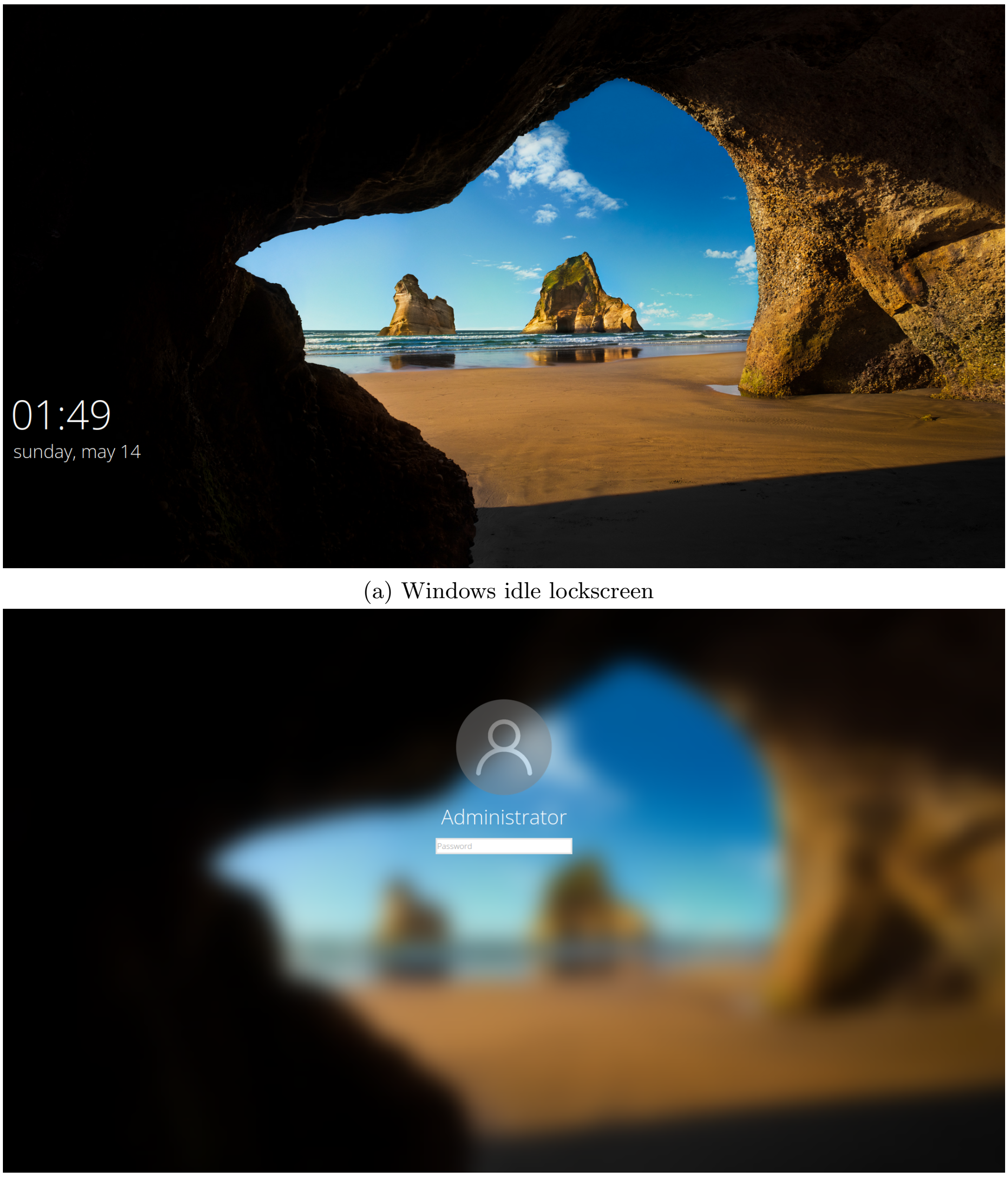


(a) Phishing lockscreen on a Xiaomi A2 Lite

(b) Phishing lockscreen on an emulated iPhone 12 Pro device

Figure 4.5: Screenshots of the lockscreen on Android and iOS

A2 Lite device, on the second stage of this technique. Figure 4.5 depicts the lockscreen as shown on the emulated iPhone 12 Pro device, on the second stage of this technique. Figure 4.6 depicts the lockscreen on the second stage of this technique, as shown on a device using a Windows operating system.



(a) Windows idle lockscreen

(b) Windows lockscreen waiting for user inputs

Figure 4.6: Screenshots of the phishing lockscreen on a Windows device

## 4.2 PIN / Password Brute-forcing

A password brute-force attack consists of an attacker submitting many passwords with the hope of eventually guessing correctly. Such an attack is usually used when there are no other viable options to bypassing the encryption mechanism of a system, e.g. by utilizing weaknesses in the encryption system used. This method is very fast when used to decrypt artifacts encrypted with short passwords. For longer passwords other methods are preferably used because a brute-force attack takes too long. Longer passwords have more possible values, making them exponentially more difficult to crack than shorter ones [42]. A variation of this technique is a dictionary attack [7], where instead of trying every possible password, we use a collection of the more probable passwords.

### 4.2.1 Methodology

In our implementation we constantly try passwords or PIN codes on a mobile device lockscreen. One way we could enable this approach is by making use of an OTG cable [71] and a USB Rubber Ducky. The combination of those two pieces of hardware will allow us automatic password input submission on a mobile device. The OTG cable allows us to connect any supported peripheral piece of equipment to the target device, such as a keyboard. Then we would use the OTG cable to connect a USB Rubber Ducky as shown in Figure 4.7, which has the ability to emulate a keyboard and inject keystrokes on any connected device in an automated fashion. By inserting an appropriate payload containing inputs from a desired wordlist in a USB Rubber Ducky and then connecting the aforementioned USB in an Android device through the OTG cable we can orchestrate a dictionary attack on the mobile device.

As soon as the USB Rubber Ducky connects to a target device it will start following the instructions contained in the payload. Those instructions could be typing inputs from the provided wordlist and waiting for a designated period of time, before moving to the next instruction. That way we could address possible timeouts by the Android operating system after a certain amount of failed password or PIN input attempts.

One disadvantage of this brute-forcing technique is that we do not get any programmatically attainable feedback in case of a successful attempt. This obstacle can be overcome in various ways. For example, by placing a recording camera on top of the mobile device, facing the screen of the device, we would have a record of the brute-forcing session and witness at which point the brute-forcing attack was successful and which was the correct password or PIN code.

### 4.2.2 Implementation & Usage

The way we integrated this technique in our tool, is by implementing a way in which users could create payloads for a USB Rubber Ducky. This can later be used to

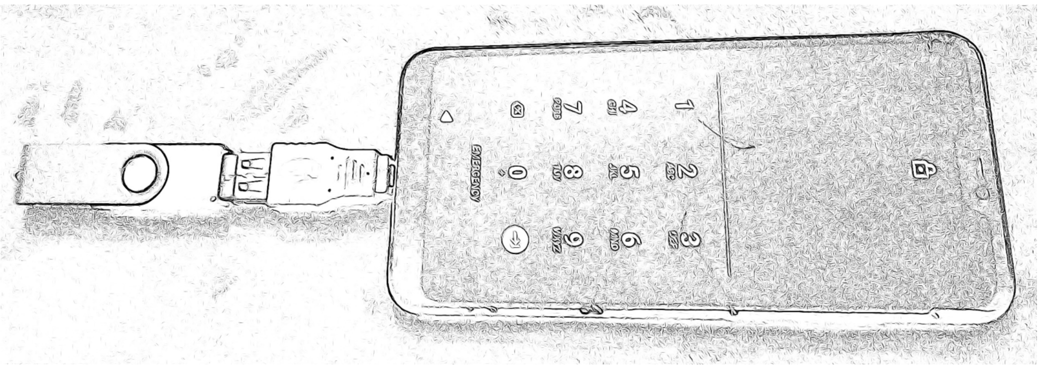
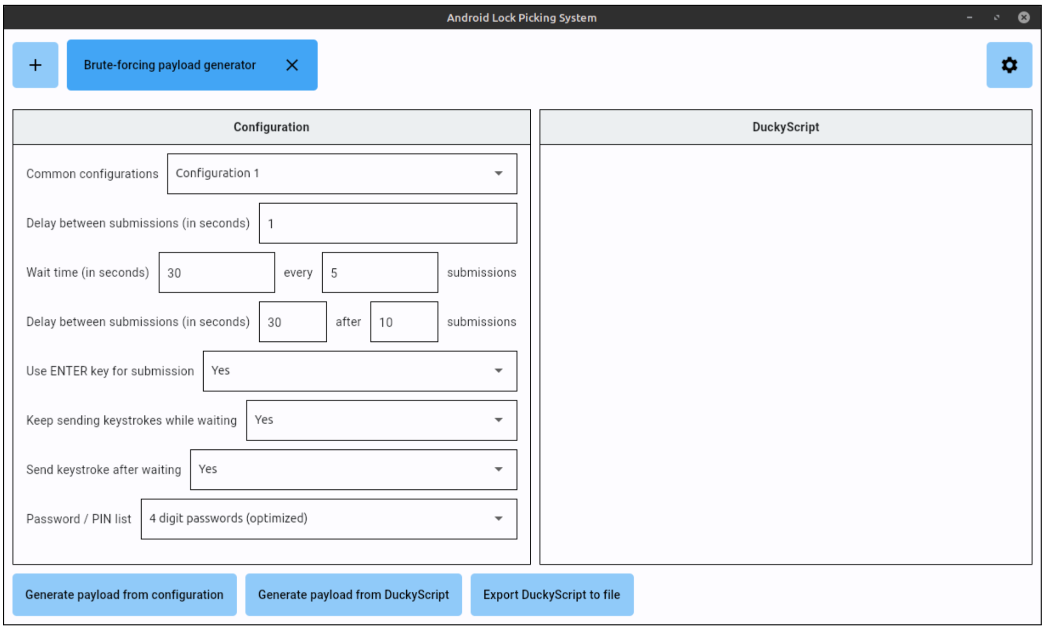


Figure 4.7: A USB Rubber Ducky connected on mobile device using OTG adapter

brute-force the password or PIN code of a mobile device. While implementing this feature of our tool we worked on two separate components which come together to enable the payload creation feature.


Android Lock Picking System
+
Brute-forcing payload generator
Configuration
DuckyScript
Generate payload from configuration
Generate payload from DuckyScript
Export DuckyScript to file

Common configurations: Configuration 1

Delay between submissions (in seconds): 1

Wait time (in seconds): 30 every 5 submissions

Delay between submissions (in seconds): 30 after 10 submissions

Use ENTER key for submission: Yes

Keep sending keystrokes while waiting: Yes

Send keystroke after waiting: Yes

Password / PIN list: 4 digit passwords (optimized)

Figure 4.8: Payload generation panel

The first component is the payload generation panel which enables our users to interact with the payload generator. It consists of two sections, one configuration section and one DuckyScript [40] editing section. The configuration section enables us to edit various aspects of the brute-forcing process. This feature is needed since Android allows up to a specific number of failed authentication attempts before setting a timeout for the next attempt. Below we list the configuration options that are available:

- Common configurations: Allows us to select between 3 hardcoded useful configurations (seen in Table 4.1).
- Delay between submissions (in seconds): Sets the delay time between each password / PIN code submission.
- Wait time (in seconds) every N submissions: Sets the time the USB Rubber Ducky should wait after trying N password / PIN code submissions. The value N is also configurable.
- Delay between submissions (in seconds) after N submissions: Sets what the new delay should be after N password / PIN code submissions. The value N is also configurable.
- Use ENTER key for submissions: Sets if an ENTER keystroke is needed after each password / PIN code submission.
- Keep sending keystrokes while waiting: Sets if keystrokes should be submitted at a constant rate, while waiting for timeouts or password and PIN submissions. In most cases it is used to keep the screen active during the brute-forcing processes. The TAB key is used for this action.
- Send keystroke after waiting: Sets if an extra keystroke should be send after each timeout or other delay. In most cases it is used to prepare the target input for the next submission after a timeout. The TAB key is used for this action.
- Password / PIN list: Allows the user to choose between a selected collection of wordlist for usage in the brute-force attack. Through this option users can also add a custom wordlist, which will then be visible in the available options. A user added wordlist is not preserved between sessions.

After selecting the preferable options for their use case, users can generate a payload, by clicking the `Generate payload from configuration` button. When generating a payload, the DuckyScript representation of that payload is also created and displayed on the DuckyScript editing section. Users are also given the option to make further changes to the produced payload using the DuckyScript section and generate a new payload using the button `Generate payload from DuckyScript` or save the results in a DuckyScript file using the button `Export DuckyScript to file`.

The second component is the USB Rubber Ducky encoder which given the options selected through the user interface, has the responsibility of creating the DuckyScript file and the payload. These, depending on the action performed by the user are saved in the files duckyscript.txt and inject.bin respectively.

After the payload generation process is completed, users can find the inject.bin file in the output folder of the tool. The inject.bin file can then be transferred to the microSD card of the USB Rubber Ducky for usage on a target device.

| Common configurations | | | |
|---|---|---|---|
| | Configuration 1 | Configuration 2 | Configuration 3 |
| Delay between submissions | 1 | 1 | 1 |
| Wait time every N submissions (in seconds) | 30 every 5 submissions | 30 every 1 submissions | 30 every 1 submissions |
| Delay between submissions (in seconds) after N submissions | 30 after 10 submissions | 60 after 40 submissions | 0 after 0 submissions |
| Use ENTER key for submissions | Yes | Yes | Yes |
| Keep sending keystrokes while waiting | Yes | Yes | No |
| Send keystroke after waiting | Yes | Yes | No |
| Password / PIN list | 4 digit passwords (optimized) | 4 digit passwords (optimized) | 4 digit passwords (optimized) |

Table 4.1: Common configurations

| Xiaomi A2 Lite brute-forcing configuration | |
|---|---|
| Delay between submissions | 3 |
| Wait time every N submissions (in seconds) | 30 every 5 submissions |
| Delay between submissions (in seconds) after N submissions | 30 after 10 submissions |
| Use ENTER key for submissions | Yes |
| Keep sending keystrokes while waiting | Yes |
| Send keystroke after waiting | Yes |
| Password / PIN list | 4 digit passwords (optimized) |

Table 4.2: Xiaomi A2 Lite brute-forcing configuration

### 4.2.3 Evaluation

The way we evaluated our technique is by attempting brute-forcing attacks using the USB Rubber Ducky on 2 devices, a Xiaomi A2 Lite running Android 10 and a Genymotion [32] emulated Samsung Galaxy S6 running Android 6. For these tests we used a simple payload which was programmed to try all possible 4 digit PIN codes from 0000 up to 9999. The actual PIN code that was set on both devices was 1996.

We were not able to conclude our evaluation process on the Xiaomi A2 Lite device since the brute-force throttling mechanism of the Gatekeeper [6] service that has been introduced in Android 7, starts adding timeouts exponentially after 29 failed attempts. The timeouts sequence after every attempt for N number of failed attempts is shown below.

- 0 to 4 failed attempts: 0 seconds
- 5th failed attempt: 30 seconds
- 6 to 10 failed attempts: 0 seconds
- 11 to 29 failed attempts: 30 seconds
- 30 to 139 failed attempts: $30 \cdot 2^{\lfloor \frac{x-30}{10} \rfloor}$
- 140 or more failed attempts: 1 day

The exact configuration options that were used during the payload generation process for the evaluation of the brute-forcing technique on the Xiaomi A2 Lite device are seen in Table 4.2.

| Emulated Samsung Galaxy S6 brute-forcing configuration | |
|---|---|
| Delay between submissions | 3 |
| Wait time every N submissions (in seconds) | 30 every 5 submissions |
| Delay between submissions (in seconds) after N submissions | 30 after 10 submissions |
| Use ENTER key for submissions | Yes |
| Keep sending keystrokes while waiting | No |
| Send keystroke after waiting | Yes |
| Password / PIN list | 4 digit passwords (optimized) |

Table 4.3: Emulated Samsung Galaxy S6 brute-forcing configuration

The evaluation process concluded successfully on the emulated Samsung Galaxy S6 device, where the PIN was brute-forced in approximately 3 hours, as expected after 361 attempts. The exact configuration options that were used during the payload generation process for the evaluation of the brute-forcing technique on the Android 7 device are seen in in Table 4.3.

# Chapter 5

# Extracting Encryption Keys and Decrypting Application Artifacts

Mobile devices, especially smartphones, have become a central part of our daily lives. They contain a wealth of information about us, our activities, and our connections. This information is not only valuable to the users themselves but also to law enforcement agencies and forensic investigators. However, accessing this information can be a challenging task, especially when the device and the artifacts stored within it are encrypted or protected by password.

To address this challenge, a component of our tool targets encrypted artifacts stored in Android devices. This component of the tool is similar to Autopsy [8], the well-known open-source digital forensics platform and its Android targeted module called ALEAPP [10], but is designed specifically to handle encrypted artifacts and the unique characteristics of mobile applications.

With this function of our tool, forensic investigators can, analyze, and report on a wide range of data types from Android devices, including encrypted passwords, encrypted databases and encrypted messages or media. The tool performs on a logical extraction of an Android device which has to be acquired beforehand with a different tool. Most of the time, full access to the device is required to perform a logical extraction, which on the surface makes the usage of our tool's function sound of no need. However the value of this function comes from the fact that it provides the possibility to decrypt artifacts and extract encryption keys which even if we had full access to the device, we would not be able to access.

One of the strengths of this tool's function is its ease of use and configurability. Its intuitive user interface makes it accessible to both novice and experienced users, allowing them to quickly and efficiently extract the information they need such as paths where the artifacts were found, encryption keys and encryption schemes used.

In this chapter we discuss ways to decrypt artifacts from 19 Android applications and one Android system component and we show how we implemented those decryptions techniques in our tool. The Android applications and system component analyzed are listed below.

- `Android Lockscreen Pattern`
- `WeChat`
- `Wickr`
- `Signal`
- `Snapchat`
- `Viber`
- `Chromium Based Web Browsers`
  - `Chrome`
  - `Edge`
  - `Brave`
- `Mega`
- `AVG`
- `AppLock`
- `WeVault`
- `Sgallery and similar 'Calculator' apps`
  - `Sgallery - hide photos & video`
  - `Calculator - hide photos`
  - `Calculator - photo vault`
  - `LOCKED Secret Photo Vault`
- `Apps Lock & File Encryption`
- `Secret Calculator Lock Vault`
- `Vaulty :  Hide Pictures Vid`

## 5.1 Methodology

The way artifacts are analyzed on our tool is through separate modules which handle those artifacts as part of the the application or Android system component being analyzed.

### 5.1.1 Android Lockscreen Pattern

On Android versions 5.1 and lower the pattern used for locking the screen is stored under `/data/system/gesture.key`. The pattern is stored as an unsalted SHA-1 [84] value and the SHA-1 value is calculated using the method shown in the code below, where each point in the pattern is represented by a number between 0 and 8 as shown in Figure 5.1.

```
private static byte[] patternToHash(List pattern) {
    if (pattern == null) {
        return null;
    }

    final int patternSize = pattern.size();
    byte[] res = new byte[patternSize];
    for (int i = 0; i &lt; patternSize; i++) {
        LockPatternView.Cell cell = pattern.get(i);
        res[i] = (byte) (cell.getRow() * 3 + cell.getColumn());
    }
    try {
        MessageDigest md = MessageDigest.getInstance("SHA-1");
        byte[] hash = md.digest(res);
        return hash;
    } catch (NoSuchAlgorithmException nsa) {
        return res;
    }
}
```

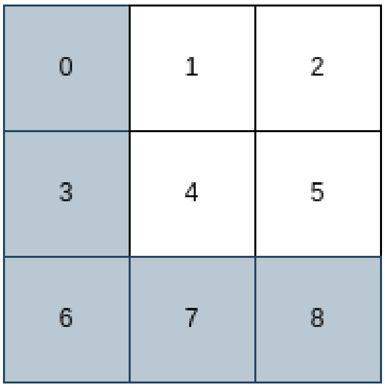


Figure 5.1: Interpretation of an L shaped pattern on the Android pattern lock

The amount of possible patterns that can be created on the Android lockscreen is 389,112. Since the amount is a finite and relatively small, someone could brute-force the pattern almost instantly with the use of a rainbow table [62] containing the precomputed hashes of all the possible patterns.

### 5.1.2 WeChat

`WeChat` [54] is an instant messaging, social media, and mobile payment app available for a multitude of platforms including Android and iOS. On Android, WeChat stores all the messages, contacts, and other data exchanged through the application on a database called `EnMicroMsg.db`. That database can be found under `/data/data/com.tencent.mm/MicroMsg/{a-random-hexadecimal-string}/EnMicroMsg.db` and is encrypted using SQLCipher 3 [86] and the PRAGMA value options shown below.

- `PRAGMA cipher_use_hmac = OFF`
- `PRAGMA cipher_page_size = 1024`
- `PRAGMA kdf_iter = '4000'`

The encryption key of the database is created using the IMEI number of the phone and the UIN of WeChat. On newer versions of Android such as 10 or higher a fixed hexadecimal string is used for the encryption key creation process, specifically `1234567890ABCDEF`. The IMEI is the 15 digits unique number that you can usually get at the back of the mobile phone, or by entering *#06# on the phone. The UIN is the unique identification number tied to the user's WeChat account. Inside the Android filesystem we can possibly find the IMEI number under `/data/data/com.tencent.mm/MicroMsg/CompatibleInfo.cfg`. Similarly the UIN can be found under the following paths.

- `/data/data/com.tencent.mm/shared_prefs/system_config_prefs.xml`
- `/data/data/com.tencent.mm/shared_prefs/com.tencent.mm_preferences.xml`
- `/data/data/com.tencent.mm/shared_prefs/auth_info_key_prefs.xml`
- `/data/data/com.tencent.mm/MicroMsg/systemInfo.cfg`

The way the encryption key is created is by concatenating the IMEI and UIN strings, applying the MD5 hash function on their concatenation and then extracting the seven first characters of the hash. That substring of the resulting hash constitutes the key used for the encryption of `EnMicroMsg.db`. That way if we are able to extract the IMEI(or use the fixed hexadecimal string instead) and UIN values from the Android filesystem, then we should be able to decrypt the database using the aforementioned PRAGMA values. The key creation procedure is shown in Figure 5.2.

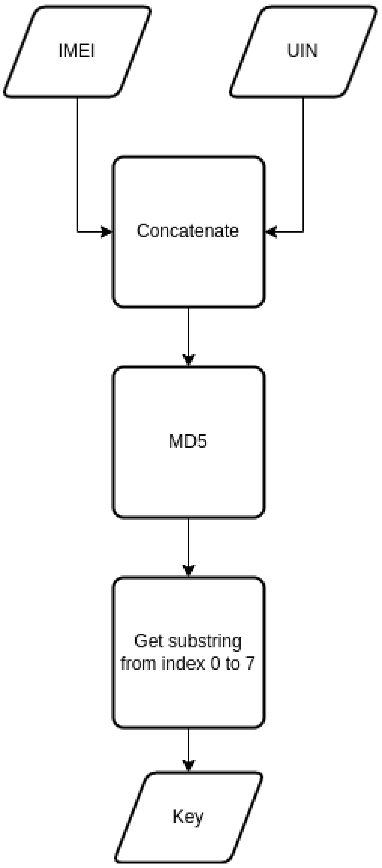


Figure 5.2: WeChat's EnMicroMsg.db encryption key creation process

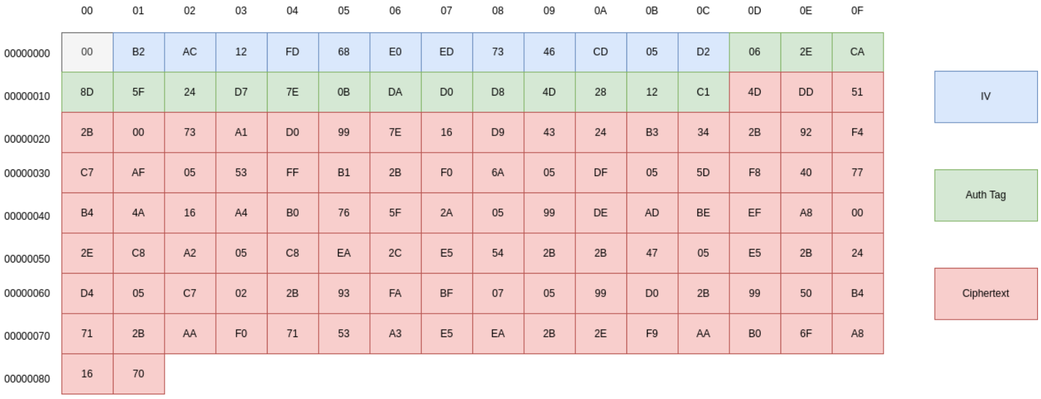
### 5.1.3 Wickr

`Wickr Me - Private Messenger` [45] is an instant messenger application, specifically designed for encrypted communications. It is available for Windows, macOS, Ubuntu, iOS and Android. On Android, messages that are stored in the device, are saved in a database called `wickr_db`. That database can be found under `/data/data/com.mywickr.wickr2/databases/wickr_db` and is encrypted using SQLCipher 4 [87] and the default PRAGMA value options.

The encryption key of the database can be created using two different ways, depending on wether the user enabled password login on the application. If password login is enabled, the encryption key is created using the provided password. If automatic login is enabled, the encryption key can be created using a file stored in the filesystem. The encryption key can always be derived using the second method, since both methods create the same encryption key and automatic login is enabled by default when installing the application. Furthermore, if the user changes the method of logging in, the files created for the automatic login process, are not deleted [47]. The files used for the encryption key derivation, based on the automatic login process are shown below.

- `/data/system/users/0/settings_ssaid.xml`
- `/data/data/com.mywickr.wickr2/files/kcd.wic`
- `/data/data/com.mywickr.wickr2/files/kck.wic`

The database encryption key is stored using double encryption. Firstly the id value of Wickr from `settings_ssaid.xml` is used to decrypt the data in kcd.wic. The id value is passed through the MD5 hash function, then from the generated hash, offset 14 is changed to 3 and offset 19 is changed to 8, 9, a or b according to the remainder of the division of the value by 4. By hashing the generated value using the SHA-256 hash function, we can get the key to decrypt the data inside kcd.wic using AES-256-GCM decryption. The kcd.wic file includes the ciphertext, authentication tag and IV. The structure of the file is shown in Figure 5.3. Then using the value from kck.wic we can further decrypt the value we got from the decryption of the data inside kcd.wic, using AES-256-GCM. The structure of kck.wic is similar to that of kcd.wic. Finally the database key will be contained between the offsets 4 and 25 of the resulting value after the second decryption. The key derivation process is shown in Figure 5.4.



| | 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 0A | 0B | 0C | 0D | 0E | 0F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 00000000 | 00 | B2 | AC | 12 | FD | 68 | E0 | ED | 73 | 46 | CD | 05 | D2 | 06 | 2E | CA |
| 00000010 | 8D | 5F | 24 | D7 | 7E | 0B | DA | D0 | D8 | 4D | 28 | 12 | C1 | 4D | DD | 51 |
| 00000020 | 2B | 00 | 73 | A1 | D0 | 99 | 7E | 16 | D9 | 43 | 24 | B3 | 34 | 2B | 92 | F4 |
| 00000030 | C7 | AF | 05 | 53 | FF | B1 | 2B | F0 | 6A | 05 | DF | 05 | 5D | F8 | 40 | 77 |
| 00000040 | B4 | 4A | 16 | A4 | B0 | 76 | 5F | 2A | 05 | 99 | DE | AD | BE | EF | A8 | 00 |
| 00000050 | 2E | C8 | A2 | 05 | C8 | EA | 2C | E5 | 54 | 2B | 2B | 47 | 05 | E5 | 2B | 24 |
| 00000060 | D4 | 05 | C7 | 02 | 2B | 93 | FA | BF | 07 | 05 | 99 | D0 | 2B | 99 | 50 | B4 |
| 00000070 | 71 | 2B | AA | F0 | 71 | 53 | A3 | E5 | EA | 2B | 2E | F9 | AA | B0 | 6F | A8 |
| 00000080 | 16 | 70 | | | | | | | | | | | | | | |

Figure 5.3: The structure of kcd.wic and kck.wic

### 5.1.4 Signal

`Signal Private Messenger` [29] is an encrypted messaging service for instant messaging, voice, and video calls. It is available for Android, iOS and Desktop. Signal uses an encrypted database for storage of messages and data used throughout the application. On Android, the database can be found under `/data/data/org.thoughtcrime.securesms/databases/signal.db` and is encrypted using SQLCipher 4 and the PRAGMA value options shown below.

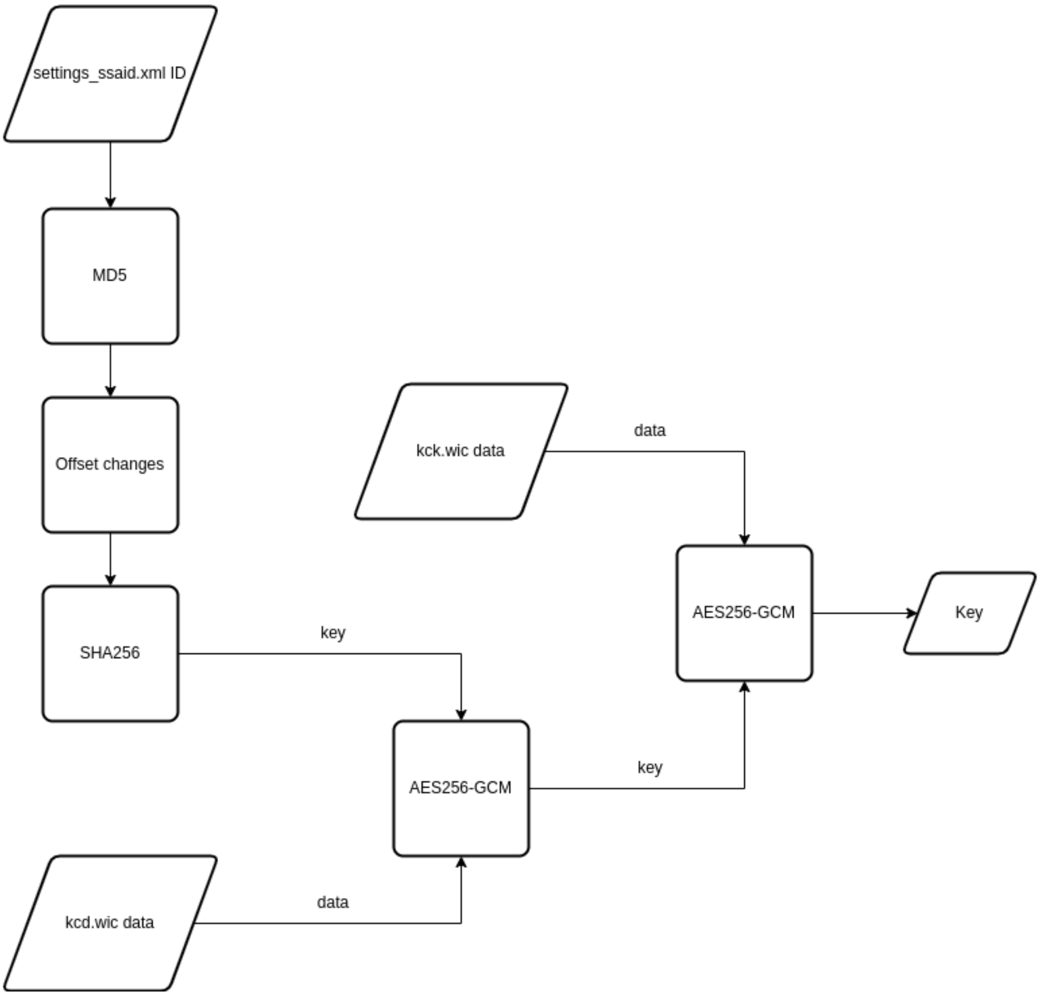


Figure 5.4: Wickr's wickr_db encryption key derivation process

- `PRAGMA cipher_default_page_size = 4096`
- `PRAGMA cipher_default_kdf_iter = 1`
- `PRAGMA cipher_hmac_algorithm = HMAC_SHA1`
- `PRAGMA cipher_kdf_algorithm = PBKDF2_HMAC_SHA1`

The encryption key of the database is stored in an encrypted format inside an XML file which can be found under `/data/data/org.thoughtcrime.securesms/shared_prefs/org.thoughtcrime.securesms_preferences.xml`. The key that decrypts the encrypted database key is stored inside the Android keystore. Since the Android keystore implements extraction prevention mechanisms, we can't extract the key directly from it [35]. We can though access Signal's keystore through a secondary application pretending to be Signal [47] [33]. The only downside to this solution is that the secondary application has to be installed on the same Android device where the Signal application is installed and root access should be available on that device.

After the secondary application gets installed on the Android device containing the Signal application whose database we want to decrypt, it will create its own keystore. Then after its keystore is replaced with Signal's keystore, the application will be able to access Signal's keystore and decrypt the database decryption key using the values `data` and `iv` from `org.thoughtcrime.securesms_preferences.xml` with AES-256-GCM, the same way the actual Signal application is doing it to decrypt the `data` value and get the database decryption key. The key extraction process is shown in Figure 5.5.

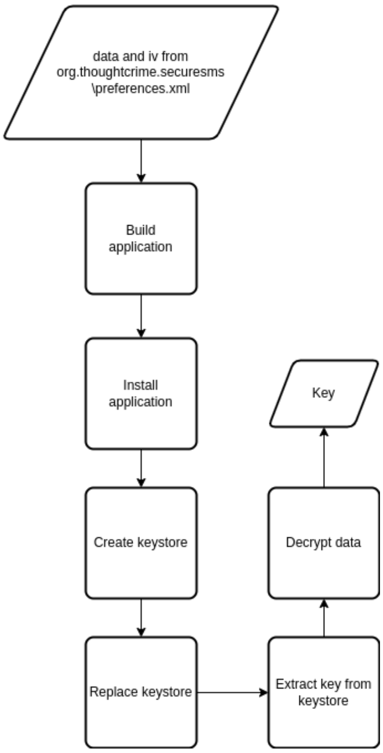


Figure 5.5: Signal's signal.db encryption key extraction process

### 5.1.5 Snapchat

`Snapchat` [44] is a multimedia instant messaging application, available for Android and iOS. The main feature of Snapchat is that pictures and messages are typically accessible for a brief duration before they become inaccessible to the recipients. Over time, the application has progressed from its initial emphasis on photo sharing to its current inclusion of users' `Stories`, which consist of 24 hours of chronological content. It also provides a feature called `My Eyes Only`. This feature that enables users to save photos and `Stories` protected by a secret PIN. Someone can only access them by entering the users' PIN. To view `My Eyes only`, users can swipe up from the camera screen to open Memories, then swipe left to the `My Eyes Only` tab and enter their PIN.

That PIN is a 4 digit number saved in a database under `/data/data/com.snapchat.android/databases/memories.db`. Inside the database on the table `memories_meo_confidential` at entry `hashed_passcode` we can find the hashed PIN which was created using bcrypt [55]. Since the PIN is only 4 digits long we can brute-force it within less than 17 seconds on most of the commercial processors currently available, even in a worst case scenario [48].

### 5.1.6 Viber

`Viber - Safe Chats And Calls` [70] is a cross-platform voice over IP (VoIP) and instant messaging (IM) software application available for Android, iOS, Windows, macOS and Linux platforms. Viber provides a feature allowing users to hide specific chats, protected by a secret PIN. On Android, to view the hidden chats, users can open the application on their device and press the search bar available at the top of the screen. Then after entering their `hidden chats` PIN in the search bar, they will be able to view all the hidden chats.

The PIN is a 4 digit number saved in a database under `/data/data/com.viber.voip/databases/viber_prefs`. Inside the database in the table `kvdata` at the entry where `key='key_hidden_chats_pin'`, we can find the encoded PIN by reading the corresponding `value` attribute. The way the PIN is stored in the database is by concatenating it with the string value `Shawl9_Valid_Yeastv` and hashing the result using SHA-256 as shown in Figure 5.6, the resulting hash then is stored in the database. Similarly to Snapchat, since the PIN is only 4 digits long we can easily brute-force it in less than a second, even in a worst case scenario.

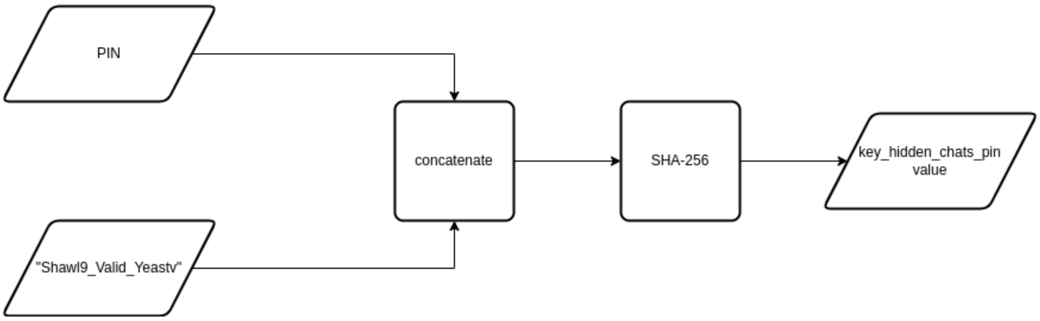


Figure 5.6: Viber's PIN hashing process

### 5.1.7 Chromium Based Web Browsers: Chrome, Edge & Brave

`Google Chrome: Fast & Secure` [51], `Microsoft Edge: Web Browser` [16] and `Brave Private Web Browser` [77] are Chromium [39] based web browsers which all use the same way of storing the saved logins data on their local databases. The saved logins data is stored in a database called `Login Data`, which can be found under `/app_chrome/Default/LoginData` relatively to the package path of the app as shown below.

- `Chrome: /data/data/com.android.chrome/app_chrome/Default/LoginData`
- `Edge: /data/data/com.microsoft.emmx/app_chrome/Default/LoginData`
- `Brave: /data/data/com.brave.browser/app_chrome/Default/LoginData`

Each saved username and its corresponding encrypted password are saved under a `logins` table inside the `Login Data` database. The passwords are encrypted using AES-128 in CBC mode with PKCS#7 padding. The key used in the encryption is created using PBKDF2 [46] as shown in Figure 5.7, using the values shown below.

- password = peanuts
- salt = saltysalt
- dkLen (size of the derived key in bytes) = 16
- count (number of iterations) = 1

A 16 character empty string is used as the IV in the encryption. After encryption the encrypted password is sometimes prepended with the strings `v10` or `v11`. During decryption, the prepended strings should be removed before the decryption process.

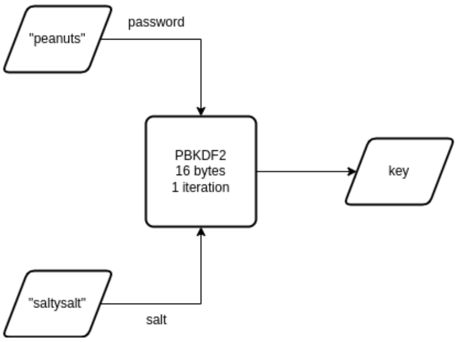


Figure 5.7: Chromium based browser's login data encryption key creation process

### 5.1.8 Mega

MEGA [53] is a cloud storage and file hosting service which provides user-controlled encrypted cloud storage that is accessed with web browsers and dedicated applications for mobile devices. On Android devices Mega stores its data needed for usage of the application in a database under data/data/mega.privacy.android.app/databases/megapreferences. Although the database itself is not encrypted, the entries of the tables inside it are. The table entries are encrypted using AES-256 in ECB mode using as key the byte representation of the string android_idfkvn8w4y*(NC$G*(G($*G with PKCS#7 padding.

One table called completedtransfers holds the info related to the files uploaded, through the service. This means that although the files are uploaded in a secure and privacy preserving way, we can still extract info related to those files like file names and file sizes, by decrypting the entries of the aforementioned table.

### 5.1.9 AVG

AVG AntiVirus & Security [57] is a line of antivirus software available for Windows, macOS, iOS and Android. AVG encompasses many standard functions found in modern antivirus and internet security programs. These features include regular scans, scanning of incoming and outgoing emails (with the option to add footers indicating this), the capacity to repair certain virus-infected files, and a quarantine area (virus vault) where infected files are securely stored.

On the Android version of the application there exists an extra functionality called AVG Vault enabling users to encrypt and hide media files protected by a PIN or a pattern provided by the user. The PIN or pattern set by the user is saved in an XML file under /data/data/com.antivirus/shared_prefs/PinSettingsImpl.xml. The PIN is a 4 digit number and it is saved under the encrypted_pin entry as a SHA1 hash. The pattern is saved under the encrypted_pattern entry as a SHA1 hash created using the same method as with the gesture.key file of Android.

Since the PIN is a 4 digit number it can easily be brute-forced in less than one second, even in a worst case scenario. The same applies to the pattern which can be brute-forced almost instantly as previously shown with the gesture.key file.

### 5.1.10 AppLock

AppLock [49] is an Android and iOS application which provides a wide array of functionalities. It can hide itself, it can lock Settings, incoming calls and any app the user chooses, protected by a user provided PIN or pattern. Moreover AppLock can encrypt and hide pictures and videos. Hidden pictures and videos are vanished from the gallery and are only visible in the pictures and video vault within the application.

After running the application for the first time the user is prompted to create a pattern which will be used for future authentications in the application. Accessing all the hidden content inside the app requires authentication to the app when lauching it. Through the user settings, it is also possible to change from a pattern to a PIN consisting of 1 to 16 digits.

The encrypted pattern or PIN in use, as well as the length of the PIN if a PIN is used, are saved in an XML file under `/data/data/com.domobile.applockwatcher/shared_prefs/com.domobile.applockwatcher\_preferences.xml`.

The encrypted PIN is saved under the `password` entry as the MD5 hash of the user provided PIN concatenated with the string `domobile`. The PIN length is saved under the `password_length` entry unencrypted. The encrypted pattern is saved under the `image_lock_pattern` entry. The pattern is hashed using a method similar to the gesture.key file of Android, then the produced hash is encoded using BASE64.

The BASE64 value is encrypted using AES-256 in CBC mode with the key and IV being the SHA-256 hash value of `domobile2011` and the hexadecimal representation of `2011071120170711` respectively. The padding used for the encryption is PKCS#7. The encryption process of the pattern is demonstrated in Figure 5.8.

The encrypted photos and videos are stored in a folder under `/data/media/0/.do0mo7bi1le1/medias`, which does not require root privileges to be accessed. Each photo or video is encrypted in a file where the end of the file contains unecrypted metadata related to the file in JSON format, with the last 11 character denoting the size of the metadata section. Both photos and videos are encrypted using AES in CBC mode with the key and IV being the value `d7d19a4b2d25ff8bd3fc7f3b97b19a5ac6def11d6377ab176eba1db365d07336` and the hexadecimal representation of `2011071120170711` respectively. The padding used for encryption is PKCS#7.

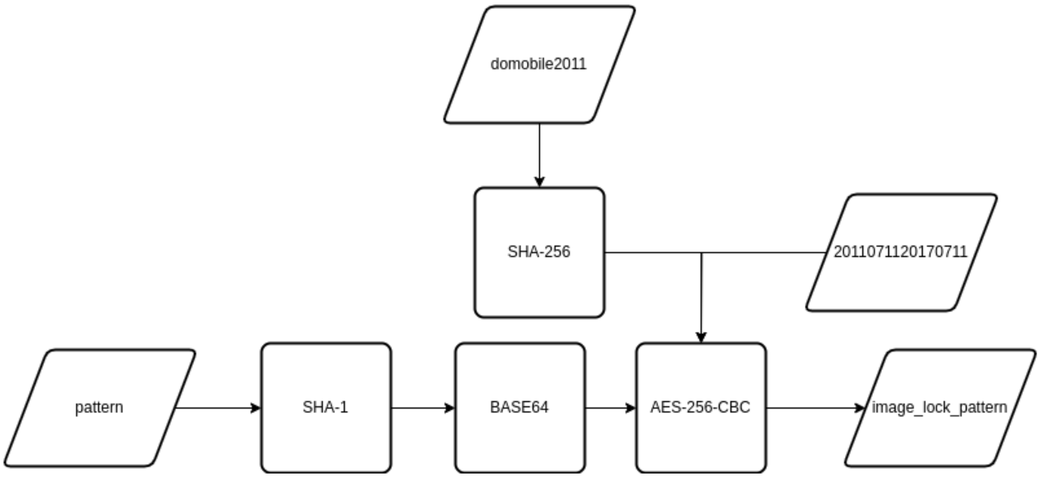


Figure 5.8: AppLock's pattern encryption process

### 5.1.11 WeVault

`WeVault - Hide Photos & Videos` [18] is an Android application providing a way to encrypt and hide your photos and videos protected by a PIN, pattern or a fingerprint lock. It also provides a private browser which can be accessed through the application.

After running the application for the first time the user is prompted to create a pattern which will be used for future authentication in the application. Accessing all the hidden content inside the application requires authentication. Through the user settings, it is also possible to change from a pattern to a PIN or a fingerprint lock.

The encrypted pattern or PIN in use, as well as the encrypted user email, are saved on an XML file under `/data/data/com.iqmor.vault/shared_prefs/com.iqmor.vault_preferences.xml`. The user email is encrypted using AES-128 in CBC mode using as key the hex representation of the string `wxg20200415iqmor` and as IV the hex representation of the string `iqmor20200415wxg`. Then it is encoded using BASE64 and it is saved under the `user_email` entry.

The PIN is encrypted using AES-256 in CBC mode. The key used for encrypting the PIN and the rest of the artifacts in WeVault, except the email, is created by interlacing the characters of the string `qmor` after each letter in the email and then hashing the created string using SHA-256 as shown in Figure 5.8. The IV used for encrypting the PIN is the same as the one used for encrypting the user email. After encryption, it is encoded using BASE64 and it is saved under the `number_lock_password` entry.

The pattern is converted to a hash using the same method as described for the gesture.key file of Android in Subsection 5.1.1, then the hash is converted to BASE64 and the encoded value is encrypted using the same method as used for the PIN. Then the encrypted value is encoded using BASE64 and it is saved under the `pattern_lock_pwd` entry.

The encrypted media files are saved in a folder under `/data/media/0/.WeVault/SFiles`, which does not require root privileges to be accessed. The files are encrypted using the same method as the one used for encrypting the PIN. Then the encrypted content of the files is prepended with some JSON formatted metadata related to the file. The email of the user is included unencrypted in the metadata section. Finally another line of text is prepended to the resulting content, containing the length of the metadata section and the encrypted content section seperated by an underscore. More metadata related to the encrypted files is also saved in encrypted files inside a folder under `/data/media/0/.WeVault/SMetas`, which again does not require root privileges to be accessed. The metadata files are encrypted using the same method as the one used for encrypting the PIN, before being encoded to BASE64.

The padding used for all encryption processes in WeVault is PKCS#7.

### 5.1.12 Sgallery and similar 'Calculator' apps

The applications listed below, provide the same functionality of encrypting and hiding media files, while also providing a camouflaging mechanism for the application itself. Specifically, the application will present itself as a functioning calculator

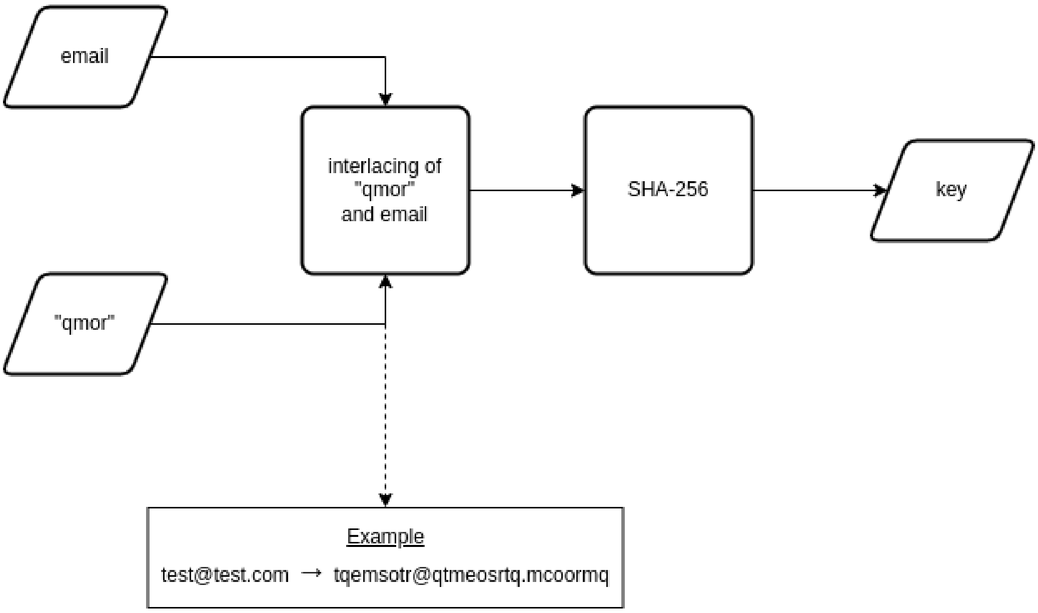

email
interlacing of "qmor" and email
SHA-256
key
"qmor"
Example
test@test.com → tqemsotr@qtmeosrtq.mcoormq

Figure 5.9: WeVault's encryption key creation process

and when a secret input will be passed to it, it will reveal the actual application and the data hidden within.

- `Sgallery - hide photos & video:` `com.hld.anzenbokusufake` [25]
- `Calculator - hide photos:` `com.hld.anzenbokusu` [23]
- `Calculator - photo vault:` `com.hld.anzenbokusucal` [24]
- `LOCKED Secret Photo Vault:` `com.lkd.calculator` [31]

The applications listed above save their encrypted artifacts in the same locations, with the only difference being the path of the package of the application as shown above.

The encrypted PIN, pattern and security question are stored in an XML file under `/data/data/{package-name}/shared_prefs/share_locked_vault.xml` The entries where they are stored are shown below.

- `PIN: 57DFEA9AEC99CD87013E3862B9DE5B7D`
- `pattern: 85B064D26810275C89F1F2CC15E20B442E98874398F16F6717BBD5D34920E3F8`
- `security question: A4CDA7B11C92D7943C99DFE00A362E1503D904E2932364D4D66F22966789A926`

The encrypted media files are stored inside a folder under `/data/media/0/.locked_vault`, which does not require root privileges to be accessed. The encrypted database is stored under `/data/data/{package-name}/db/privacy_safe.db`.

All of the artifacts except the database are encrypted using AES-128 in CBC mode using as key and IV the hex representation of the string `Rny48Ni8aPjYCnUI`, with PKCS#7 padding. The database is encrypted using SQLCipher 4 using the key `Rny48Ni8aPjYCnUI` and the default PRAGMA values options.

### 5.1.13 Apps Lock & File Encryption

`Apps Lock & File Encryption` [67] is an Android application providing a way to encrypt and hide media files stored on the phone behind a pattern lock. Similarly with other media encrypting applications, hidden media file are vanished from the gallery and are only visible in the photo and video vault within the application.

After running the application for the first time the user is prompted to create a pattern which will be used for future authentications. The pattern is saved in an SQLite database under `data/data/playground.develop.applocker/databases/applocker.db`. The pattern is stored in the table `pattern` in the `pattern_metadata` entry as a JSON formatted string. In the example shown in Figure 5.10 it is demonstrated how an L shaped pattern would be saved in the database of this application.

The encrypted media files are stored in a folder under `/data/media/0/applocker/vault`, which does not require root privileges to be accessed. Each media file is encrypted using Meta's Conceal API [56]. In the background Conceal uses AES in GCM mode. The key can be either 128 or 256 bits and the IV can be of variable length. In this case, the size of key is 256 bits and the size of the IV is 24 bits. The IV is stored in the beggining of each encrypted file, from the 3rd to the 14th byte, while the data after the 14th byte up to the last 32 bits consists the encrypted content. The last 32 bits consists the GCM Authentication Tag. The structure of an encrypted file can be seen in Figure 5.11. The key used for the encryption of every media file is the same and it can be found inside an XML file under `/data/data/playground.develop.applocker/shared\_prefs/crypto.KEY\_256.xml` in the `cipher_key` entry encoded using BASE64.

### 5.1.14 Secret Calculator Lock Vault

`Secret Calculator Lock Vault` [17] provides the ability to encrypt and hide media files, while also providing a camouflaging mechanism for the application itself, similarly to other `calculator` applications. The authentication mechanism used is a PIN, which has to be entered as an input in the calculator to open the application, the PIN can be up to 8 digits long.

The application stores the values related to its encrypted artifacts in an XML file stored under `/data/data/com.photovault.secret.calculator/shared_prefs/AppPreferences.xml`. Inside this file we can find encoded using BASE64 the encrypted encryption key for the media files under the entry `symmetric_encrypted_files_encryption_` the SHA-256 hash of that encryption key under the `hashed_files_encryption_key` entry and a PBKDF2 salt value which is used to generate the encryption key which

```
{
    "a": [
        {
            "a": 0,
            "b": 0
        },
        {
            "a": 0,
            "b": 1
        },
        {
            "a": 0,
            "b": 2
        },
        {
            "a": 1,
            "b": 2
        },
        {
            "a": 2,
            "b": 2
        }
    ]
}
```

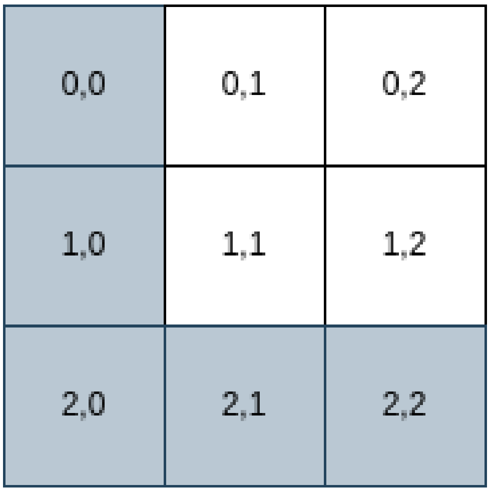
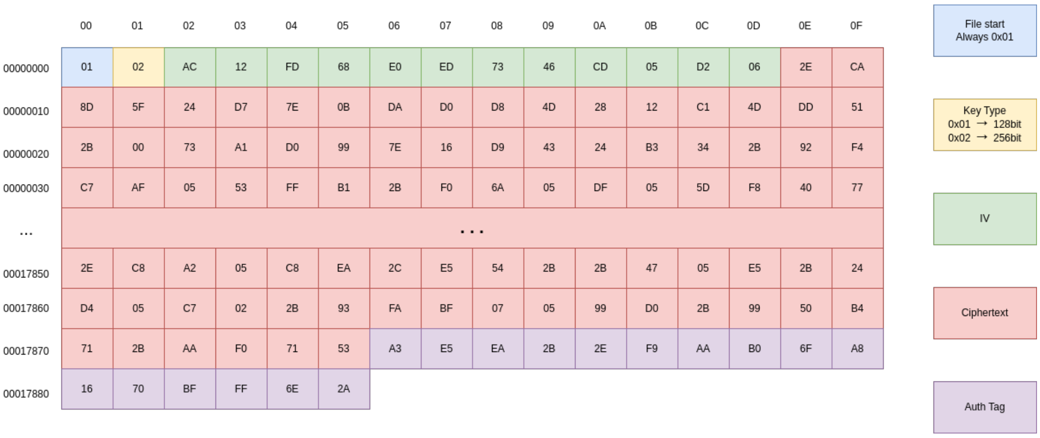


Figure 5.10: Interpration of an L shaped pattern on the Apps Lock & File Encryption pattern lock



| | 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 0A | 0B | 0C | 0D | 0E | 0F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 00000000 | 01 | 02 | AC | 12 | FD | 68 | E0 | ED | 73 | 46 | CD | 05 | D2 | 06 | 2E | CA |
| 00000010 | 8D | 5F | 24 | D7 | 7E | 0B | DA | D0 | D8 | 4D | 28 | 12 | C1 | 4D | DD | 51 |
| 00000020 | 2B | 00 | 73 | A1 | D0 | 99 | 7E | 16 | D9 | 43 | 24 | B3 | 34 | 2B | 92 | F4 |
| 00000030 | C7 | AF | 05 | 53 | FF | B1 | 2B | F0 | 6A | 05 | DF | 05 | 5D | F8 | 40 | 77 |
| ... | ... | | | | | | | | | | | | | | | |
| 00017850 | 2E | C8 | A2 | 05 | C8 | EA | 2C | E5 | 54 | 2B | 2B | 47 | 05 | E5 | 2B | 24 |
| 00017860 | D4 | 05 | C7 | 02 | 2B | 93 | FA | BF | 07 | 05 | 99 | D0 | 2B | 99 | 50 | B4 |
| 00017870 | 71 | 2B | AA | F0 | 71 | 53 | A3 | E5 | EA | 2B | 2E | F9 | AA | B0 | 6F | A8 |
| 00017880 | 16 | 70 | BF | FF | 6E | 2A | | | | | | | | | | |

Figure 5.11: The structure of a file encrypted using Meta's Conceal API

encrypts the media files encryption key under the `pbkdf2_salt` entry.

The encryption key is encrypted using Meta's Conceal API, in other words AES in GCM mode. The key used for the encryption is generated through PBKDF2 with SHA-256 HMAC. The values passed to the key derivation function are shown below. The IV used in the encryption can be found in the first bytes of the `symmetric_encrypted_files_encryption_key` value, from the 3rd to the 14th byte, while the data after the 14th byte up to the last 32 bits consists the encrypted content. The last 32 bits consists the GCM Authentication Tag.

- password = PIN
- salt = the value pbkdf2_salt
- dkLen (size of the derived key in bytes) = 32
- count (number of iterations) = 4096

The SHA-256 hash of the media files encryption key is stored on the `hashed_files_encryption_key` and is used for verification of the PIN during login attempts, since the PIN and the encryption key of the media files are directly connected, the later being encrypted with a key created by using the former. The key decryption process used by the application is shown in Figure 5.12.

Since we can create the encryption key for the media files encryption key from the PIN, we can reverse the process and through brute-forcing, identify the PIN used in the application by comparing the SHA-256 hash of the decrypted media files encryption key with the value we got from `hashed_files_encryption_key`.

The encrypted media files are stored in a folder under `/data/data/com.photovault.secret.calculator/files/calculator_encrypted_DoNotDelete`. Each media file is encrypted using Meta's Conceal API. The size of the key is 256 bits and the size of the IV is 24 bits. As with the encrypted encryption key, the IV is stored in the beggining of each encrypted file, from the 3rd to the 14th index, while the data after the 14th index up to the last 32 bits consists the encrypted content. The last 32 bits consists the GCM Authentication Tag. The key used for the encryption of every media file is the same and it can be found using the brute-forcing method mentioned above. The structure of an encrypted file can be seen in Figure 5.11.

### 5.1.15 Vaulty : Hide Pictures Videos

`Vaulty : Hide Pictures Videos` [52] is an Android application providing a variety of features enabling users to hide media files. It can hide media files, it can disguise itself as a Calculator application, it provides a decoy mechanism where you can enter the application with alternative inputs and it can store online backup copies of the files hidden in the application.

After running the application for the first time the user is prompted to create a pattern which will be used for future authentications. Accessing all the hidden

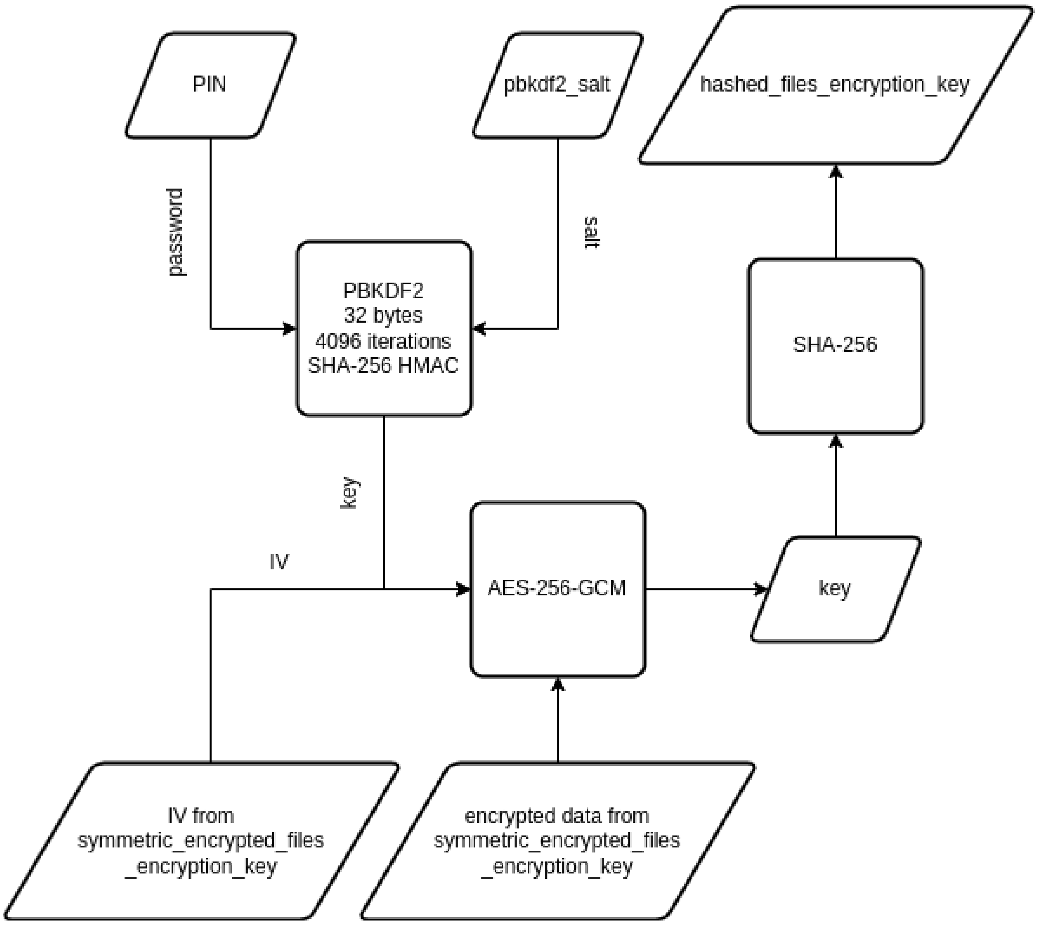


Figure 5.12: Secret Calculator Lock Vault's key decryption process

content inside the application requires authentication to the app when lauching it. Through the user settings, it is also possible to change from a pattern to a fingerprint lock.

The PIN is saved in a file under `/data/data/com.theronrogers.vaultyfree/shared_prefs/com.theronrogers.vaultyfree_preferences.xml`. The PIN is a 4 digit or a 6 digit number and it is saved under the `password_hash` entry after getting hashed using the MD5 hashing algorithm and encoded using BASE64.

Since the PIN is a relatively small at 4 or 6 digits long, it can easily be brute-forced in less than one second, even in a worst case scenario.

## 5.2 Implementation & Usage

The way we intergrated the findings related to the analysis of applications data, is by implementing a configurable and extendable component in our tool, specifically designed for performing encryption-centric logical extraction analysis on data acquired from Android devices. This component can be accessed through the `Encrypted artifacts analysis` panel as seen in Figure 5.13.

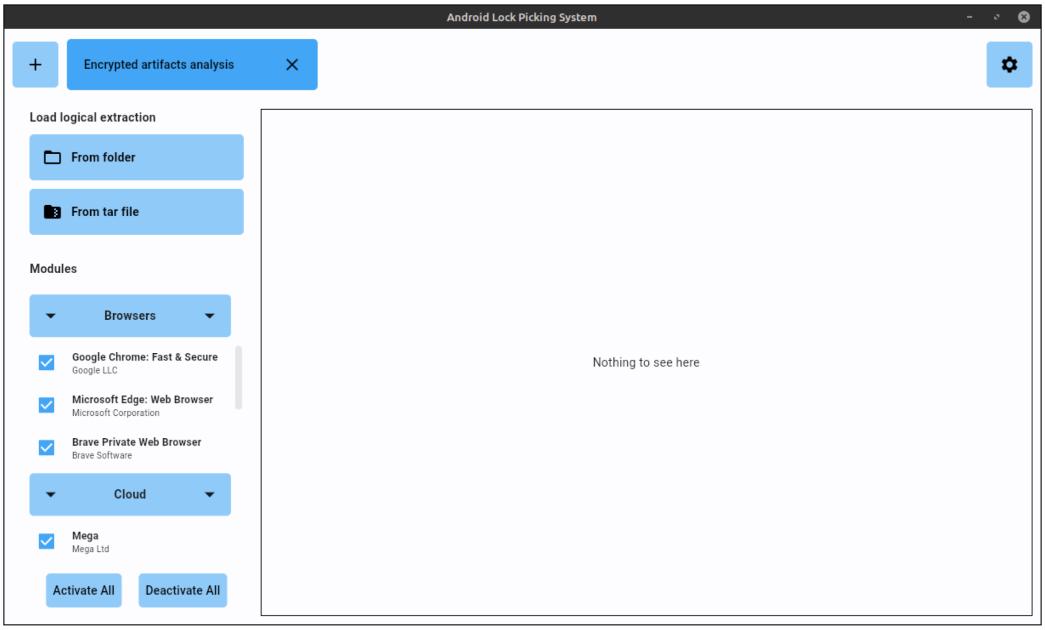


Figure 5.13: Encrypted artifacts analysis panel

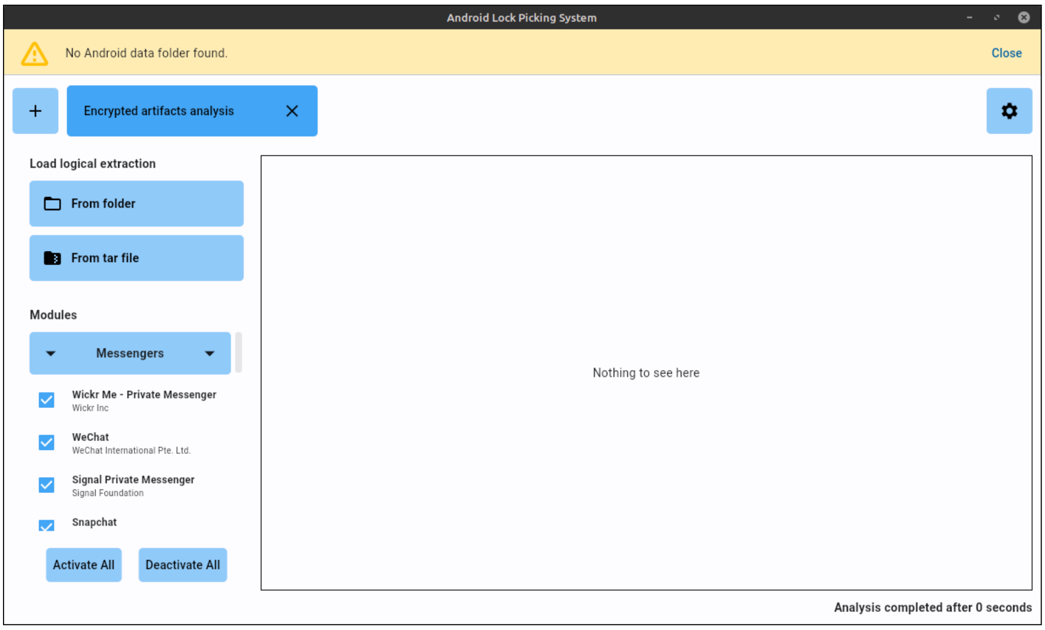
Users can load a logical extraction as a folder or as a compressed tar file containing the extracted `data` folder of an Android device. If a folder is passed, the tool will search all subdirectories of that folder until it finds a suitable Android `data` folder, if none is found, an error message will be displayed to the user, as shown in Figure 5.14.

If a tar file is passed the file will first be decompressed on a temporary folder in the output folder of the tool, before the logical extraction analysis starts. If after the decompression the extracted folder does not contain a suitable Android `data` folder, an error message will be displayed to the user, as shown in Figure 5.14.

During analysis, the tool will search and try to decrypt artifacts related to any module that was enabled when we loaded the logical extraction. In addition to the `Encrypted artifacts analysis` panel, modules can also be enabled or disabled through the configuration file or through the `Settings` panel. If a change related to the modules occurs in the `Settings` panel, it will be displayed in any new `Encrypted artifacts analysis` panel that the user opens.

During the analysis, users will be able to see the progress of the analysis process via the progress bar shown at the botttom of the panel which displays the modules remaining to be executed.

Any executed module with successfully located artifacts will be displayed in the modules list on the left side of the panel. While the analysis is ongoing users can view the results of the already executed modules by clicking on their specific



No Android data folder found.

☑ Wickr Me - Private Messenger
Wickr Inc

☑ WeChat
WeChat International Pte. Ltd.

☑ Signal Private Messenger
Signal Foundation

☑ Snapchat

Nothing to see here

Analysis completed after 0 seconds

Figure 5.14: The encrypted artifacts analysis panel error message banner

module button, which will then open the module's view. Any executed module will be presented with a colored bar on the bottom of its specific button as shown in Figure 5.15. The colors idicate the level of success the module achieved.

- Green: the module completed execution successfully and was able to exctract encrypted data from tha artifacts it found.
- Orange: although the module completed execution successfully and artifacts were found, further actions are required by the user for the decryption of available artifacts, usually through the module's view extension.
- Red: although the module completed and artifacts that could be decrypted were found, no decryption was possible.

If a module does not find any artifacts it will not be displayed in the modules list. If a module fails while searching for artifacts it will be dispalyed in the modules list with its button being greyed out and disabled to indicate that a failure happenend while searching for artifacts.

Inside a module's view users will see the `Artifacts` and `Results` tables. The artifacts found by the module will be displayed under the `Artifacts` table, while any results related to successfully decrypted data will be displayed under the `Results` table as shown in Figure 5.15. In occasions where a module includes a `view extension`, it will be displayed under the `Results` table.

The `Artifacts` table provides three columns.

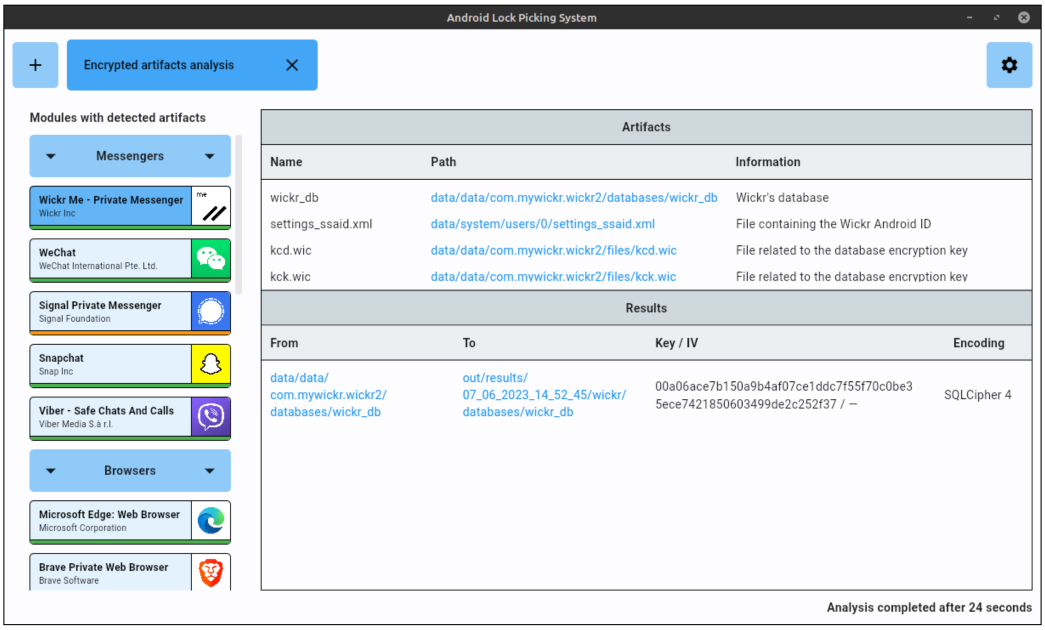
- Name = name of the artifact
- Path = path where the artifact was found (can be relative or full depending on the settings option)
- Information = information about the artifact

The `Results` table provides four columns.

- From = the artifact or data that was decrypted
- To = where (path) or to what value it was decrypted
- Key / IV = key and IV used for decryption (if applicable)
- Encoding = encryption scheme / encoding used

Any paths displayed inside the `Artifacts` and `Results` tables are clickable, if they exist as actual paths on the users filesystem. If a user clicks on a displayed clickable path, a new operating system window will be automatically opened on that path.

The encrypted artifacts analysis panel during the analysis process can be seen in Figure 5.15.


Android Lock Picking System
+
Encrypted artifacts analysis
Modules with detected artifacts
Messengers
Wickr Me - Private Messenger
Wickr Inc
WeChat
WeChat International Pte. Ltd.
Signal Private Messenger
Signal Foundation
Snapchat
Snap Inc
Viber - Safe Chats And Calls
Viber Media S.à r.l.
Browsers
Microsoft Edge: Web Browser
Microsoft Corporation
Brave Private Web Browser
Brave Software

Artifacts

| Name | Path | Information |
|---|---|---|
| wickr_db | data/data/com.mywickr.wickr2/databases/wickr_db | Wickr's database |
| settings_ssaid.xml | data/system/users/0/settings_ssaid.xml | File containing the Wickr Android ID |
| kcd.wic | data/data/com.mywickr.wickr2/files/kcd.wic | File related to the database encryption key |
| kck.wic | data/data/com.mywickr.wickr2/files/kck.wic | File related to the database encryption key |

Results

| From | To | Key / IV | Encoding |
|---|---|---|---|
| data/data/com.mywickr.wickr2/databases/wickr_db | out/results/07_06_2023_14_52_45/wickr/databases/wickr_db | 00a06ace7b150a9b4af07ce1ddc7f55f70c0be35ece7421850603499de2c252f37 / – | SQLCipher 4 |

Analysis completed after 24 seconds

Figure 5.15: The encrypted artifacts analysis panel during the analysis process

This component is extendable through the configuration file of the tool. Developers can add new code responsible for locating and decrypting a specific application's artifacts by adding a new `module` to our tool. A module can be added by creating a new `module` entry under the `modules` field in the configuration file and by adding a resspective Python file in the `modules` folder of the tool. The module entry should be similar to the one shown below.

```
"example": {
    "id": "example_id",
    "name": "Example Name",
    "developer": "Example Inc",
    "package": "com.example",
    "category": "Examples",
    "view_extension": null,
    "enabled": true
}
```

The way each entry is linked to its Python implementation file is by the value of the `id` entry of the module, which has to be the same as the filename of its Python implementation file inside the `modules` folder, excluding the .py extension.

More information related to the configurability and extendability of this component can be found in Appendix B.

## 5.3 Evaluation

The way we evaluated this componenet of the tool is by following the steps shown below.

1. Installed the applications that were mentioned in two devices (both rooted)[1]
    - Xiaomi A2 Lite device running Android 10
    - Gneymotion emulated Pixel 5 device running Android 11
2. Executed the applications so that some encrypted artifacts were created
3. Performed a logical extraction of the complete data inside the devices
4. Loaded the logical extractions in our tool and in ALEAPP
5. Compared the results of the two logical extractions loaded in our tool with each other, to see that the same artifacts could be extracted from different version of Android

[1] For validating the gesture.key bruteforcing technique we used a separate emulated Pixel 2 device running Android 6 with root access.

6. Compared the results of the logical extractions loaded in our tool and the logical extractions that were loaded in ALEAPP

The versions of the applications that were installed in both devices, are shown in Figure 5.16. The versions are the latest as of 10/5/2023.

The way we validated and compared the results is by verifying the decryption of the located encrypted artifacts and data using `CyberChef` [30] and other appropriate tools to open and view the decrypted artifacts.

For the located databases that were encrypted using SQLCipher, we confirmed that the databases shown in the `From` column were encrypted by trying to open the databases using an online SQLite viewer [61] and by trying to read the first bytes of the encrypted database file using a simple text editor. In both cases the database files were unreadable. In a similar way, we confirmed that the resulting databases shown in the `To` column were decrypted successfully by opening them using the online SQLite viewer mentioned previously. We also confirmed that we could manually decrypt the databases shown in the `From` column by using the keys displayed in the `Key/IV` column and the software tools `SQLiteManager` [78] and `DB Browser for SQLite` [66] on Windows. In both cases the database file were readable.

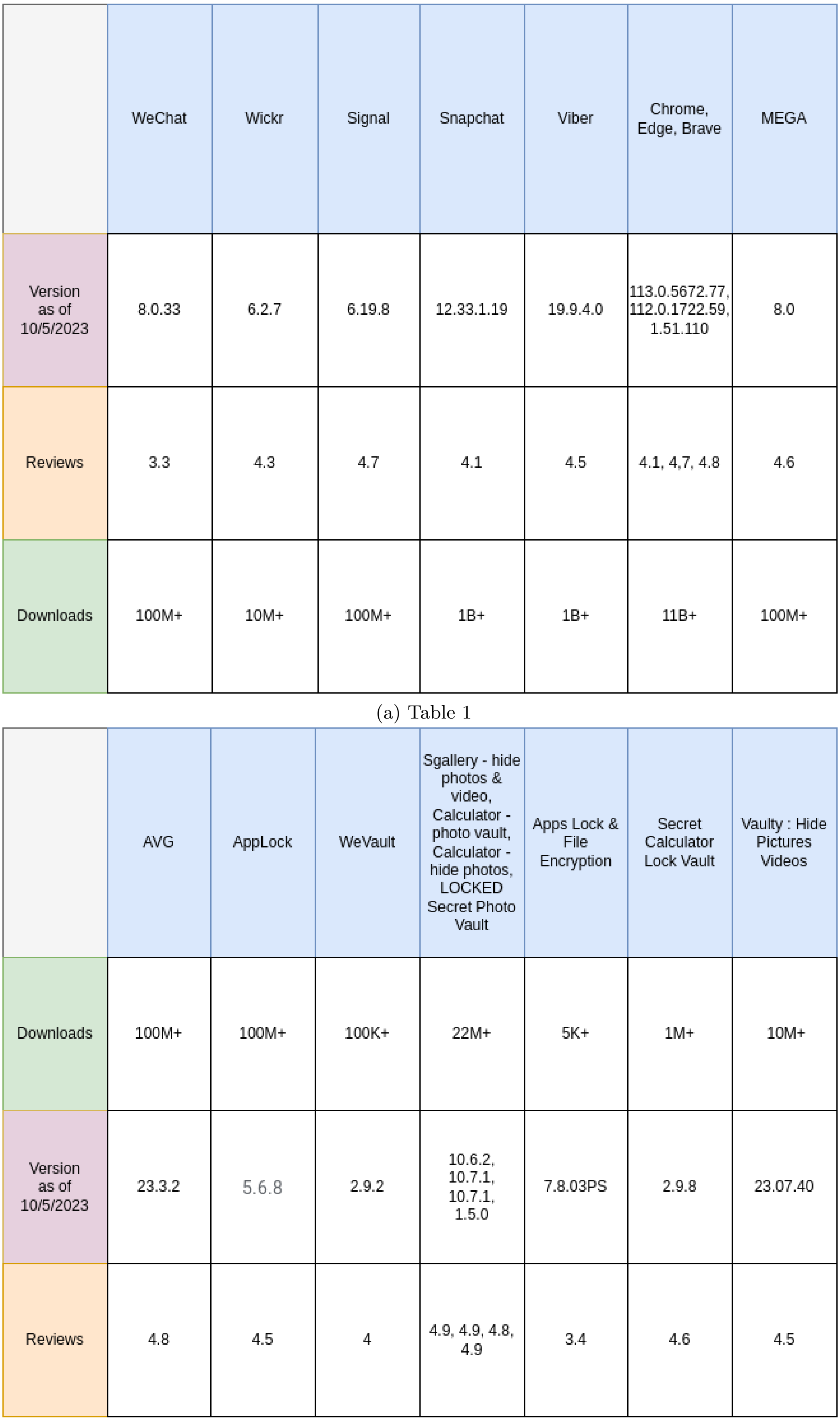
For the encrypted multimedia files that were located, we confirmed that the encrypted images, videos and audio files that were found, were encrypted by tring to open them using the standard multimedia applications of Ubuntu while having all the apppropriate codecs installed. In a similar way, we confirmed that the resulting multimedia files shown in the `To` column were decrypted successfully by opening them using the same multimedia applications. We also confirmed that we could manually decrypt the multimedia files shown in the `From` column by using the keys displayed in the `Key/IV` column and CyberChef to decrypt the files using the encryption scheme shown in the `Encoding` column.

For the encrypted binary files that were located we resorted to verifying that the decryption of the file shown in the `From` column using the keys displayed in the `Key/IV` column with CyberChef and the encryption scheme shown in the `Encoding` column, results in the same file as the one shown in the `To` column.

For any arbitrary encrypted data retrieved from XML files, databases or other files, we verified that the data shown in the `To` column was the same as the data we got after decrypting the data shown in `From` column, using the keys displayed in the `Key/IV` column with CyberChef and the encryption scheme shown in the `Encoding` column.

From the comparisons between the two logical extractions being loaded and examined on our tool we didn't notice any difference in the results.

From the comparisons between the results of our tool and ALEAPP, we noticed that ALEAPP as of 10/5/2023 didn't support 7 out of the 19 applications available in the logical extractions. On the other hand our tool was able to decrypt artifacts successfully for all the applications available in the logical extractions. The applications that are supported by our tool but were not supported by ALEAPP



| | WeChat | Wickr | Signal | Snapchat | Viber | Chrome, Edge, Brave | MEGA |
|---|---|---|---|---|---|---|---|
| Version as of 10/5/2023 | 8.0.33 | 6.2.7 | 6.19.8 | 12.33.1.19 | 19.9.4.0 | 113.0.5672.77,<br>112.0.1722.59,<br>1.51.110 | 8.0 |
| Reviews | 3.3 | 4.3 | 4.7 | 4.1 | 4.5 | 4.1, 4,7, 4.8 | 4.6 |
| Downloads | 100M+ | 10M+ | 100M+ | 1B+ | 1B+ | 11B+ | 100M+ |

(a) Table 1

| | AVG | AppLock | WeVault | Sgallery - hide photos & video,<br>Calculator - photo vault,<br>Calculator - hide photos,<br>LOCKED Secret Photo Vault | Apps Lock & File Encryption | Secret Calculator Lock Vault | Vaulty : Hide Pictures Videos |
|---|---|---|---|---|---|---|---|
| Downloads | 100M+ | 100M+ | 100K+ | 22M+ | 5K+ | 1M+ | 10M+ |
| Version as of 10/5/2023 | 23.3.2 | 5.6.8 | 2.9.2 | 10.6.2,<br>10.7.1,<br>10.7.1,<br>1.5.0 | 7.8.03PS | 2.9.8 | 23.07.40 |
| Reviews | 4.8 | 4.5 | 4 | 4.9, 4.9, 4.8, 4.9 | 3.4 | 4.6 | 4.5 |

(b) Table 2

Figure 5.16: Summary of applications studied

are listed below.

- Wickr
- Wechat
- Signal
- AppLock
- WeVault
- Apps Lock & File Encryption
- Secret Calculator Lock Vault

As of June 14, any encryption related application supported by ALEAPP, is also supported by our tool.

# Chapter 6

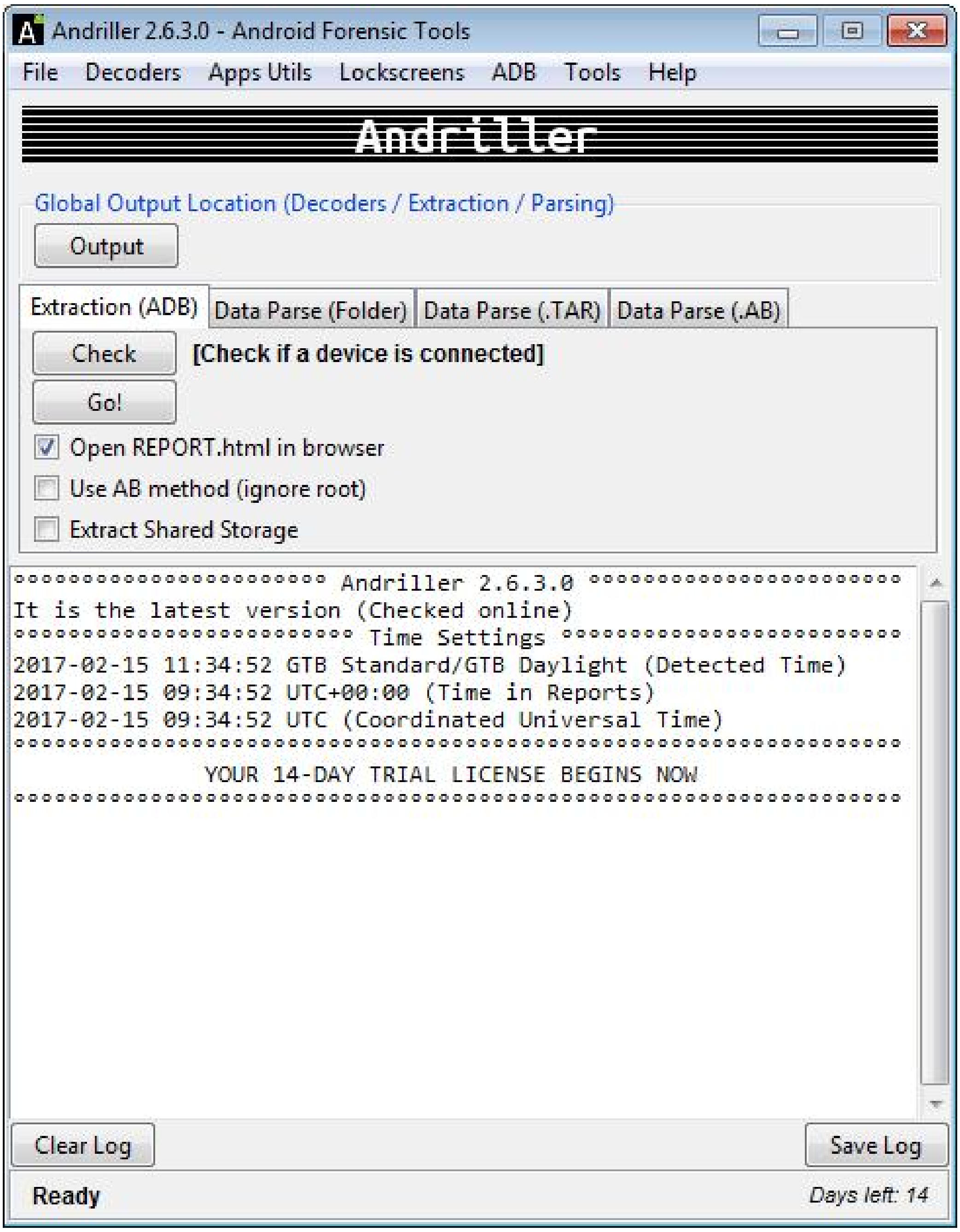
# Related Work

There are a several Android forensics related tools available currently, some of them commercial, like Belkasoft X and Cellebrite UFED. Some tools are open source like Andriller [72] and ALEAPP. We chose to compare our tool to the two afformentioned open source tools.

## 6.1 Andriller

Andriller, is a software utility, implemented in Python, which provides a collection of forensic tools for smartphones. The tool has the capability to perform forensically sound, data acquisition from Android devices. It offers various features, including the ability to crack Lockscreen patterns, PIN codes, and passwords. It also provides custom decoders for extracting data from databases of Android, iOS, and PC Windows application. The extracted data can be decoded and presented in reports available in HTML and Excel formats. For a comprehensive overview of the tool's features, please refer to the list provided below.

- Extraction of data from non-rooted devices using Android Backup.
- Extraction of data with root permissions, using options such as ADB (as root).
- Analysis of extracted data.
- Decryption of encrypted WhatsApp databases (e.g., converting .crypt to .crypt12, with the appropriate key file).
- Lockscreen pattern, PIN, and password cracking (excluding gatekeeper).
- Extraction of Android backup files.
- Screen capture of an Android device's display screen.



[Check if a device is connected]

☑ Open REPORT.html in browser

□ Use AB method (ignore root)

□ Extract Shared Storage

```
°°°°°°°°°°°°°°°°°°°°°°° Andriller 2.6.3.0 °°°°°°°°°°°°°°°°°°°°°°°
It is the latest version (Checked online)
°°°°°°°°°°°°°°°°°°°°°°°°° Time Settings °°°°°°°°°°°°°°°°°°°°°°°°°
2017-02-15 11:34:52 GTB Standard/GTB Daylight (Detected Time)
2017-02-15 09:34:52 UTC+00:00 (Time in Reports)
2017-02-15 09:34:52 UTC (Coordinated Universal Time)
°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°
              YOUR 14-DAY TRIAL LICENSE BEGINS NOW
°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°
```

Ready

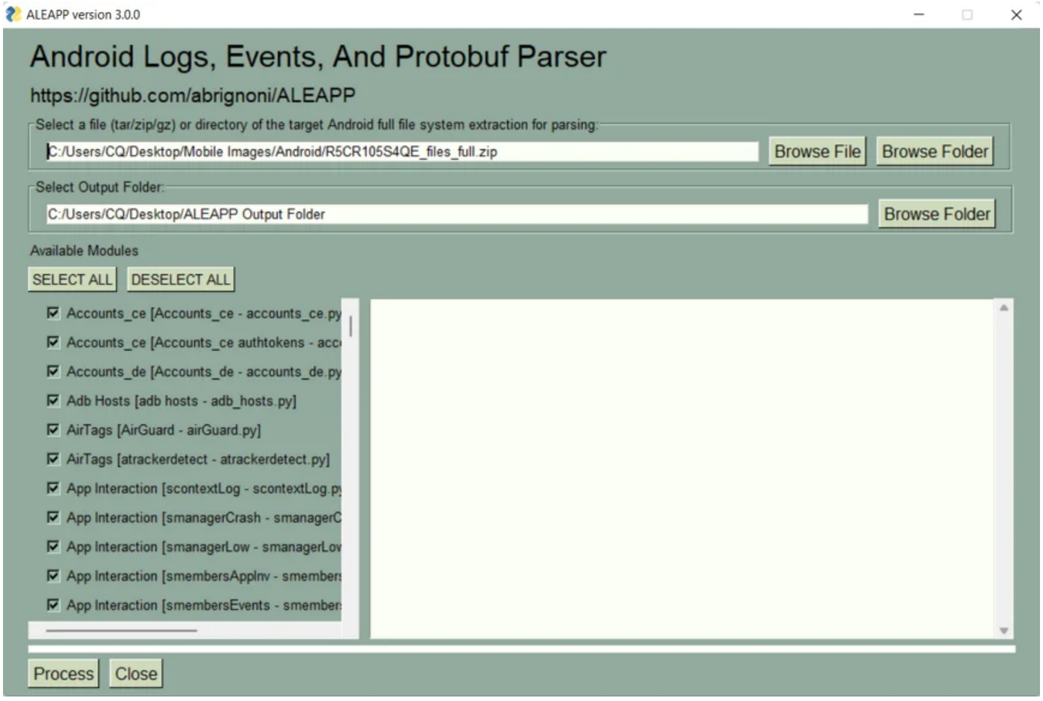
Days left: 14

Figure 6.1: Andriller home screen

## 6.2 ALEAPP

Android Logs Events And Protobuf Parser (ALEAPP) is a tool, implemented in Python, that utilizes a collection of scripts used for parsing Android application data and Android OS components data. The tool can parse logical file systems, tar and zip extractions and provide reports in html and csv formats. Each script, also referenced as an `artifact plugin` by the tool developers, is responsible for parsing the data of a specific application or system component.

As the developers of the tool state on their GitHub page: “this tool is the result of a collaborative effort of many people in the DFIR community”. This means, other developers can contibute new artifact plugins to the tool by following the relative documentation. ALEAPP is used as a plugin by other digital forensics suites such as Autopsy.



ALEAPP version 3.0.0

Android Logs, Events, And Protobuf Parser

https://github.com/abrignoni/ALEAPP

Select a file (tar/zip/gz) or directory of the target Android full file system extraction for parsing:
C:/Users/CQ/Desktop/Mobile Images/Android/R5CR105S4QE_files_full.zip

Select Output Folder:
C:/Users/CQ/Desktop/ALEAPP Output Folder

Available Modules

☑ Accounts_ce [Accounts_ce - accounts_ce.py
☑ Accounts_ce [Accounts_ce authtokens - acc
☑ Accounts_de [Accounts_de - accounts_de.py
☑ Adb Hosts [adb hosts - adb_hosts.py]
☑ AirTags [AirGuard - airGuard.py]
☑ AirTags [atrackerdetect - atrackerdetect.py]
☑ App Interaction [scontextLog - scontextLog.p
☑ App Interaction [smanagerCrash - smanagerC
☑ App Interaction [smanagerLow - smanagerLov
☑ App Interaction [smembersAppInv - smember
☑ App Interaction [smembersEvents - smember

Figure 6.2: ALEAPP home screen

## 6.3 ALPS Comparison with ALEAPP and Andriller

Although our tool specializes in encryption bypassing techniques, it has a lot of similarities with the two open source tools mentioned above, while it also provides a lot of new fetures not seen in any publicly available tool that incorporates publicly available techniques.

One such feature not available in ALEAPP, Andriller or other similar tools, is the option for incorporating `view extensions` in the modules/plugins of the tool, which provides each module with more capabilities for further actions and makes our tool even more extendable. Using a `view extensions` module developers can add custom user interface components and actions in a module's view and in that way provide users with fucntionalities that our tool does not normally support.

# Chapter 7

# Future Work & Conclusions

The `encrypted artifcats analysis` feature seems to be the most prominent feature that our tool currently supports since it allows us to perform forensic analysis on logical extractions which is on par with similar currently available tools and in some occasions it even exceeds the capabilities of the other tools. Also the configurability and extendability of this feature makes this component of the tool easy to maintain and keep updated with a reasonable level of development support. Furthermore, the findings that this feature could provide to forensic investigators could have many side benefits not immediately visible to a non forensic user, some of them are shown below.

- Brute-forced PINs, passwords and patterns could be used for other services/applications used by the target, e.g. online social media platforms

- Decryption of metadata related to the files provides users of our tool with valuable info related to the taget user actions, e.g. dates when then encryption or upload actions took place

- Enables users of our tool viewing media and messages otherwise unavailable even if access to the live unlocked device was granted, e.g. media encrypting applications

- Since the tool operates in a copy of the devices data, forensically sound search operations could be conducted on the acquired data without fear of altering the data in the device

The Lockphish technique for coordinating lockscreen oriented phishing attacks, is also an intresting feature incorporated in our tool, although it has disadvantages and blockages over the cases where it could be used lawfully since most jurisdictions do not allow for activities that target users in a malicious and deceiving manner. Even so though, this feature could be used for educational purposes and for training users and preparing them for this kind of attacks.

From the three encryption bypassing techniques that we implemented in our tool, the bruteforcing technique using a USB Rubber Ducky is currently the least viable one since it does not apply to many of the currently available Android devices.

These features alone make for a very beneficial component in the toolchain of a forensic investigator, however, there are still areas for improvement and future research. As the first step for the future work in the development of our tool we would like to focus on the continuous expansion of artifact support for the `encrypted artifcats analysis` feature. Since the Android applications we included constantly get updated and the Android operating system itself is also changing drastically over the years it is crucial to actively maintain the tool and keep it up to date and capable of extracting relevant artifacts even after changes have been made to the afformentioned applications or the Android OS. Later we could work on the addition of new features such as memory dump analysis for encrypted artifacts, to be able to extract encryption keys from the live memory of mobile devices.

Currently the tool is supported only on Ubuntu Linux and other Ubuntu based distributions. Our goal is to make the tool cross-platform and even provide a way for users to use it as a web application further in the future, so that it can be more easily available to a wider range of users.

Also another potential option for future work is expanding the tool's `encrypted artifacts analysis` feature to support other operating systems, such as iOS or Windows. By incorporating the ability to analyze multiple platforms, the tool would become even more versatile and valuable for digital forensic investigators. Some of our colleagues have already started incorporating some Windows modules in the `encrypted artifcats analysis` feature, which shows that the tool is ready for an expansion in supporting other operating systems and is also a proof of the configurability and extendability of the tool.

Regarding the user interface and the usability of the tool, although we tried our best to make the tool user friendly and intuitive, we are sure that we could benefit from further improvements. Making the tool more intuitive, visually appealing, and easier to navigate would enhance its accessibility to both experienced forensic analysts and those new to the field.

Finally we plan on documenting all the features of the tool in a user and developer guide for aiding future users and developers respectively in the usage of the tool.

# Appendices

# Appendix A

# Abbreviations

**ADB** Android Debug Bridge

**AES** Advanced Encryption Standard

**ALEAPP** Android Logs Events And Protobuf Parser

**AOSP** Android Open Source Project

**CBC** Cipher Block Chaining

**CSS** Cascading Style Sheets

**DES** Data Encryption Standard

**DFIR** Digital Forensics and Incident Response

**ECB** Electronic Codebook

**ESN** Electronic Serial Number

**FOSS** Free and Open-Source Software

**GCM** Galois/Counter Mode

**HMAC** Hash-based Message Authentication Code

**HTML** HyperText Markup Language

**IMEI** International Mobile Equipment Identity

**IP address** Internet Protocol address

**MD** Message digest

**NIST** National Institute of Standards and Technology

**OEM** Original Equipment Manufacturer

**OS** Operating System

**OTG** On-The-Go

**PC** Personal Computer

**PHP** PHP: Hypertext Preprocessor

**PIN** Personal identification number

**PKCS** Public-Key Cryptography Standards

**PKI** Public Key Infrastructure

**ROM** Read Only Memory

**SHA** Secure Hash Algorithm

**tar** Tape Archive

**U.S.** United States

**USB** Universal Serial Bus

**XML** Extensible Markup Language

# Appendix B

# ALPS Installation Instructions

### ALPS

Android Lock Picking System

### Requirements

An Ubuntu or Debian based Linux distribution and Python 3.9 or above.

### Dependencies

Make sure Python is installed, the commands below assume you are using Python 3.9, if another version of Python is used change the python3.x commands with the commands corresponding to the Python version you are using.

```
sudo apt-get install python3.9 python3.9-venv python3.9-dev
```

Some Linux packages will be needed to run the social engineering module and modules that require sqlcipher. To install them run:

```
sudo apt-get install build-essential libsqlcipher-dev
```

```
sudo apt-get install php-fpm php-cli curl
```

It is recommended to setup a python environment before installing the python dependencies. You can do it using the commands below.

```
python3.9 -m venv alps_env
```

```
source alps_env/bin/activate
```

Dependencies for your python environment are listed in requirements.txt. Install them using the below command.

```
pip install -r requirements.txt
```

### Usage

```
./alps.sh
```

### Configuration

The following values can be configured through the configuration file config.json, or through the tool's settings panel, except the modules option which can be fully configured through the configuration file and provides only the option to enable and disable modules through the settings panel.

- ngrok_api_key: value used for the API key of the ngrok service
- logging_enabled: defines if logging is enabled
- log_level: defines the logging level, it can be DEBUG, INFO or WARNING
- use_relative_paths: defines if relative or absolute paths should be displayed in the encrypted artifacts analysis panel
- modules: contains the modules used for the encrypted artifacts analysis

A custom configuration file can be used through a file named custom_config.json. If that file exists in the root directory of the tool, it will be the one used instead of config.json.

### Contributing artifact analysis modules

Each module should be declared in the configuration file config.json, as an entry in the modules object.

For example:

```
"modules": {
    "example": {
        "id": "example_id",
        "name": "Example Name",
        "developer": "Example Inc",
        "package": "com.example",
        "category": "Examples",
        "view_extension": null,
        "enabled": true
    }
    ...
}
```

Entry property definitions:

| Property | Type | Description | Required |
|---|---|---|---|
| id | String | ID linking to the module's implementation and image | Yes |
| name | String | Name of the module under analysis | Yes |
| developer | String | Developer of the module under analysis | No |
| package | String | Android package of the module under analysis | No |
| category | String | Category of the module | Yes |
| view_extension | String | Identifier linking to the module's view extension | No |
| enabled | Boolean | Declares if the module is enabled | Yes |

Each module should be implemented in a Python source file which should be added to the modules folder. The name of the source file should be the same name as the id declared on the id property of the module.

Each module can also have an image file linked to it, that image is going to be shown on the UI next to information provided for that module after a successful analysis has taken place for that corresponding module. The image file should be added to the assets/images folder. Its format should be png and its name should be the same as the id declared on the id property of the module.

ALPS identifies each module by the name (not the name property) of the entry declared in the modules object. The id property of each entry is used to identify the module file that should be loaded for that entry. The name of the entry and its id property are recommended to be the same but declaring them with different values is not prohibited.

The module's source file should implement a decrypt function which takes 3 arguments.

- Input folder: the path of the logical extraction's data folder.
- Output folder: the path where the module can store decrypted files or other artifacts found during the analysis.
- ID: id of the logical extraction analysis task (the id can be used to differentiate the task from other tasks during logging).

The decrypt function should return three arrays.

- The artifact's array containing Artifact objects, which represent the artifacts that have been found.
- The result's array containing Result objects, which represent the results that have been found.
- The data array containing arbitrary data which can later be used in a view extension if one is declared.

The Artifact class is declared in utils/artifact.py.
To initialize an artifact object you need to pass the name of the artifact, the path where it was found and a short description.

The Result class is declared in utils/result.py. To initialize a result object you need to pass the value or the path from where it got decrypted, the value or path to where it got decrypted, the key and IV combination that was used during decryption if a key and IV was used and the encoding that was used for the original encryption of the information decrypted. The key and IV combination should be passed as a KEY_IV object, the KEY_IV class is defined inside utils/result.py.

Modules can utilize the logging capabilities of the tool, by declaring and using the logging library with the ‘alps‘ logger handler.

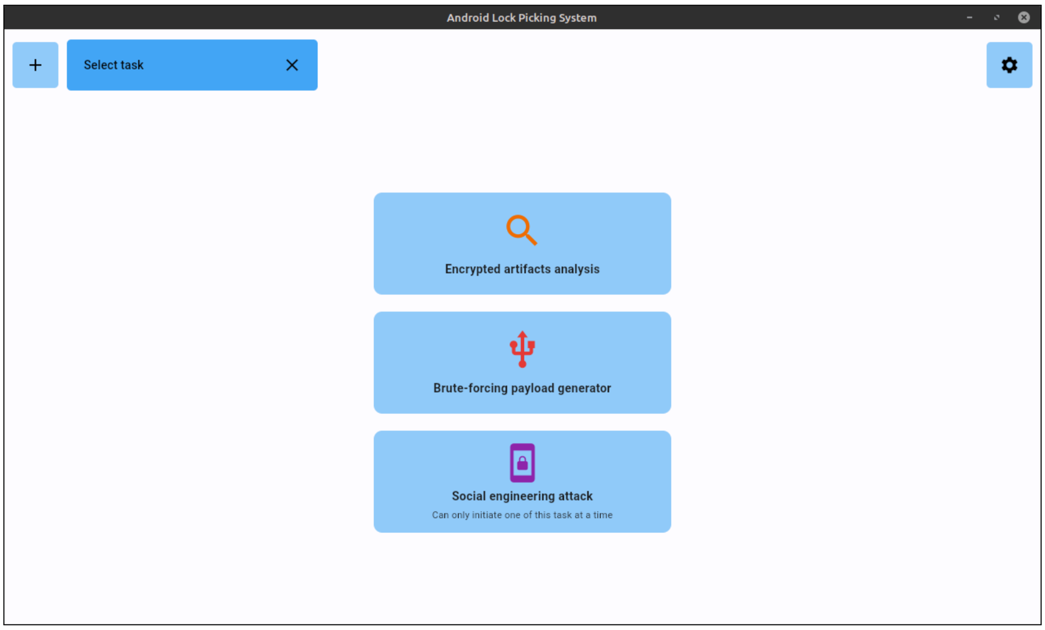
Example module implementation for finding and decrypting an artifact file named "key" under the package "com.example":

```
import logging

from utils.helper import *

from utils.artifact import Artifact
from utils.result import Result, KEY_IV

logger = logging.getLogger(’alps’)

KEY = "/data/data/com.example/key"

def decrypt(input_path, output_path, id = 0):

    artifacts = []
    results = []
    data = []

    key_path = input_path + KEY
    key_name = get_name(KEY)

    try:
        with open(key_path, "rb") as f:
            key = f.read()
            key_hex = key.hex()
    except Exception as e:
        logger.warning(log_message(id, e))
        logger.info(log_message(id,
            unable_to_extract_artifacts_or_missing_message(
                key_name)
            )
        )
        pass
    else:
        artifact = Artifact(get_name(key_path),
            android_filesystem_path(key_path),
            "The file containing the hash of the key"
        )
        artifacts.append(artifact)
        logger.info(artifact_log_message(id, artifact))

    try:

        actual_key = find_key(key_hex)

    except Exception as e:
        logger.warning(log_message(id, e))
        logger.info(log_message(id, unable_to_extract("key")))
        pass
    else:
        result = Result(key_hex, pattern, KEY_IV(), "SHA1")
        results.append(result)
        logger.info(result_log_message(id, result))

    return artifacts, results, data
```

During each artifacts analysis task run, the enabled modules are loaded dynamically. The artifacts and results returned are displayed in their corresponding artifacts and results tables on the module's view. If a view extension is provided and the data array returned is not empty, the view extension pointed by the view_extension property of the module is attached on the module's view.

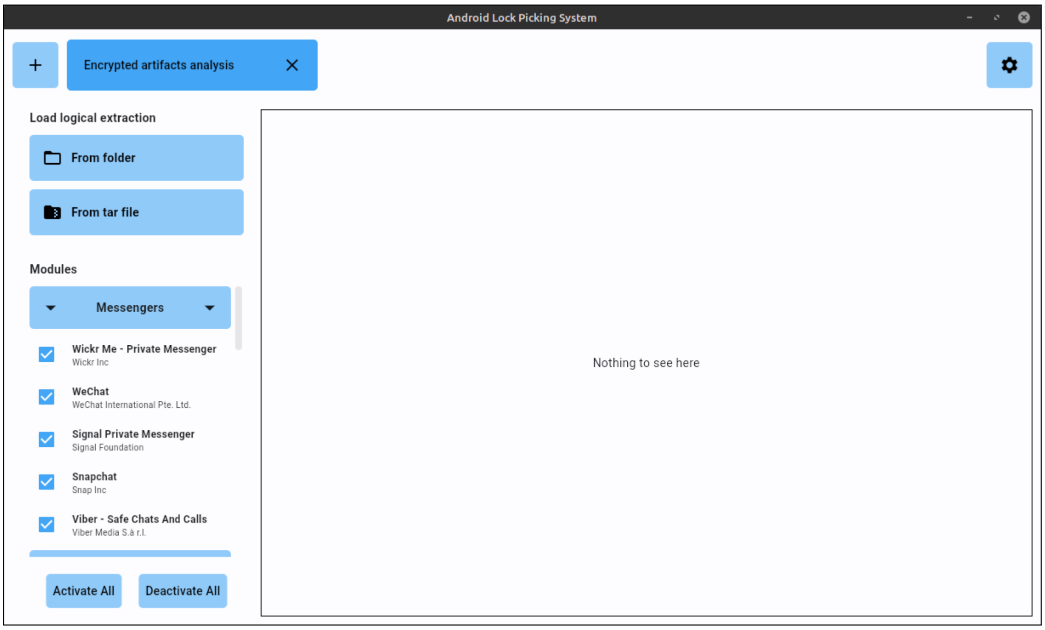
# Appendix C

# ALPS Screenshots


Android Lock Picking System
+
Select task
Encrypted artifacts analysis
Brute-forcing payload generator
Social engineering attack

Can only initiate one of this task at a time

Figure C.1: ALPS home screen



☑ Wickr Me - Private Messenger
Wickr Inc

☑ WeChat
WeChat International Pte. Ltd.

☑ Signal Private Messenger
Signal Foundation

☑ Snapchat
Snap Inc

☑ Viber - Safe Chats And Calls
Viber Media S.à r.l.

Figure C.2: Encrypted artifacts analysis expanded

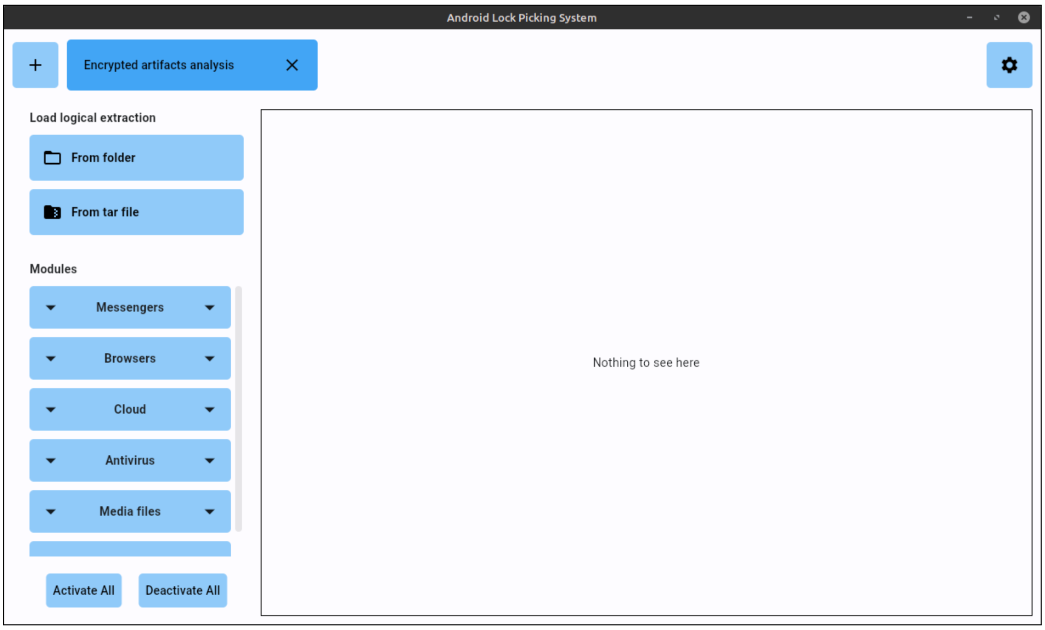
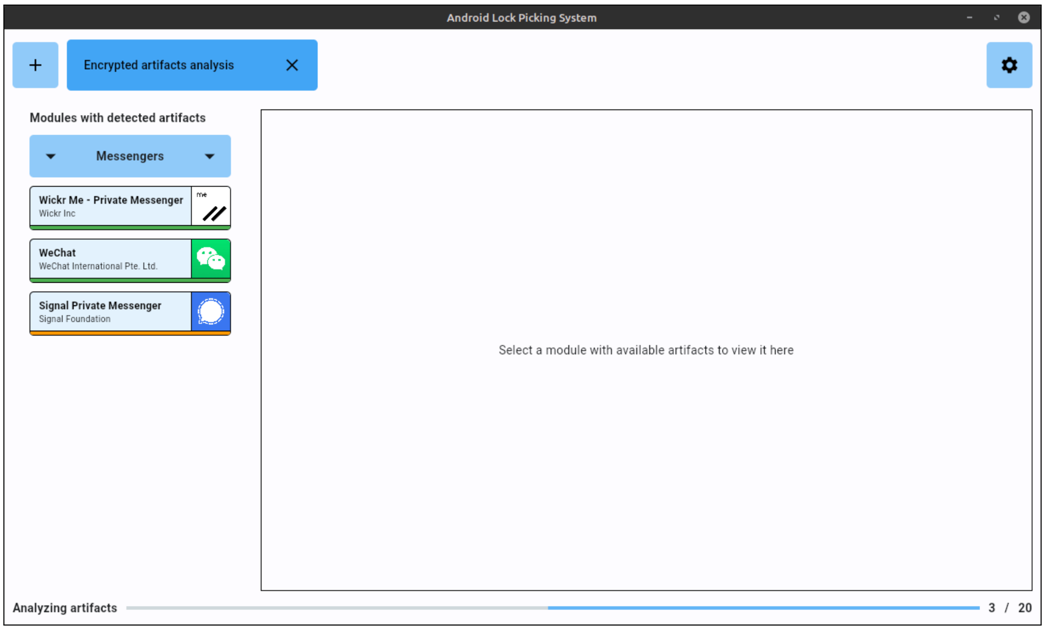
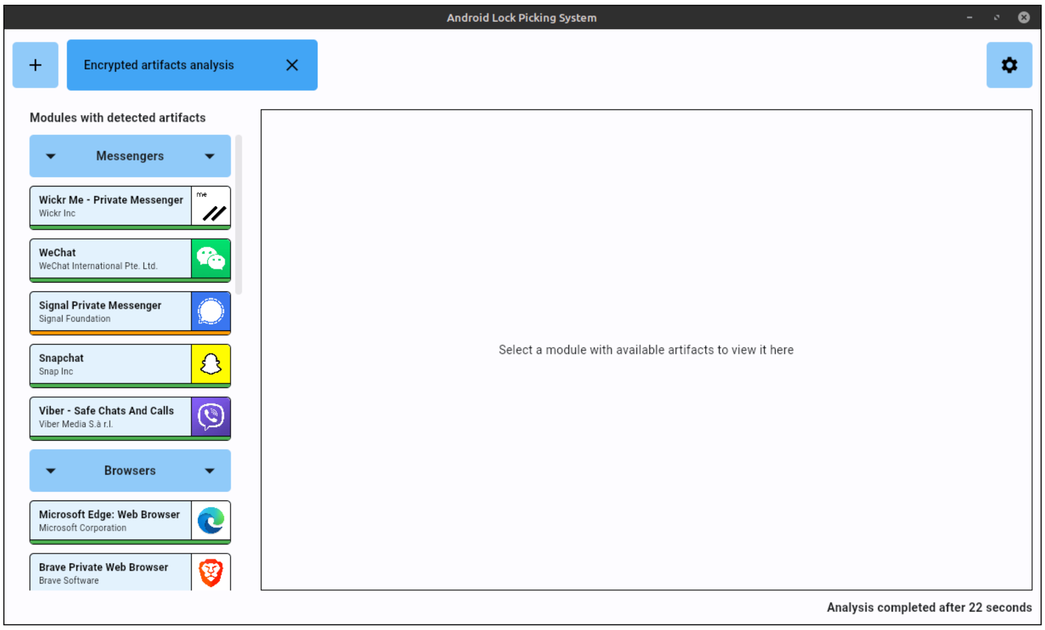


Figure C.3: Encrypted artifacts analysis collapsed


Android Lock Picking System
+
Encrypted artifacts analysis
Modules with detected artifacts
Messengers
Wickr Me - Private Messenger
Wickr Inc
WeChat
WeChat International Pte. Ltd.
Signal Private Messenger
Signal Foundation
Analyzing artifacts
3 / 20

Select a module with available artifacts to view it here

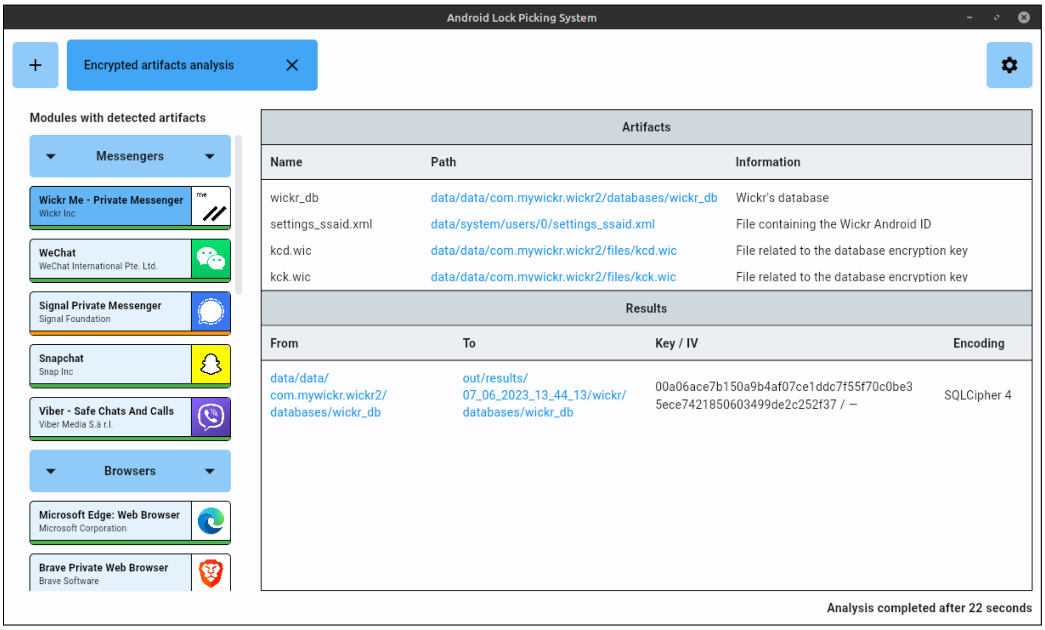
Figure C.4: Encrypted artifacts analysis - Analyzing artifacts


Android Lock Picking System
+
Encrypted artifacts analysis
Modules with detected artifacts
Messengers
Wickr Me - Private Messenger
Wickr Inc
WeChat
WeChat International Pte. Ltd.
Signal Private Messenger
Signal Foundation
Snapchat
Snap Inc
Viber - Safe Chats And Calls
Viber Media S.à r.l.
Browsers
Microsoft Edge: Web Browser
Microsoft Corporation
Brave Private Web Browser
Brave Software

Select a module with available artifacts to view it here

Analysis completed after 22 seconds

Figure C.5: Encrypted artifacts analysis - Analysis finished


Android Lock Picking System
+
Encrypted artifacts analysis
Modules with detected artifacts
Messengers
Wickr Me - Private Messenger
Wickr Inc
WeChat
WeChat International Pte. Ltd.
Signal Private Messenger
Signal Foundation
Snapchat
Snap Inc
Viber - Safe Chats And Calls
Viber Media S.à r.l.
Browsers
Microsoft Edge: Web Browser
Microsoft Corporation
Brave Private Web Browser
Brave Software

Artifacts

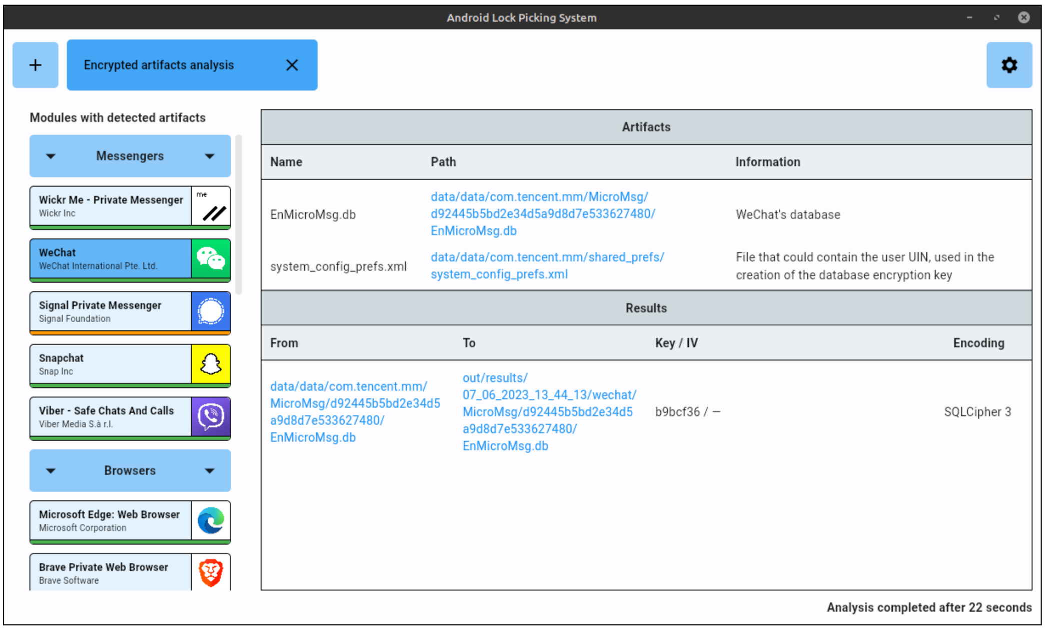
| Name | Path | Information |
|---|---|---|
| wickr_db | data/data/com.mywickr.wickr2/databases/wickr_db | Wickr's database |
| settings_ssaid.xml | data/system/users/0/settings_ssaid.xml | File containing the Wickr Android ID |
| kcd.wic | data/data/com.mywickr.wickr2/files/kcd.wic | File related to the database encryption key |
| kck.wic | data/data/com.mywickr.wickr2/files/kck.wic | File related to the database encryption key |

Results

| From | To | Key / IV | Encoding |
|---|---|---|---|
| data/data/com.mywickr.wickr2/databases/wickr_db | out/results/07_06_2023_13_44_13/wickr/databases/wickr_db | 00a06ace7b150a9b4af07ce1ddc7f55f70c0be35ece7421850603499de2c252f37 / – | SQLCipher 4 |

Analysis completed after 22 seconds

Figure C.6: Encrypted artifacts analysis - Wickr


Android Lock Picking System
+
Encrypted artifacts analysis
Modules with detected artifacts
Messengers
Wickr Me - Private Messenger
Wickr Inc
WeChat
WeChat International Pte. Ltd.
Signal Private Messenger
Signal Foundation
Snapchat
Snap Inc
Viber - Safe Chats And Calls
Viber Media S.à r.l.
Browsers
Microsoft Edge: Web Browser
Microsoft Corporation
Brave Private Web Browser
Brave Software

Artifacts

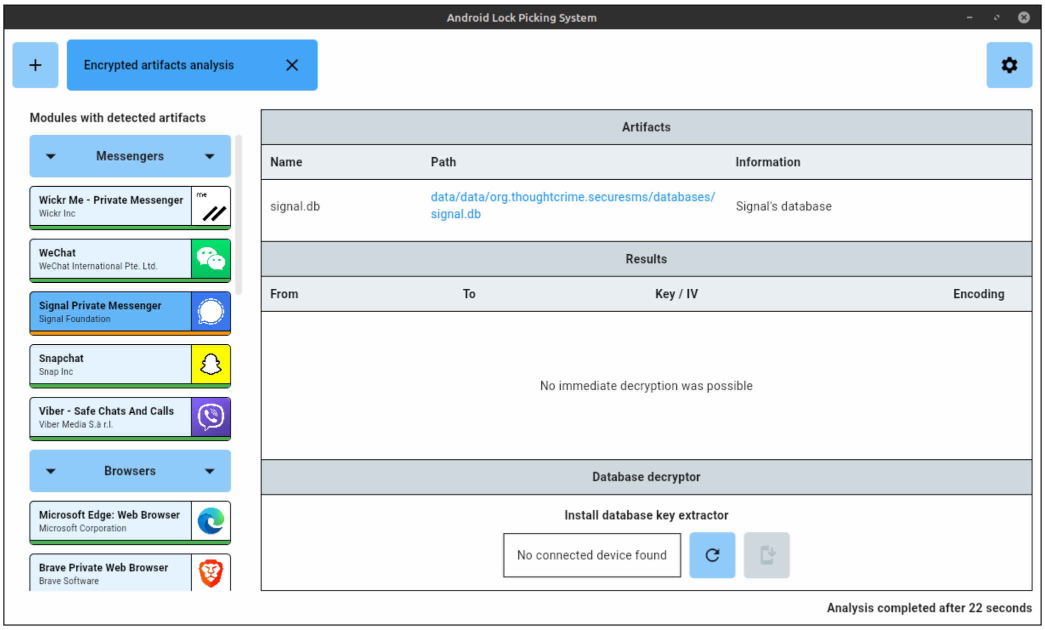
| Name | Path | Information |
|---|---|---|
| EnMicroMsg.db | data/data/com.tencent.mm/MicroMsg/d92445b5bd2e34d5a9d8d7e533627480/EnMicroMsg.db | WeChat's database |
| system_config_prefs.xml | data/data/com.tencent.mm/shared_prefs/system_config_prefs.xml | File that could contain the user UIN, used in the creation of the database encryption key |

Results

| From | To | Key / IV | Encoding |
|---|---|---|---|
| data/data/com.tencent.mm/MicroMsg/d92445b5bd2e34d5a9d8d7e533627480/EnMicroMsg.db | out/results/07_06_2023_13_44_13/wechat/MicroMsg/d92445b5bd2e34d5a9d8d7e533627480/EnMicroMsg.db | b9bcf36 / – | SQLCipher 3 |

Analysis completed after 22 seconds

Figure C.7: Encrypted artifacts analysis - WeChat



Android Lock Picking System

+
Encrypted artifacts analysis

Modules with detected artifacts

Messengers
Wickr Me - Private Messenger
Wickr Inc
WeChat
WeChat International Pte. Ltd.
Signal Private Messenger
Signal Foundation
Snapchat
Snap Inc
Viber - Safe Chats And Calls
Viber Media S.à r.l.
Browsers
Microsoft Edge: Web Browser
Microsoft Corporation
Brave Private Web Browser
Brave Software

Artifacts

| Name | Path | Information |
|---|---|---|
| signal.db | data/data/org.thoughtcrime.securesms/databases/signal.db | Signal's database |

Results

| From | To | Key / IV | Encoding |
|---|---|---|---|

No immediate decryption was possible

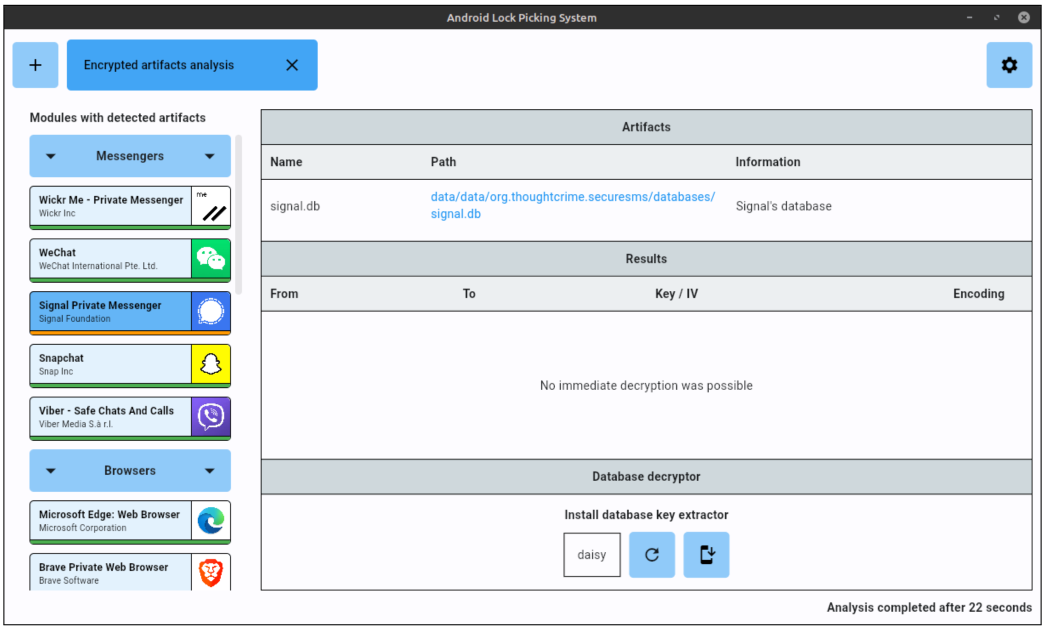
Database decryptor

Install database key extractor

No connected device found

Analysis completed after 22 seconds

Figure C.8: Encrypted artifacts analysis - Signal



Android Lock Picking System

+
Encrypted artifacts analysis

Modules with detected artifacts

Messengers
Wickr Me - Private Messenger
Wickr Inc
WeChat
WeChat International Pte. Ltd.
Signal Private Messenger
Signal Foundation
Snapchat
Snap Inc
Viber - Safe Chats And Calls
Viber Media S.à r.l.
Browsers
Microsoft Edge: Web Browser
Microsoft Corporation
Brave Private Web Browser
Brave Software

Artifacts

| Name | Path | Information |
|---|---|---|
| signal.db | data/data/org.thoughtcrime.securesms/databases/signal.db | Signal's database |

Results

| From | To | Key / IV | Encoding |
|---|---|---|---|

No immediate decryption was possible

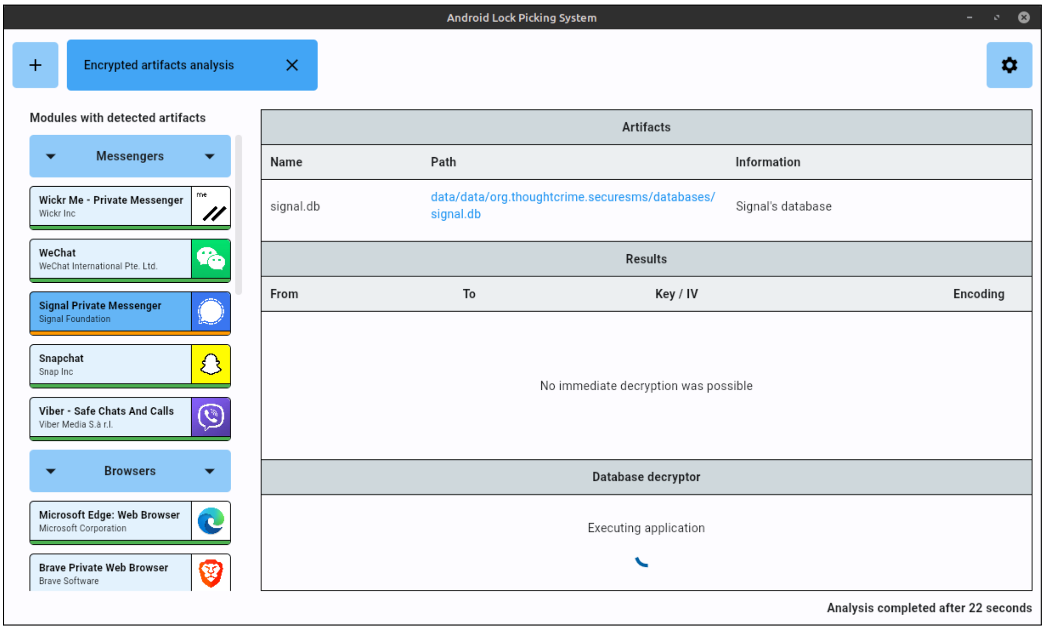
Database decryptor

Install database key extractor

daisy

Analysis completed after 22 seconds

Figure C.9: Encrypted artifacts analysis - Signal: device found


Android Lock Picking System
+
Encrypted artifacts analysis
Modules with detected artifacts
Messengers
Wickr Me - Private Messenger
Wickr Inc
WeChat
WeChat International Pte. Ltd.
Signal Private Messenger
Signal Foundation
Snapchat
Snap Inc
Viber - Safe Chats And Calls
Viber Media S.à r.l.
Browsers
Microsoft Edge: Web Browser
Microsoft Corporation
Brave Private Web Browser
Brave Software

Artifacts

| Name | Path | Information |
|---|---|---|
| signal.db | data/data/org.thoughtcrime.securesms/databases/signal.db | Signal's database |

Results

| From | To | Key / IV | Encoding |
|---|---|---|---|

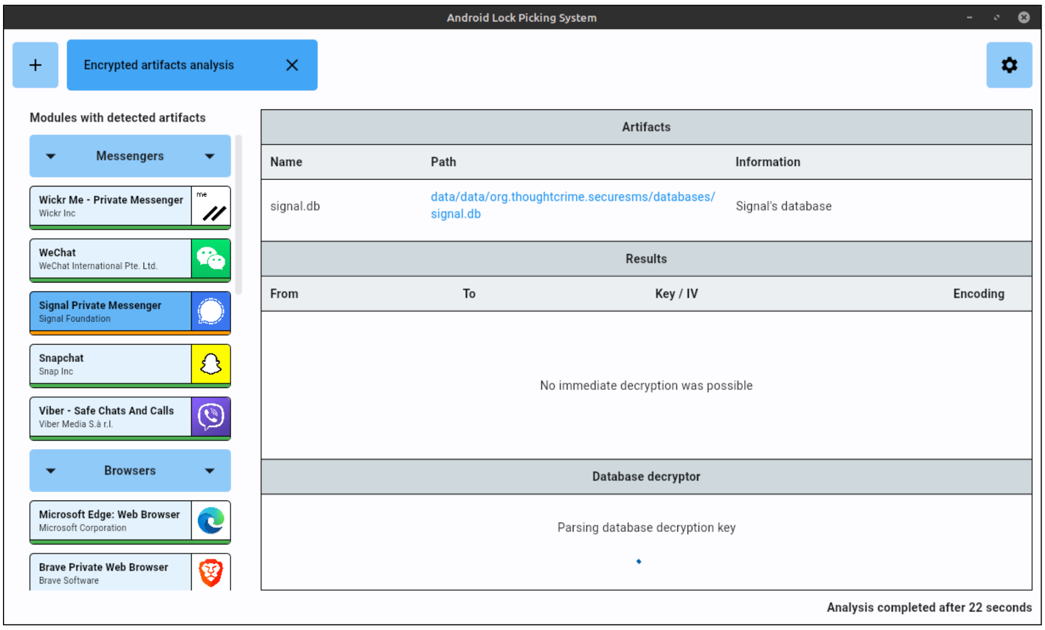
No immediate decryption was possible

Database decryptor

Executing application

Analysis completed after 22 seconds

Figure C.10: Encrypted artifacts analysis - Signal: executing application


Android Lock Picking System
+
Encrypted artifacts analysis
Modules with detected artifacts
Messengers
Wickr Me - Private Messenger
Wickr Inc
WeChat
WeChat International Pte. Ltd.
Signal Private Messenger
Signal Foundation
Snapchat
Snap Inc
Viber - Safe Chats And Calls
Viber Media S.à r.l.
Browsers
Microsoft Edge: Web Browser
Microsoft Corporation
Brave Private Web Browser
Brave Software

Artifacts

| Name | Path | Information |
|---|---|---|
| signal.db | data/data/org.thoughtcrime.securesms/databases/signal.db | Signal's database |

Results

| From | To | Key / IV | Encoding |
|---|---|---|---|

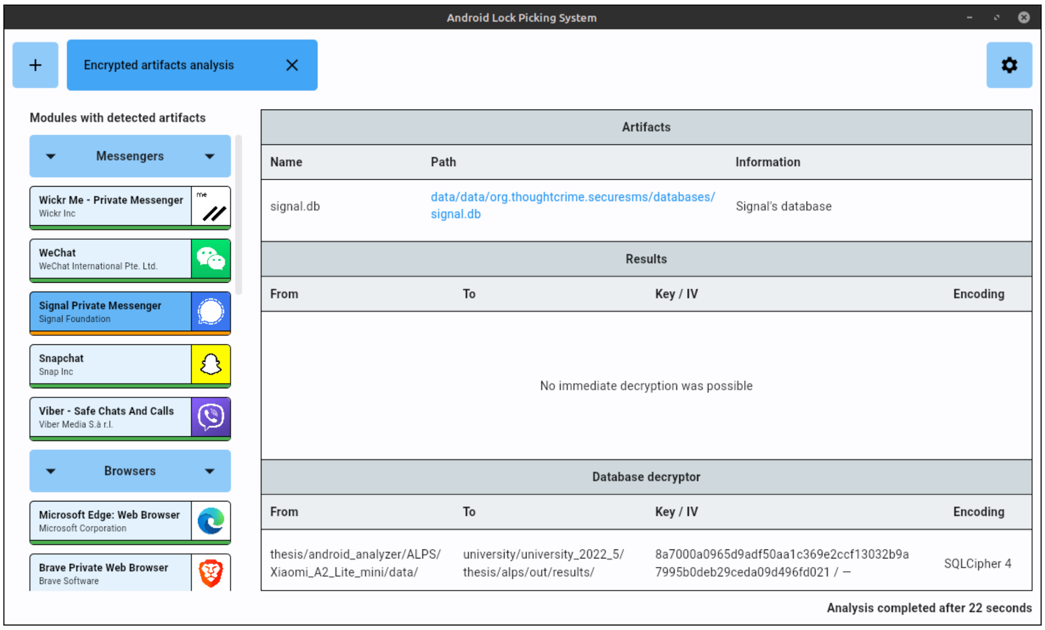
No immediate decryption was possible

Database decryptor

Parsing database decryption key

Analysis completed after 22 seconds

Figure C.11: Encrypted artifacts analysis - Signal: parsing key



Android Lock Picking System

Encrypted artifacts analysis

Modules with detected artifacts

Messengers

Wickr Me - Private Messenger<br>Wickr Inc

WeChat<br>WeChat International Pte. Ltd.

Signal Private Messenger<br>Signal Foundation

Snapchat<br>Snap Inc

Viber - Safe Chats And Calls<br>Viber Media S.à r.l.

Browsers

Microsoft Edge: Web Browser<br>Microsoft Corporation

Brave Private Web Browser<br>Brave Software

Artifacts

| Name | Path | Information |
|---|---|---|
| signal.db | data/data/org.thoughtcrime.securesms/databases/signal.db | Signal's database |

Results

| From | To | Key / IV | Encoding |
|---|---|---|---|

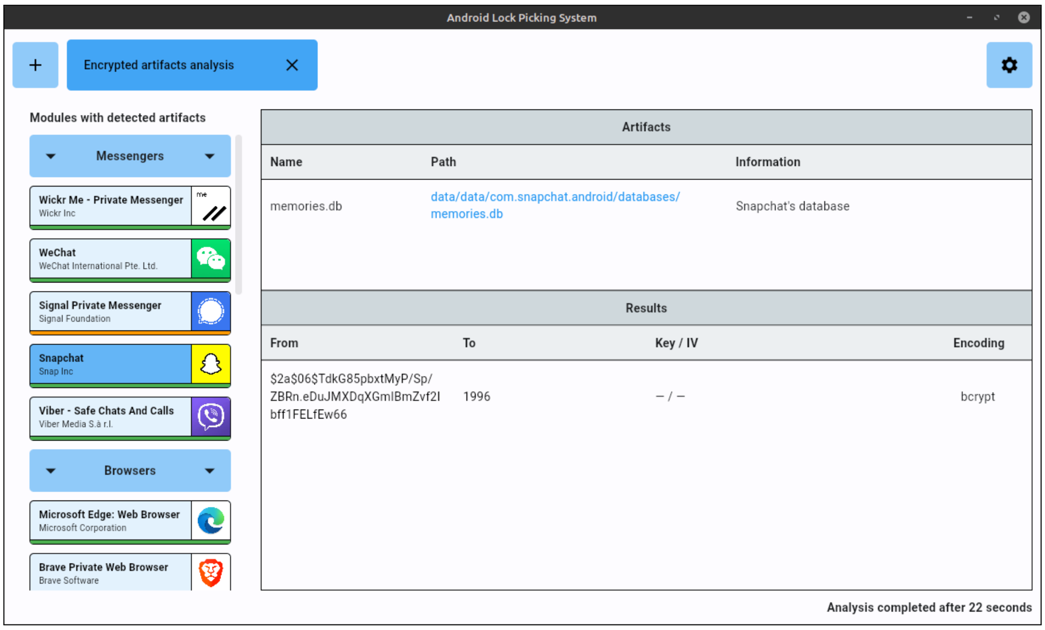
No immediate decryption was possible

Database decryptor

| From | To | Key / IV | Encoding |
|---|---|---|---|
| thesis/android_analyzer/ALPS/Xiaomi_A2_Lite_mini/data/ | university/university_2022_5/thesis/alps/out/results/ | 8a7000a0965d9adf50aa1c369e2ccf13032b9a7995b0deb29ceda09d496fd021 / — | SQLCipher 4 |

Analysis completed after 22 seconds

Figure C.12: Encrypted artifacts analysis - Signal: key found



Android Lock Picking System

Encrypted artifacts analysis

Modules with detected artifacts

Messengers

Wickr Me - Private Messenger<br>Wickr Inc

WeChat<br>WeChat International Pte. Ltd.

Signal Private Messenger<br>Signal Foundation

Snapchat<br>Snap Inc

Viber - Safe Chats And Calls<br>Viber Media S.à r.l.

Browsers

Microsoft Edge: Web Browser<br>Microsoft Corporation

Brave Private Web Browser<br>Brave Software

Artifacts

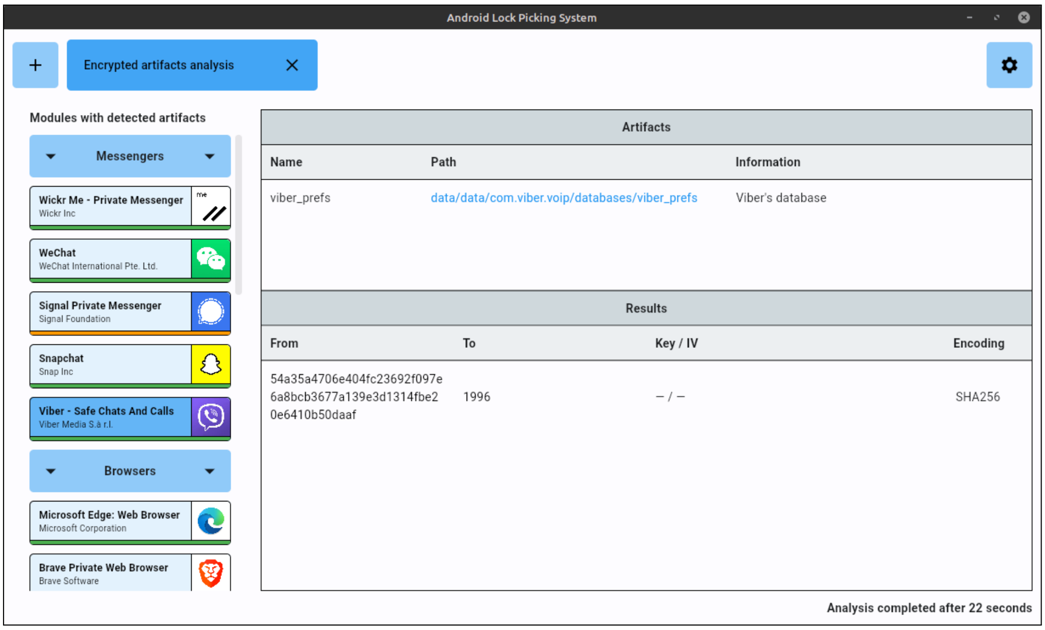
| Name | Path | Information |
|---|---|---|
| memories.db | data/data/com.snapchat.android/databases/memories.db | Snapchat's database |

Results

| From | To | Key / IV | Encoding |
|---|---|---|---|
| $2a$06$TdkG85pbxtMyP/Sp/ZBRn.eDuJMXDqXGmlBmZvf2Ibff1FELfEw66 | 1996 | — / — | bcrypt |

Analysis completed after 22 seconds

Figure C.13: Encrypted artifacts analysis - Snapchat


Android Lock Picking System
+
Encrypted artifacts analysis
Modules with detected artifacts
Messengers
Wickr Me - Private Messenger
Wickr Inc
WeChat
WeChat International Pte. Ltd.
Signal Private Messenger
Signal Foundation
Snapchat
Snap Inc
Viber - Safe Chats And Calls
Viber Media S.à r.l.
Browsers
Microsoft Edge: Web Browser
Microsoft Corporation
Brave Private Web Browser
Brave Software

Artifacts

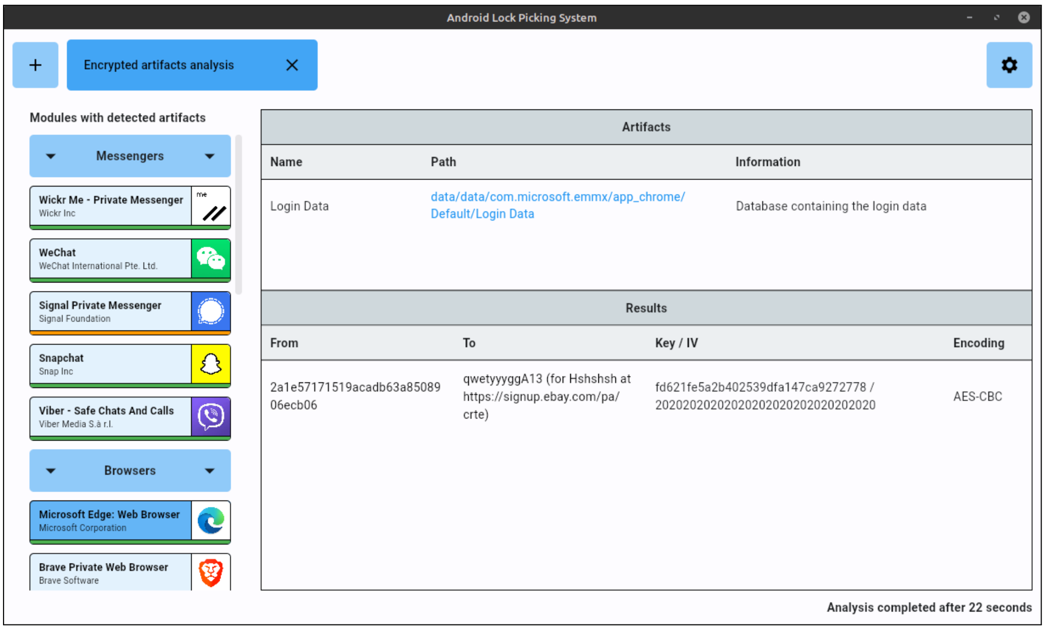
| Name | Path | Information |
|---|---|---|
| viber_prefs | data/data/com.viber.voip/databases/viber_prefs | Viber's database |

Results

| From | To | Key / IV | Encoding |
|---|---|---|---|
| 54a35a4706e404fc23692f097e6a8bcb3677a139e3d1314fbe20e6410b50daaf | 1996 | – / – | SHA256 |

Analysis completed after 22 seconds

Figure C.14: Encrypted artifacts analysis - Viber


Android Lock Picking System
+
Encrypted artifacts analysis
Modules with detected artifacts
Messengers
Wickr Me - Private Messenger
Wickr Inc
WeChat
WeChat International Pte. Ltd.
Signal Private Messenger
Signal Foundation
Snapchat
Snap Inc
Viber - Safe Chats And Calls
Viber Media S.à r.l.
Browsers
Microsoft Edge: Web Browser
Microsoft Corporation
Brave Private Web Browser
Brave Software

Artifacts

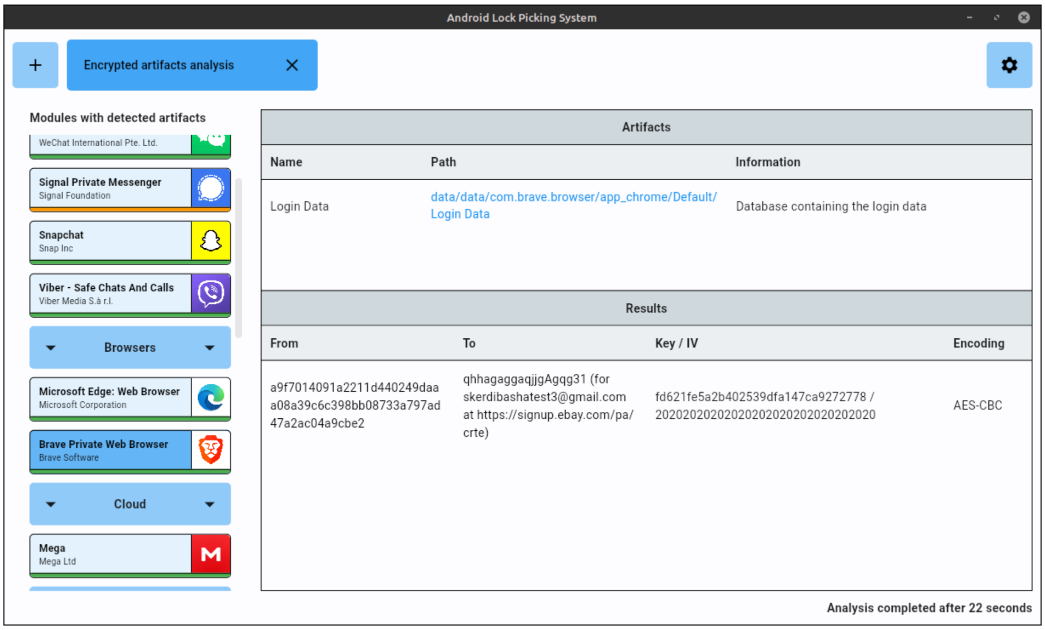
| Name | Path | Information |
|---|---|---|
| Login Data | data/data/com.microsoft.emmx/app_chrome/Default/Login Data | Database containing the login data |

Results

| From | To | Key / IV | Encoding |
|---|---|---|---|
| 2a1e57171519acadb63a8508906ecb06 | qwetyyyggA13 (for Hshshsh at https://signup.ebay.com/pa/crte) | fd621fe5a2b402539dfa147ca9272778 / 20202020202020202020202020202020 | AES-CBC |

Analysis completed after 22 seconds

Figure C.15: Encrypted artifacts analysis - Microsoft Edge


Android Lock Picking System
+
Encrypted artifacts analysis
Modules with detected artifacts
WeChat International Pte. Ltd.
Signal Private Messenger
Signal Foundation
Snapchat
Snap Inc
Viber - Safe Chats And Calls
Viber Media S.à r.l.
Browsers
Microsoft Edge: Web Browser
Microsoft Corporation
Brave Private Web Browser
Brave Software
Cloud
Mega
Mega Ltd

Artifacts

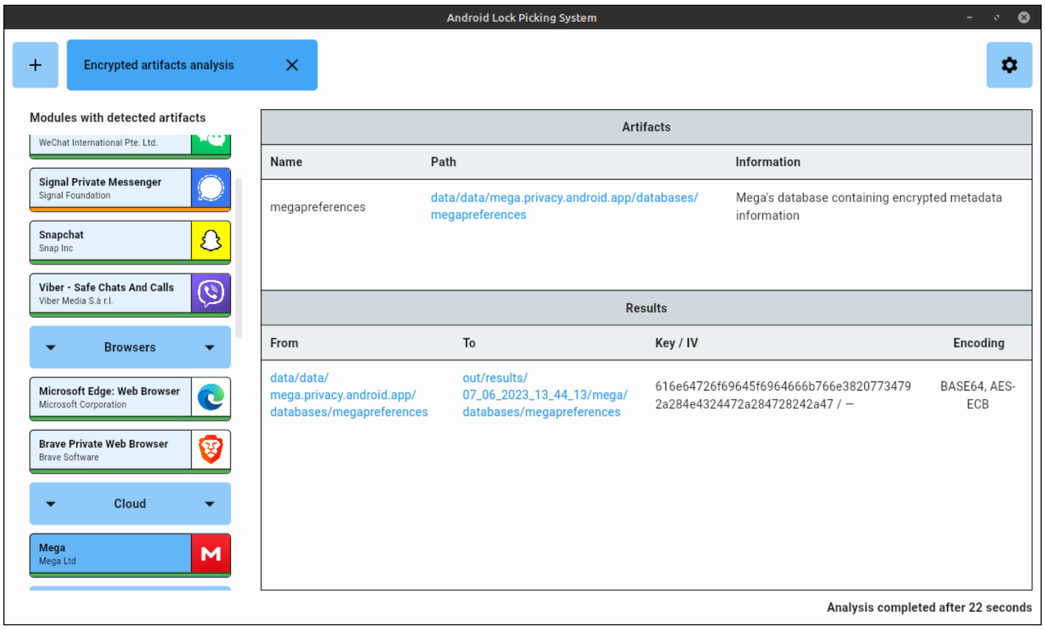
| Name | Path | Information |
|---|---|---|
| Login Data | data/data/com.brave.browser/app_chrome/Default/Login Data | Database containing the login data |

Results

| From | To | Key / IV | Encoding |
|---|---|---|---|
| a9f7014091a2211d440249daaa08a39c6c398bb08733a797ad47a2ac04a9cbe2 | qhhagaggaqjjgAgqg31 (for skerdibashatest3@gmail.com at https://signup.ebay.com/pa/crte) | fd621fe5a2b402539dfa147ca9272778 / 20202020202020202020202020202020 | AES-CBC |

Analysis completed after 22 seconds

Figure C.16: Encrypted artifacts analysis - Brave


Android Lock Picking System
+
Encrypted artifacts analysis
Modules with detected artifacts
WeChat International Pte. Ltd.
Signal Private Messenger
Signal Foundation
Snapchat
Snap Inc
Viber - Safe Chats And Calls
Viber Media S.à r.l.
Browsers
Microsoft Edge: Web Browser
Microsoft Corporation
Brave Private Web Browser
Brave Software
Cloud
Mega
Mega Ltd

Artifacts

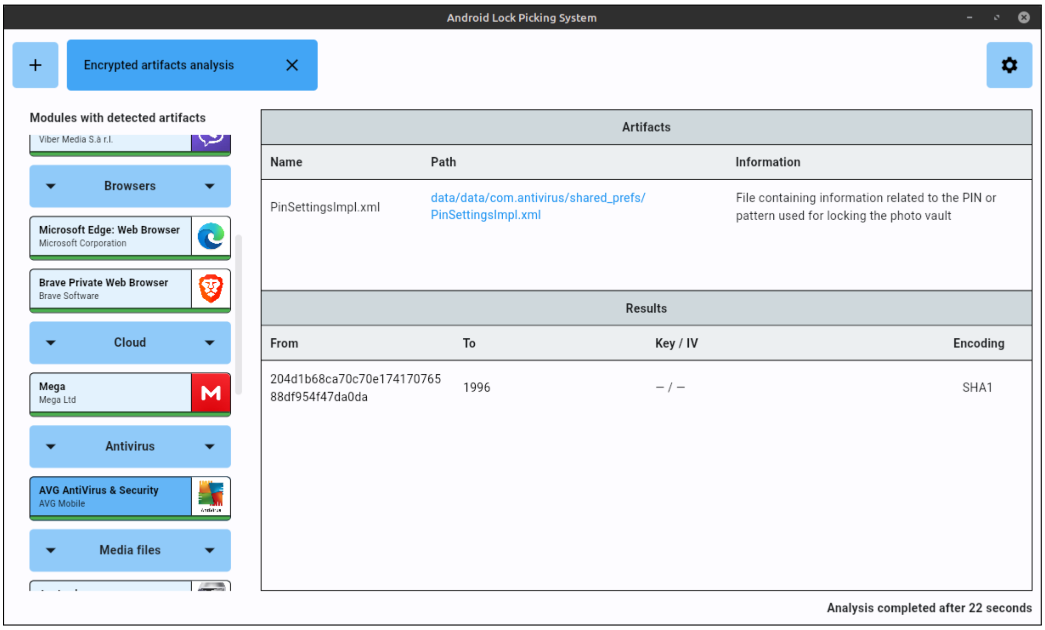
| Name | Path | Information |
|---|---|---|
| megapreferences | data/data/mega.privacy.android.app/databases/megapreferences | Mega's database containing encrypted metadata information |

Results

| From | To | Key / IV | Encoding |
|---|---|---|---|
| data/data/mega.privacy.android.app/databases/megapreferences | out/results/07_06_2023_13_44_13/mega/databases/megapreferences | 616e64726f69645f6964666b766e38207734792a284e4324472a284728242a47 / — | BASE64, AES-ECB |

Analysis completed after 22 seconds

Figure C.17: Encrypted artifacts analysis - Mega


Android Lock Picking System
+
Encrypted artifacts analysis
Modules with detected artifacts
Viber Media S.à r.l.
Browsers
Microsoft Edge: Web Browser
Microsoft Corporation
Brave Private Web Browser
Brave Software
Cloud
Mega
Mega Ltd
Antivirus
AVG AntiVirus & Security
AVG Mobile
Media files

Artifacts

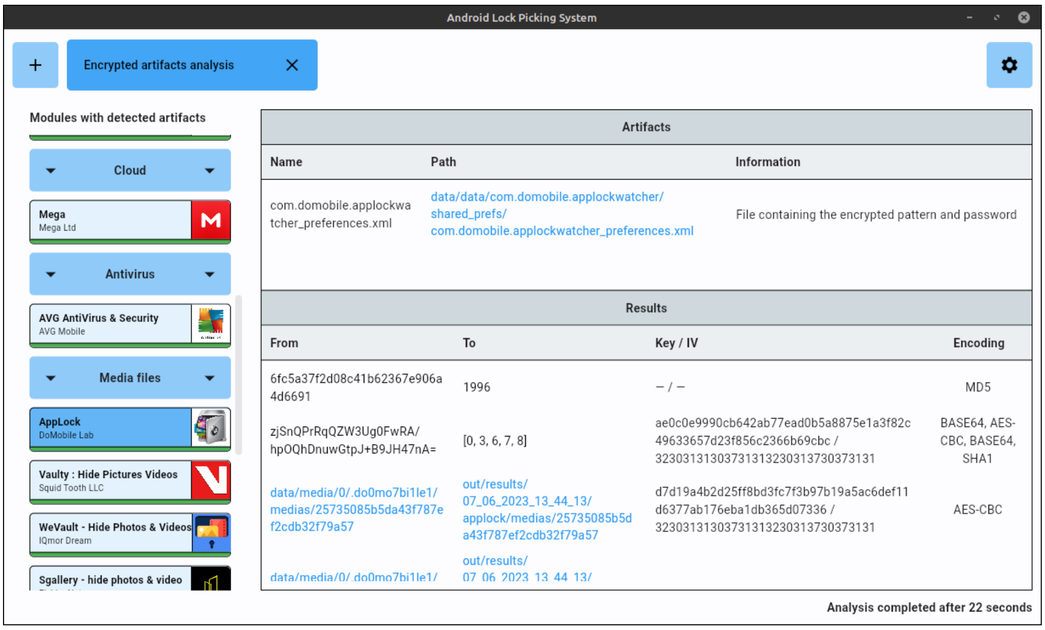
| Name | Path | Information |
|---|---|---|
| PinSettingsImpl.xml | data/data/com.antivirus/shared_prefs/PinSettingsImpl.xml | File containing information related to the PIN or pattern used for locking the photo vault |

Results

| From | To | Key / IV | Encoding |
|---|---|---|---|
| 204d1b68ca70c70e17417076588df954f47da0da | 1996 | – / – | SHA1 |

Analysis completed after 22 seconds

Figure C.18: Encrypted artifacts analysis - AVG


Android Lock Picking System
+
Encrypted artifacts analysis
Modules with detected artifacts
Cloud
Mega
Mega Ltd
Antivirus
AVG AntiVirus & Security
AVG Mobile
Media files
AppLock
DoMobile Lab
Vaulty : Hide Pictures Videos
Squid Tooth LLC
WeVault - Hide Photos & Videos
IQmor Dream
Sgallery - hide photos & video

Artifacts

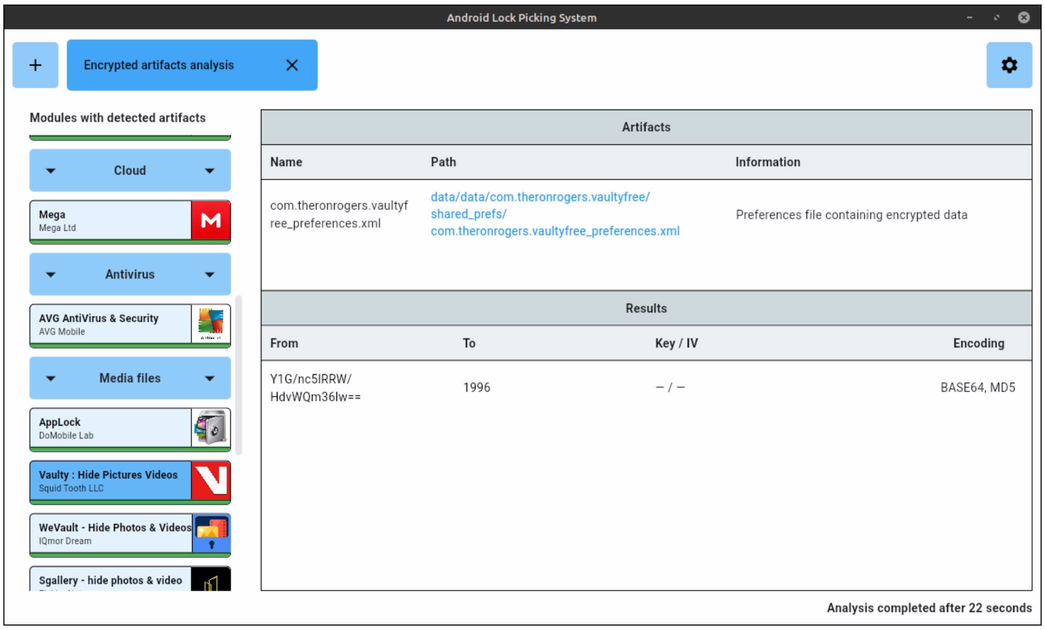
| Name | Path | Information |
|---|---|---|
| com.domobile.applockwatcher_preferences.xml | data/data/com.domobile.applockwatcher/shared_prefs/com.domobile.applockwatcher_preferences.xml | File containing the encrypted pattern and password |

Results

| From | To | Key / IV | Encoding |
|---|---|---|---|
| 6fc5a37f2d08c41b62367e906a4d6691 | 1996 | – / – | MD5 |
| zjSnQPrRqQZW3Ug0FwRA/hpOQhDnuwGtpJ+B9JH47nA= | [0, 3, 6, 7, 8] | ae0c0e9990cb642ab77ead0b5a8875e1a3f82c49633657d23f856c2366b69cbc / 32303131303731313230313730373131 | BASE64, AES-CBC, BASE64, SHA1 |
| data/media/0/.do0mo7bi1le1/medias/25735085b5da43f787ef2cdb32f79a57 | out/results/07_06_2023_13_44_13/applock/medias/25735085b5da43f787ef2cdb32f79a57 | d7d19a4b2d25ff8bd3fc7f3b97b19a5ac6def11d6377ab176eba1db365d07336 / 32303131303731313230313730373131 | AES-CBC |
| data/media/0/.do0mo7bi1le1/ | out/results/07_06_2023_13_44_13/ | | |

Analysis completed after 22 seconds

Figure C.19: Encrypted artifacts analysis - AppLock


Android Lock Picking System
+
Encrypted artifacts analysis
Modules with detected artifacts
Cloud
Mega
Mega Ltd
Antivirus
AVG AntiVirus & Security
AVG Mobile
Media files
AppLock
DoMobile Lab
Vaulty : Hide Pictures Videos
Squid Tooth LLC
WeVault - Hide Photos & Videos
IQmor Dream
Sgallery - hide photos & video

Artifacts

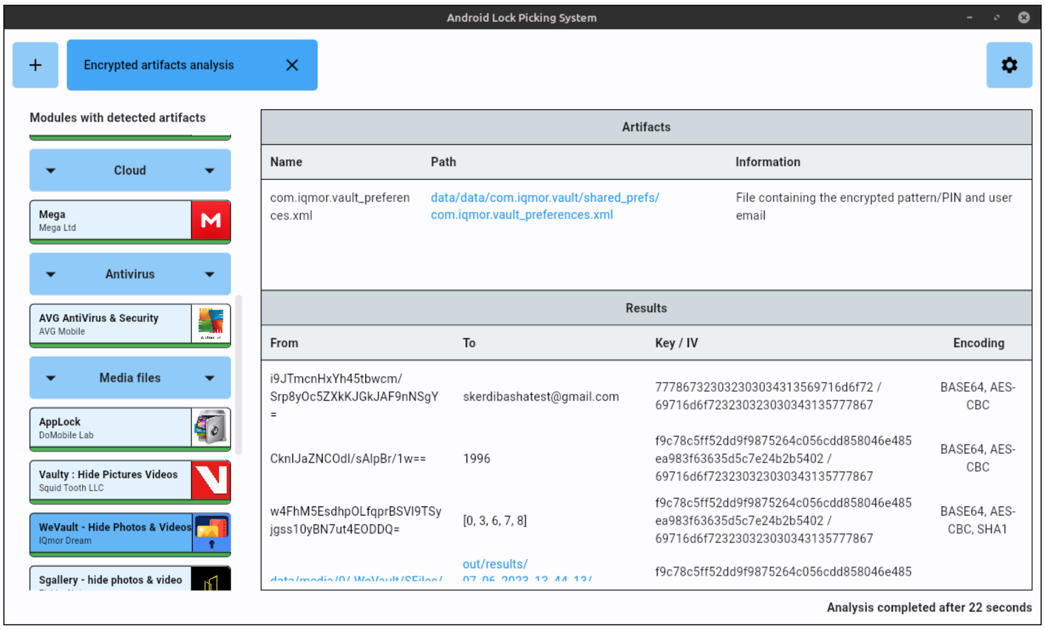
| Name | Path | Information |
|---|---|---|
| com.theronrogers.vaultyfree_preferences.xml | data/data/com.theronrogers.vaultyfree/shared_prefs/com.theronrogers.vaultyfree_preferences.xml | Preferences file containing encrypted data |

Results

| From | To | Key / IV | Encoding |
|---|---|---|---|
| Y1G/nc5IRRW/HdvWQm36lw== | 1996 | – / – | BASE64, MD5 |

Analysis completed after 22 seconds

Figure C.20: Encrypted artifacts analysis - Vaulty


Android Lock Picking System
+
Encrypted artifacts analysis
Modules with detected artifacts
Cloud
Mega
Mega Ltd
Antivirus
AVG AntiVirus & Security
AVG Mobile
Media files
AppLock
DoMobile Lab
Vaulty : Hide Pictures Videos
Squid Tooth LLC
WeVault - Hide Photos & Videos
IQmor Dream
Sgallery - hide photos & video

Artifacts

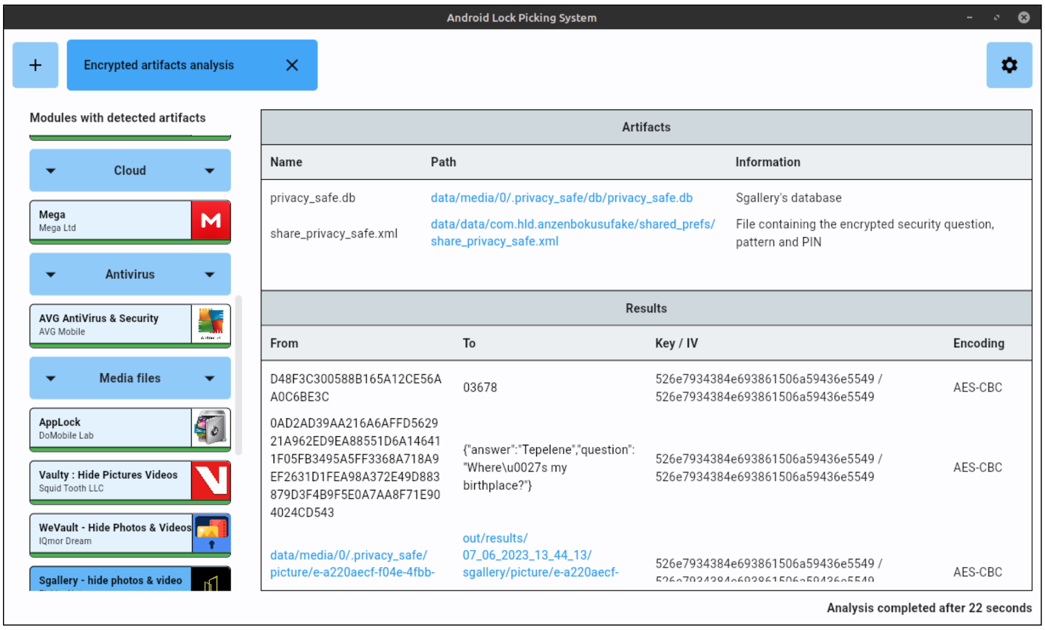
| Name | Path | Information |
|---|---|---|
| com.iqmor.vault_preferences.xml | data/data/com.iqmor.vault/shared_prefs/com.iqmor.vault_preferences.xml | File containing the encrypted pattern/PIN and user email |

Results

| From | To | Key / IV | Encoding |
|---|---|---|---|
| i9JTmcnHxYh45tbwcm/Srp8yOc5ZXkKJGkJAF9nNSgY= | skerdibashatest@gmail.com | 77786732303230303431356971 6d6f72 / 69716d6f72323032303034313577786 7 | BASE64, AES-CBC |
| CknIJaZNCOdl/sAlpBr/1w== | 1996 | f9c78c5ff52dd9f9875264c056cdd858046e485ea983f63635d5c7e24b2b5402 / 69716d6f7232303230303431357778 67 | BASE64, AES-CBC |
| w4FhM5EsdhpOLfqprBSVI9TSyjgss10yBN7ut4EODDQ= | [0, 3, 6, 7, 8] | f9c78c5ff52dd9f9875264c056cdd858046e485ea983f63635d5c7e24b2b5402 / 69716d6f7232303230303431357778 67 | BASE64, AES-CBC, SHA1 |
| data/media/0/.WeVault/SFiles/ | out/results/07_06_2023_13_44_13/ | f9c78c5ff52dd9f9875264c056cdd858046e485 | |

Analysis completed after 22 seconds

Figure C.21: Encrypted artifacts analysis - WeVault


Android Lock Picking System
+
Encrypted artifacts analysis
Modules with detected artifacts
Cloud
Mega
Mega Ltd
Antivirus
AVG AntiVirus & Security
AVG Mobile
Media files
AppLock
DoMobile Lab
Vaulty : Hide Pictures Videos
Squid Tooth LLC
WeVault - Hide Photos & Videos
IQmor Dream
Sgallery - hide photos & video

Artifacts

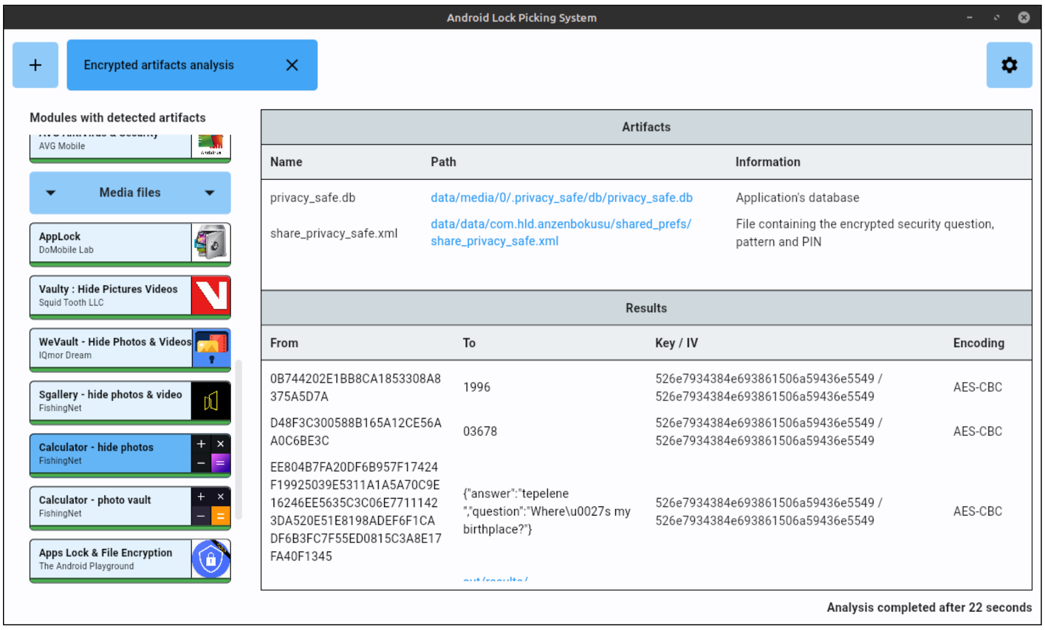
| Name | Path | Information |
|---|---|---|
| privacy_safe.db | data/media/0/.privacy_safe/db/privacy_safe.db | Sgallery's database |
| share_privacy_safe.xml | data/data/com.hld.anzenbokusufake/shared_prefs/share_privacy_safe.xml | File containing the encrypted security question, pattern and PIN |

Results

| From | To | Key / IV | Encoding |
|---|---|---|---|
| D48F3C300588B165A12CE56AA0C6BE3C | 03678 | 526e7934384e693861506a59436e5549 / 526e7934384e693861506a59436e5549 | AES-CBC |
| 0AD2AD39AA216A6AFFD562921A962ED9EA88551D6A146411F05FB3495A5FF3368A718A9EF2631D1FEA98A372E49D883879D3F4B9F5E0A7AA8F71E904024CD543 | {"answer":"Tepelene","question":"Where\u0027s my birthplace?"} | 526e7934384e693861506a59436e5549 / 526e7934384e693861506a59436e5549 | AES-CBC |
| data/media/0/.privacy_safe/picture/e-a220aecf-f04e-4fbb- | out/results/07_06_2023_13_44_13/sgallery/picture/e-a220aecf- | 526e7934384e693861506a59436e5549 / | AES-CBC |

Analysis completed after 22 seconds

Figure C.22: Encrypted artifacts analysis - Sgallery


Android Lock Picking System
+
Encrypted artifacts analysis
Modules with detected artifacts
AVG Mobile
Media files
AppLock
DoMobile Lab
Vaulty : Hide Pictures Videos
Squid Tooth LLC
WeVault - Hide Photos & Videos
IQmor Dream
Sgallery - hide photos & video
FishingNet
Calculator - hide photos
FishingNet
Calculator - photo vault
FishingNet
Apps Lock & File Encryption
The Android Playground

Artifacts

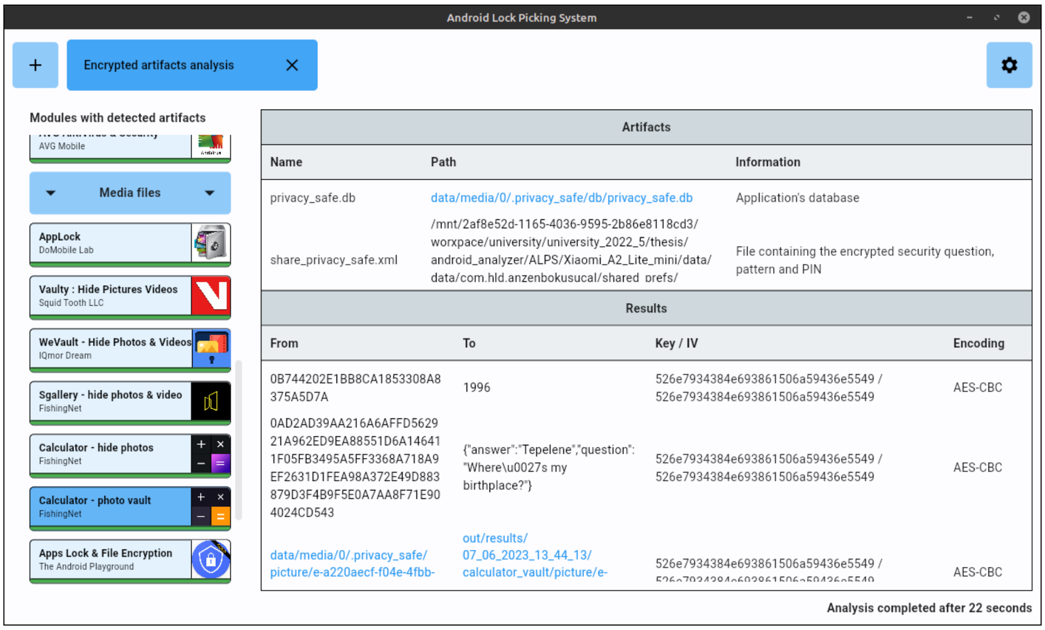
| Name | Path | Information |
|---|---|---|
| privacy_safe.db | data/media/0/.privacy_safe/db/privacy_safe.db | Application's database |
| share_privacy_safe.xml | data/data/com.hld.anzenbokusu/shared_prefs/share_privacy_safe.xml | File containing the encrypted security question, pattern and PIN |

Results

| From | To | Key / IV | Encoding |
|---|---|---|---|
| 0B744202E1BB8CA1853308A8375A5D7A | 1996 | 526e7934384e693861506a59436e5549 / 526e7934384e693861506a59436e5549 | AES-CBC |
| D48F3C300588B165A12CE56AA0C6BE3C | 03678 | 526e7934384e693861506a59436e5549 / 526e7934384e693861506a59436e5549 | AES-CBC |
| EE804B7FA20DF6B957F17424F19925039E5311A1A5A70C9E16246EE5635C3C06E77111423DA520E51E8198ADEF6F1CADF6B3FC7F55ED0815C3A8E17FA40F1345 | {"answer":"tepelene ","question":"Where\u0027s my birthplace?"} | 526e7934384e693861506a59436e5549 / 526e7934384e693861506a59436e5549 | AES-CBC |

Analysis completed after 22 seconds

Figure C.23: Encrypted artifacts analysis - Calculator hide photos


Android Lock Picking System
+
Encrypted artifacts analysis
Modules with detected artifacts
AVG Mobile
Media files
AppLock
DoMobile Lab
Vaulty : Hide Pictures Videos
Squid Tooth LLC
WeVault - Hide Photos & Videos
IQmor Dream
Sgallery - hide photos & video
FishingNet
Calculator - hide photos
FishingNet
Calculator - photo vault
FishingNet
Apps Lock & File Encryption
The Android Playground

Artifacts

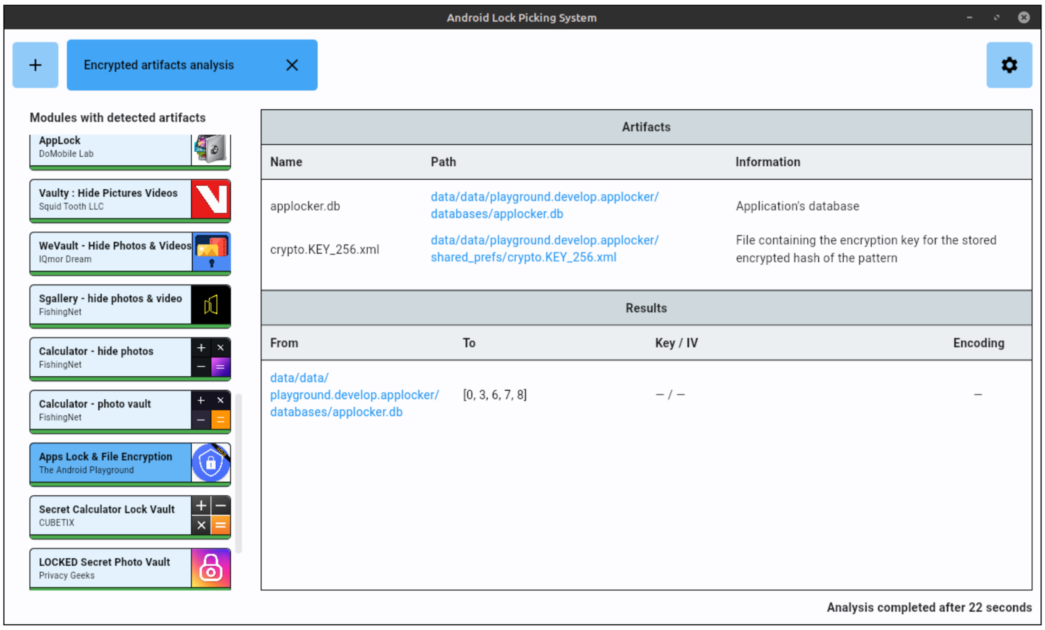
| Name | Path | Information |
|---|---|---|
| privacy_safe.db | data/media/0/.privacy_safe/db/privacy_safe.db | Application's database |
| share_privacy_safe.xml | /mnt/2af8e52d-1165-4036-9595-2b86e8118cd3/worxpace/university/university_2022_5/thesis/android_analyzer/ALPS/Xiaomi_A2_Lite_mini/data/data/com.hld.anzenbokusucal/shared_prefs/ | File containing the encrypted security question, pattern and PIN |

Results

| From | To | Key / IV | Encoding |
|---|---|---|---|
| 0B744202E1BB8CA1853308A8375A5D7A | 1996 | 526e7934384e693861506a59436e5549 / 526e7934384e693861506a59436e5549 | AES-CBC |
| 0AD2AD39AA216A6AFFD562921A962ED9EA88551D6A146411F05FB3495A5FF3368A718A9EF2631D1FEA98A372E49D883879D3F4B9F5E0A7AA8F71E904024CD543 | {"answer":"Tepelene","question":"Where\u0027s my birthplace?"} | 526e7934384e693861506a59436e5549 / 526e7934384e693861506a59436e5549 | AES-CBC |
| data/media/0/.privacy_safe/picture/e-a220aecf-f04e-4fbb- | out/results/07_06_2023_13_44_13/calculator_vault/picture/e- | 526e7934384e693861506a59436e5549 / | AES-CBC |

Analysis completed after 22 seconds

Figure C.24: Encrypted artifacts analysis - Calculator photo vault


Android Lock Picking System
+
Encrypted artifacts analysis
Modules with detected artifacts
AppLock
DoMobile Lab
Vaulty : Hide Pictures Videos
Squid Tooth LLC
WeVault - Hide Photos & Videos
IQmor Dream
Sgallery - hide photos & video
FishingNet
Calculator - hide photos
FishingNet
Calculator - photo vault
FishingNet
Apps Lock & File Encryption
The Android Playground
Secret Calculator Lock Vault
CUBETIX
LOCKED Secret Photo Vault
Privacy Geeks

Artifacts

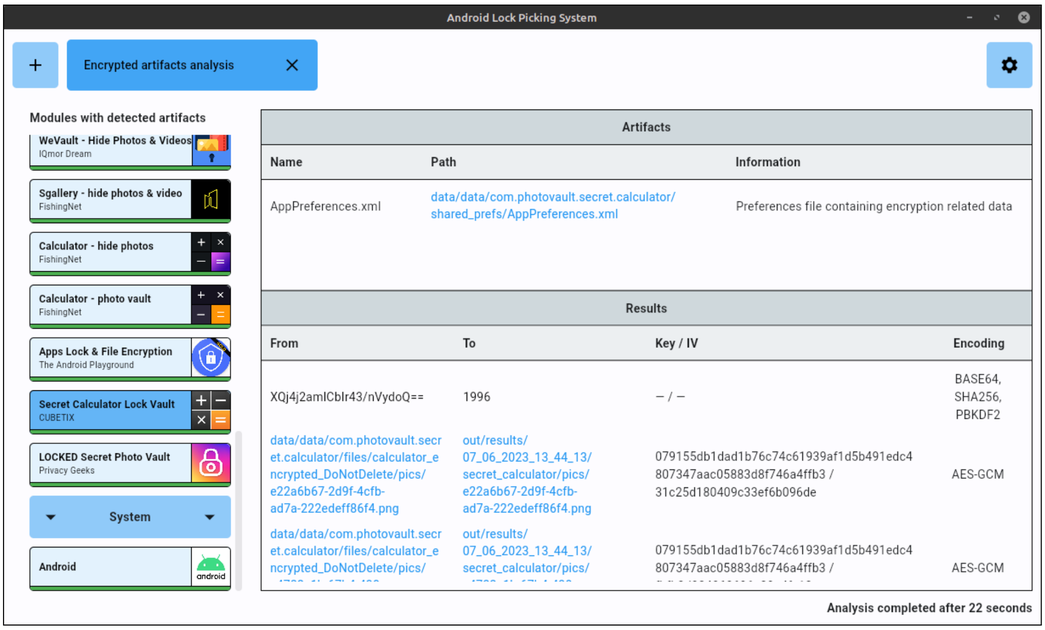
| Name | Path | Information |
|---|---|---|
| applocker.db | data/data/playground.develop.applocker/databases/applocker.db | Application's database |
| crypto.KEY_256.xml | data/data/playground.develop.applocker/shared_prefs/crypto.KEY_256.xml | File containing the encryption key for the stored encrypted hash of the pattern |

Results

| From | To | Key / IV | Encoding |
|---|---|---|---|
| data/data/playground.develop.applocker/databases/applocker.db | [0, 3, 6, 7, 8] | – / – | – |

Analysis completed after 22 seconds

Figure C.25: Encrypted artifacts analysis - Apps Lock & File Encryption


Android Lock Picking System
+
Encrypted artifacts analysis
Modules with detected artifacts
WeVault - Hide Photos & Videos
IQmor Dream
Sgallery - hide photos & video
FishingNet
Calculator - hide photos
FishingNet
Calculator - photo vault
FishingNet
Apps Lock & File Encryption
The Android Playground
Secret Calculator Lock Vault
CUBETIX
LOCKED Secret Photo Vault
Privacy Geeks
System
Android

Artifacts

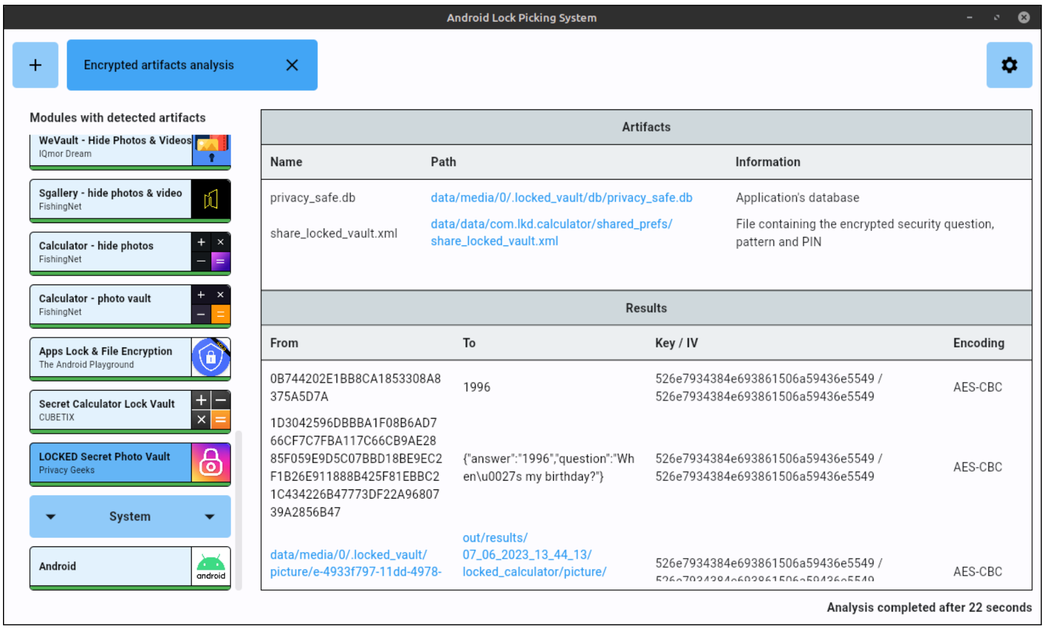
| Name | Path | Information |
|---|---|---|
| AppPreferences.xml | data/data/com.photovault.secret.calculator/shared_prefs/AppPreferences.xml | Preferences file containing encryption related data |

Results

| From | To | Key / IV | Encoding |
|---|---|---|---|
| XQj4j2amlCblr43/nVydoQ== | 1996 | – / – | BASE64,<br>SHA256,<br>PBKDF2 |
| data/data/com.photovault.secret.calculator/files/calculator_encrypted_DoNotDelete/pics/e22a6b67-2d9f-4cfb-ad7a-222edeff86f4.png | out/results/07_06_2023_13_44_13/secret_calculator/pics/e22a6b67-2d9f-4cfb-ad7a-222edeff86f4.png | 079155db1dad1b76c74c61939af1d5b491edc4807347aac05883d8f746a4ffb3 / 31c25d180409c33ef6b096de | AES-GCM |
| data/data/com.photovault.secret.calculator/files/calculator_encrypted_DoNotDelete/pics/ | out/results/07_06_2023_13_44_13/secret_calculator/pics/ | 079155db1dad1b76c74c61939af1d5b491edc4807347aac05883d8f746a4ffb3 / | AES-GCM |

Analysis completed after 22 seconds

Figure C.26: Encrypted artifacts analysis - Secret Calculator Lock Vault


Android Lock Picking System
+
Encrypted artifacts analysis
Modules with detected artifacts
WeVault - Hide Photos & Videos
IQmor Dream
Sgallery - hide photos & video
FishingNet
Calculator - hide photos
FishingNet
Calculator - photo vault
FishingNet
Apps Lock & File Encryption
The Android Playground
Secret Calculator Lock Vault
CUBETIX
LOCKED Secret Photo Vault
Privacy Geeks
System
Android

Artifacts

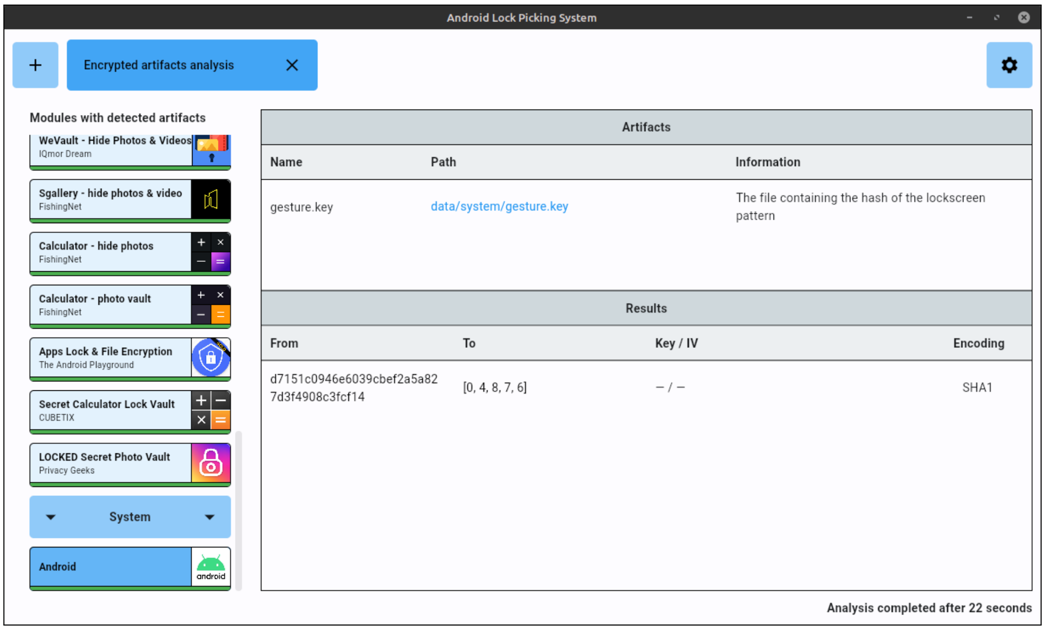
| Name | Path | Information |
|---|---|---|
| privacy_safe.db | data/media/0/.locked_vault/db/privacy_safe.db | Application's database |
| share_locked_vault.xml | data/data/com.lkd.calculator/shared_prefs/share_locked_vault.xml | File containing the encrypted security question, pattern and PIN |

Results

| From | To | Key / IV | Encoding |
|---|---|---|---|
| 0B744202E1BB8CA1853308A8375A5D7A | 1996 | 526e7934384e693861506a59436e5549 / 526e7934384e693861506a59436e5549 | AES-CBC |
| 1D3042596DBBBA1F08B6AD766CF7C7FBA117C66CB9AE2885F059E9D5C07BBD18BE9EC2F1B26E911888B425F81EBBC21C434226B47773DF22A9680739A2856B47 | {"answer":"1996","question":"When\u0027s my birthday?"} | 526e7934384e693861506a59436e5549 / 526e7934384e693861506a59436e5549 | AES-CBC |
| data/media/0/.locked_vault/picture/e-4933f797-11dd-4978- | out/results/07_06_2023_13_44_13/locked_calculator/picture/ | 526e7934384e693861506a59436e5549 / | AES-CBC |

Analysis completed after 22 seconds

Figure C.27: Encrypted artifacts analysis - LOCKED Secret Photo Vault


Android Lock Picking System
+
Encrypted artifacts analysis
Modules with detected artifacts
WeVault - Hide Photos & Videos
IQmor Dream
Sgallery - hide photos & video
FishingNet
Calculator - hide photos
FishingNet
Calculator - photo vault
FishingNet
Apps Lock & File Encryption
The Android Playground
Secret Calculator Lock Vault
CUBETIX
LOCKED Secret Photo Vault
Privacy Geeks
System
Android

Artifacts

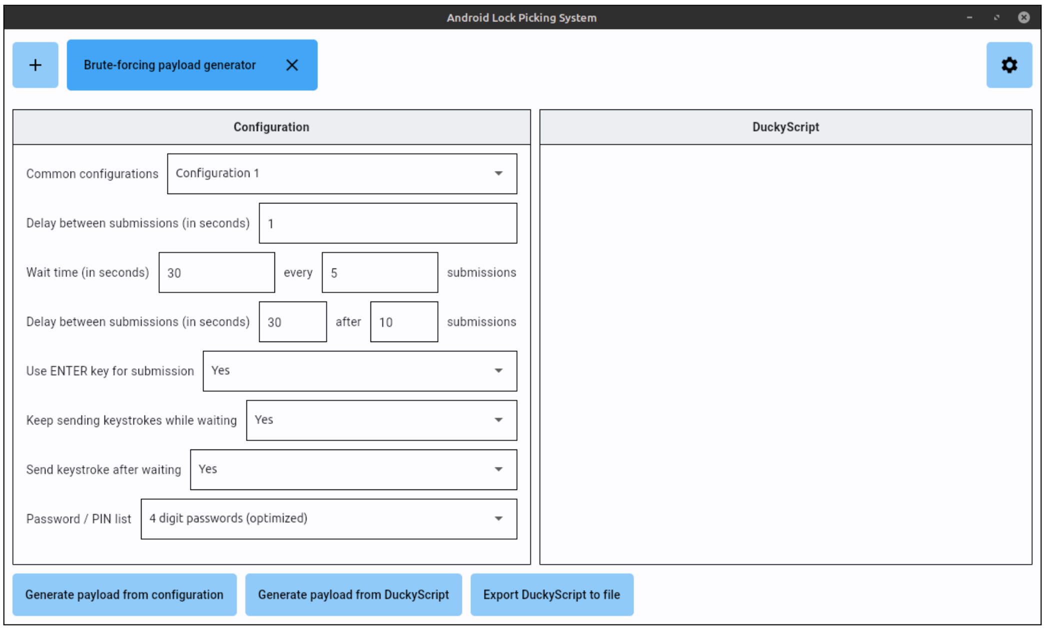
| Name | Path | Information |
|---|---|---|
| gesture.key | data/system/gesture.key | The file containing the hash of the lockscreen pattern |

Results

| From | To | Key / IV | Encoding |
|---|---|---|---|
| d7151c0946e6039cbef2a5a82 7d3f4908c3fcf14 | [0, 4, 8, 7, 6] | – / – | SHA1 |

Analysis completed after 22 seconds

Figure C.28: Encrypted artifacts analysis - Android


Android Lock Picking System
+
Brute-forcing payload generator
Configuration
DuckyScript
Generate payload from configuration
Generate payload from DuckyScript
Export DuckyScript to file

Common configurations: Configuration 1

Delay between submissions (in seconds): 1

Wait time (in seconds): 30 every 5 submissions

Delay between submissions (in seconds): 30 after 10 submissions

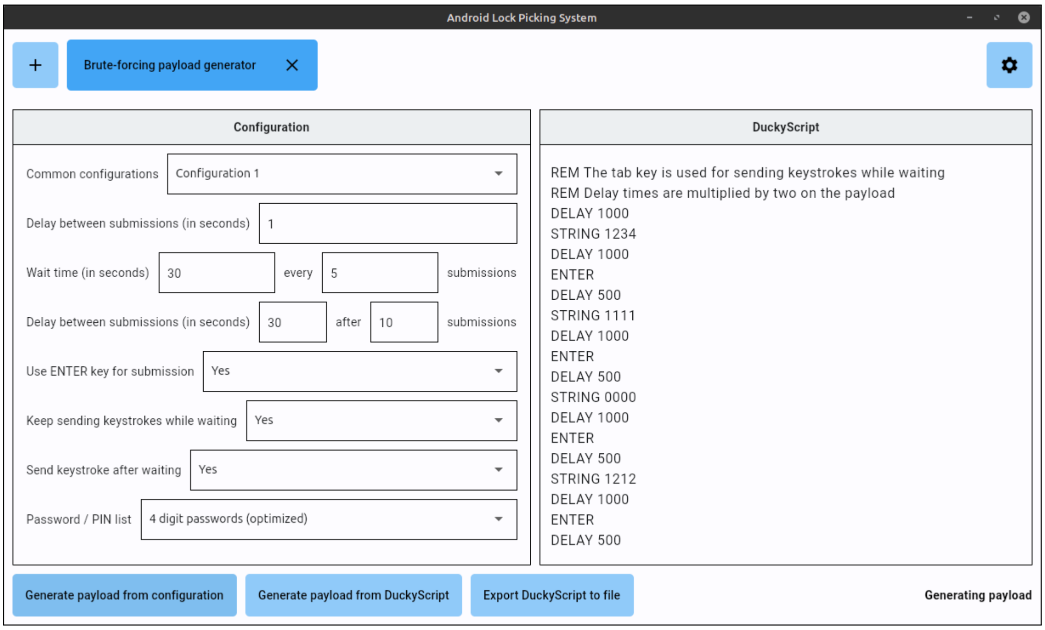
Use ENTER key for submission: Yes

Keep sending keystrokes while waiting: Yes

Send keystroke after waiting: Yes

Password / PIN list: 4 digit passwords (optimized)

Figure C.29: Brute-forcing payload generator


Android Lock Picking System
+
Brute-forcing payload generator
Configuration
DuckyScript
Generate payload from configuration
Generate payload from DuckyScript
Export DuckyScript to file

Common configurations: Configuration 1

Delay between submissions (in seconds): 1

Wait time (in seconds) 30 every 5 submissions

Delay between submissions (in seconds) 30 after 10 submissions

Use ENTER key for submission: Yes

Keep sending keystrokes while waiting: Yes

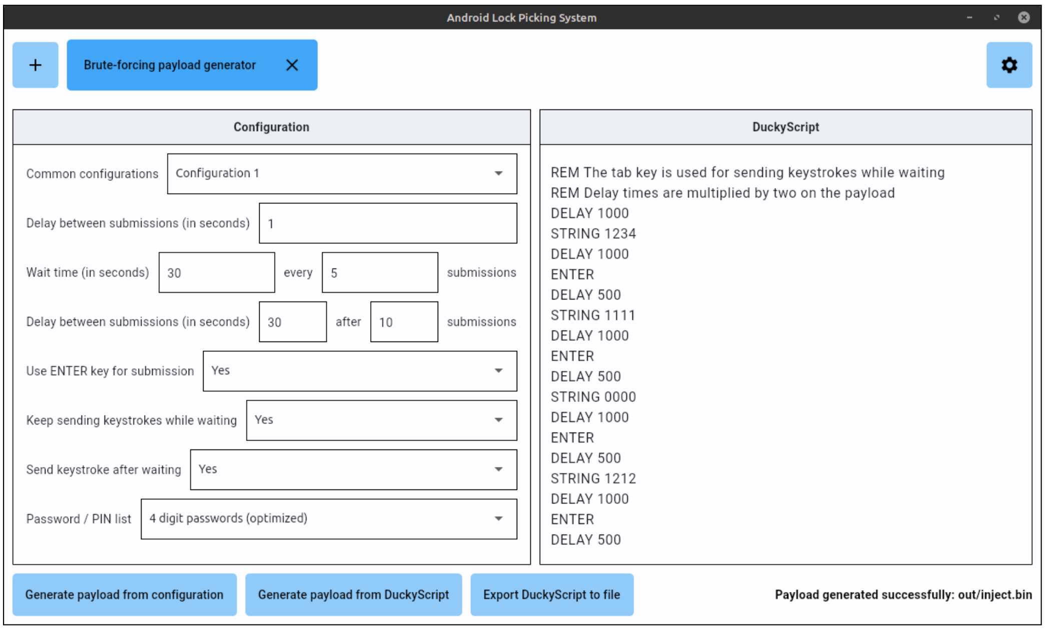
Send keystroke after waiting: Yes

Password / PIN list: 4 digit passwords (optimized)

```
REM The tab key is used for sending keystrokes while waiting
REM Delay times are multiplied by two on the payload
DELAY 1000
STRING 1234
DELAY 1000
ENTER
DELAY 500
STRING 1111
DELAY 1000
ENTER
DELAY 500
STRING 0000
DELAY 1000
ENTER
DELAY 500
STRING 1212
DELAY 1000
ENTER
DELAY 500
```

Generating payload

Figure C.30: Brute-forcing payload generator - Generating payload


Android Lock Picking System
+
Brute-forcing payload generator
Configuration
DuckyScript
Generate payload from configuration
Generate payload from DuckyScript
Export DuckyScript to file

Common configurations: Configuration 1

Delay between submissions (in seconds): 1

Wait time (in seconds) 30 every 5 submissions

Delay between submissions (in seconds) 30 after 10 submissions

Use ENTER key for submission: Yes

Keep sending keystrokes while waiting: Yes

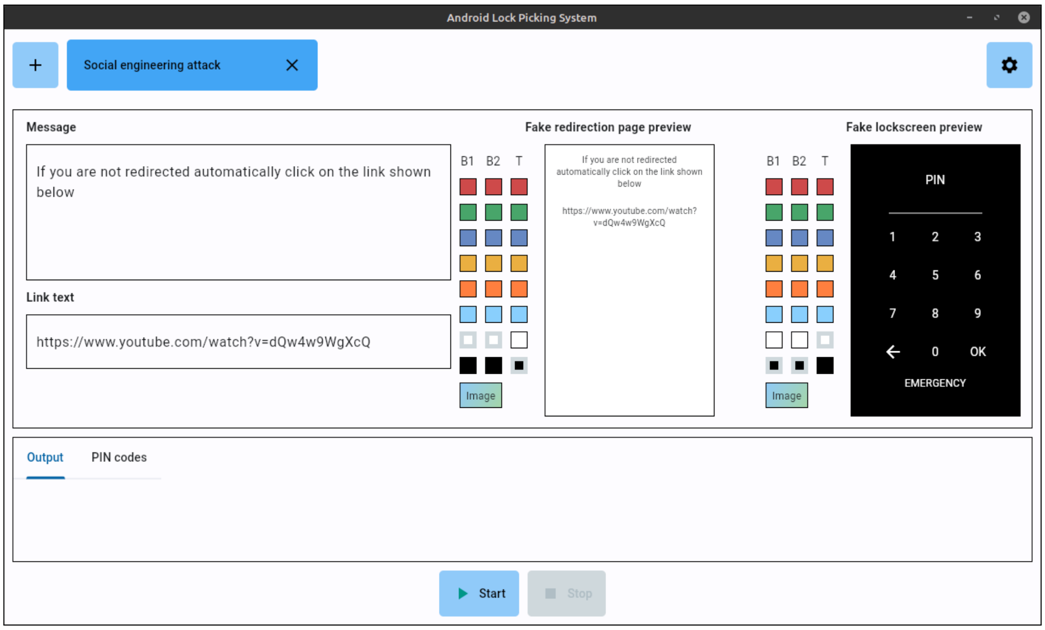
Send keystroke after waiting: Yes

Password / PIN list: 4 digit passwords (optimized)

```
REM The tab key is used for sending keystrokes while waiting
REM Delay times are multiplied by two on the payload
DELAY 1000
STRING 1234
DELAY 1000
ENTER
DELAY 500
STRING 1111
DELAY 1000
ENTER
DELAY 500
STRING 0000
DELAY 1000
ENTER
DELAY 500
STRING 1212
DELAY 1000
ENTER
DELAY 500
```

Payload generated successfully: out/inject.bin

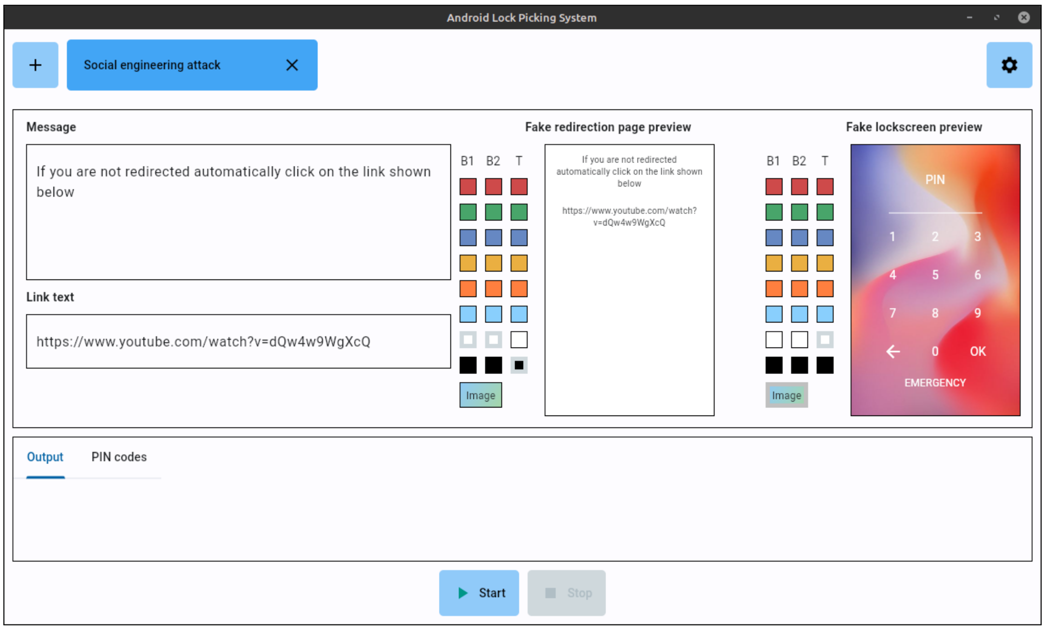
Figure C.31: Brute-forcing payload generator - Payload generated



Message: If you are not redirected automatically click on the link shown below

Link text: https://www.youtube.com/watch?v=dQw4w9WgXcQ

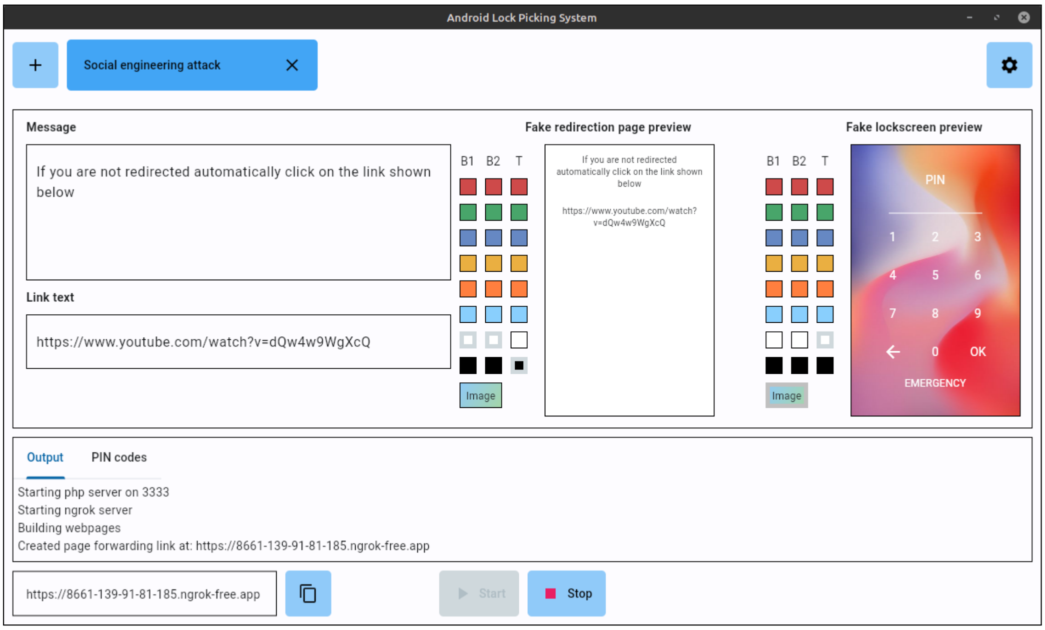
Figure C.32: Social engineering attack



Message: If you are not redirected automatically click on the link shown below

Link text: https://www.youtube.com/watch?v=dQw4w9WgXcQ

Figure C.33: Social engineering attack - Editing fake lockscreen


Android Lock Picking System
+
Social engineering attack
B1 B2 T
Image
If you are not redirected automatically click on the link shown below
https://www.youtube.com/watch?v=dQw4w9WgXcQ
PIN
1 2 3
4 5 6
7 8 9
0 OK
EMERGENCY
Output
PIN codes
Start
Stop

Message

If you are not redirected automatically click on the link shown below

Link text

https://www.youtube.com/watch?v=dQw4w9WgXcQ

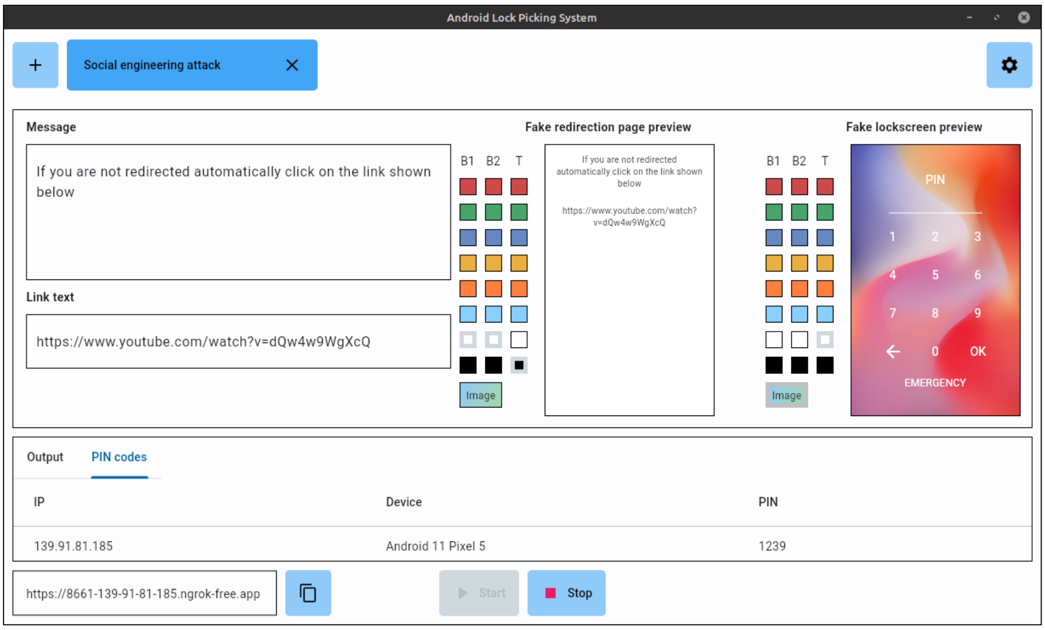
Fake redirection page preview

Fake lockscreen preview

```
Starting php server on 3333
Starting ngrok server
Building webpages
Created page forwarding link at: https://8661-139-91-81-185.ngrok-free.app
```

https://8661-139-91-81-185.ngrok-free.app

Figure C.34: Social engineering attack - Created link


Android Lock Picking System
+
Social engineering attack
B1 B2 T
Image
If you are not redirected automatically click on the link shown below
https://www.youtube.com/watch?v=dQw4w9WgXcQ
PIN
1 2 3
4 5 6
7 8 9
0 OK
EMERGENCY
Output
PIN codes
Start
Stop

Message

If you are not redirected automatically click on the link shown below

Link text

https://www.youtube.com/watch?v=dQw4w9WgXcQ

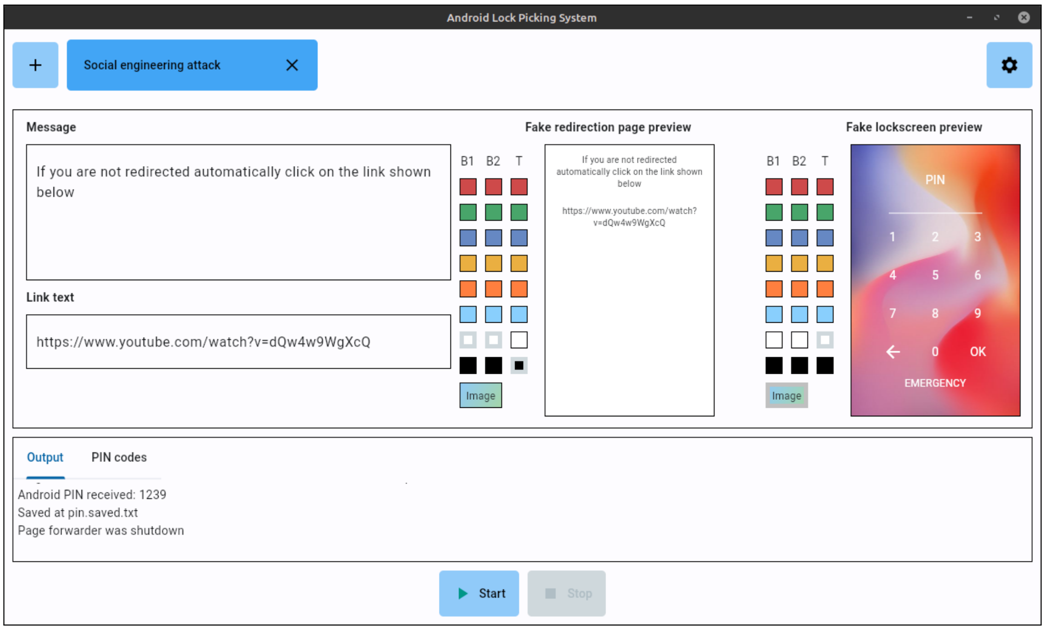
Fake redirection page preview

Fake lockscreen preview

| IP | Device | PIN |
|---|---|---|
| 139.91.81.185 | Android 11 Pixel 5 | 1239 |

https://8661-139-91-81-185.ngrok-free.app

Figure C.35: Social engineering attack - Received inputs



Message

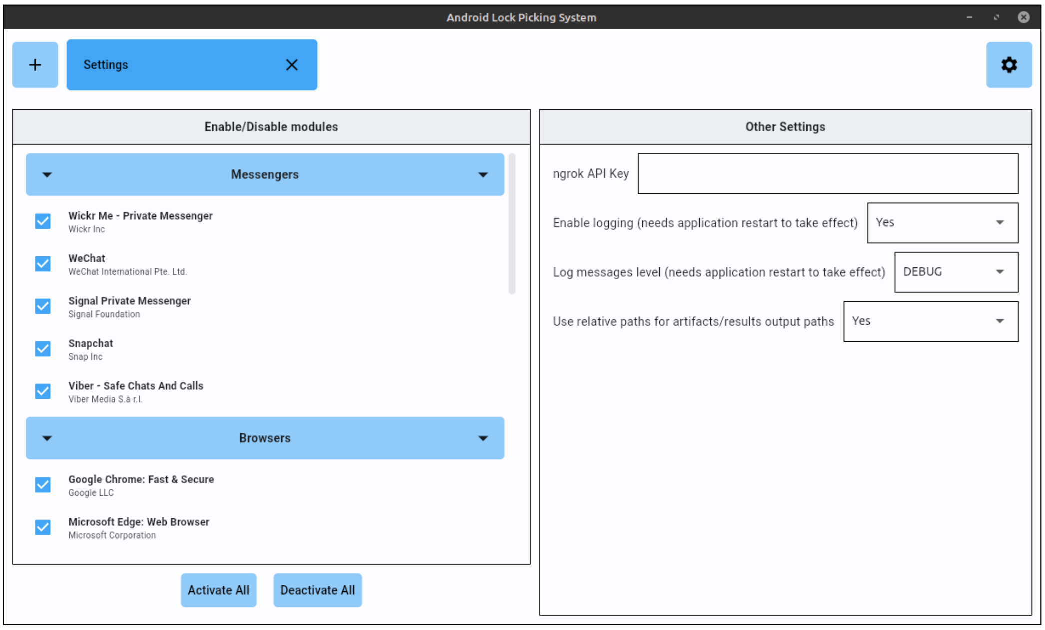
If you are not redirected automatically click on the link shown below

Link text

https://www.youtube.com/watch?v=dQw4w9WgXcQ

```
Android PIN received: 1239
Saved at pin.saved.txt
Page forwarder was shutdown
```

Figure C.36: Social engineering attack - Page forwarder shutdown



Messengers

- ☑ Wickr Me - Private Messenger — Wickr Inc
- ☑ WeChat — WeChat International Pte. Ltd.
- ☑ Signal Private Messenger — Signal Foundation
- ☑ Snapchat — Snap Inc
- ☑ Viber - Safe Chats And Calls — Viber Media S.à r.l.

Browsers

- ☑ Google Chrome: Fast & Secure — Google LLC
- ☑ Microsoft Edge: Web Browser — Microsoft Corporation

ngrok API Key:

Enable logging (needs application restart to take effect): Yes

Log messages level (needs application restart to take effect): DEBUG

Use relative paths for artifacts/results output paths: Yes

Figure C.37: Settings

# Bibliography

[1] NDTV Gadgets 360. Android 5.0 Lollipop Brings Full SD Card Access for Third-Party Apps. https://www.gadgets360.com/mobiles/news/android-50-lollipop-brings-full-sd-card-access-for-third-party-apps-616420, 2023. [Accessed 1-Jun-2023].

[2] Open Handset Alliance. Open Handset Alliance. https://www.openhandsetalliance.com, 2023. [Accessed 1-Jun-2023].

[3] Android. Authentication. https://source.android.com/docs/security/features/authentication, 2023. [Accessed 10-Jan-2023].

[4] Android. File Based Encryption. https://source.android.com/docs/security/features/encryption/file-based, 2023. [Accessed 10-Jan-2023].

[5] Android. Full Disk Encryption. https://source.android.com/docs/security/features/encryption/full-disk, 2023. [Accessed 10-Jan-2023].

[6] Android. Gatekeeper. https://source.android.com/docs/security/features/authentication/gatekeeper, 2023. [Accessed 14-May-2023].

[7] Jeff Atwood. Dictionary Attacks 101. https://blog.codinghorror.com/dictionary-attacks-101/, 2009. [Accessed 8-Jun-2023].

[8] BasisTech. Autopsy - Digital Forensics. https://www.autopsy.com/, 2023. [Accessed 1-Jun-2023].

[9] Belkasoft. Belkasoft X. https://belkasoft.com/x, 2023. [Accessed 1-Jun-2023].

[10] Alexis Brignoni. Android Logs Events And Protobuf Parser. https://github.com/abrignoni/ALEAPP, 2023. [Accessed 1-Jun-2023].

[11] Businessofapps. Android Statistics. https://www.businessofapps.com/data/android-statistics/, 2023. [Accessed 1-Jun-2023].

[12] Cellebrite. Cellebrite UFED. https://cellebrite.com/en/ufed/, 2023. [Accessed 1-Jun-2023].

[13] Hung-Yu Chien. Dynamic Public Key Certificates with Forward Secrecy. https://www.mdpi.com/2079-9292/10/16/2009, 2009. [Accessed 5-Jun-2023].

[14] The Linux Choice. Lockphish. https://github.com/jaykali/lockphish, 2023. [Accessed 10-Jan-2023].

[15] Cloudflare. What is a phishing attack? https://www.cloudflare.com/en-gb/learning/access-management/phishing-attack/, 2023. [Accessed 8-Jun-2023].

[16] Microsoft Corporation. Microsoft Edge: Web Browser. https://play.google.com/store/apps/details?id=com.microsoft.emmx&hl=en&gl=US, 2023. [Accessed 20-Apr-2023].

[17] CUBETIX. Secret Calculator Lock Vault. https://play.google.com/store/apps/details?id=com.photovault.secret.calculator&hl=en&gl=US, 2023. [Accessed 20-Apr-2023].

[18] IQmor Dream. WeVault - Hide Photos & Videos. https://play.google.com/store/apps/details?id=com.iqmor.vault&hl=en&gl=US, 2023. [Accessed 20-Apr-2023].

[19] Wladyslaw Kozaczuk. Edited and translated by Christopher Kasparek. *Enigma: How the German Machine Cipher Was Broken, and How It Was Read by the Allies in World War Two.* University Publications of America, 1984.

[20] Elcomsoft. Elcomsoft Mobile Forensic Bundle. https://www.elcomsoft.com/emfb.html, 2023. [Accessed 1-Jun-2023].

[21] Europol. Third report of the observatory function on encryption. https://www.europol.europa.eu/cms/sites/default/files/documents/3rd_report_of_the_observatory_function_on_encryption-web.pdf, 2023. [Accessed 1-Jun-2023].

[22] Fernico. ZRT3 - fernico. https://fernico.us/zrt3/, 2023. [Accessed 1-Jun-2023].

[23] FishingNet. Calculator - hide photos. https://play.google.com/store/apps/details?id=com.hld.anzenbokusufake&hl=en&gl=US, 2023. [Accessed 20-Apr-2023].

[24] FishingNet. Calculator - photo vault. https://play.google.com/store/apps/details?id=com.hld.anzenbokusucal&hl=en&gl=US, 2023. [Accessed 20-Apr-2023].

[25] FishingNet. Sgallery - hide photos & video. https://play.google.com/store/apps/details?id=com.hld.anzenbokusu&hl=en&gl=US, 2023. [Accessed 20-Apr-2023].

[26] Feodor Fitsner. Flet. https://flet.dev/, 2023. [Accessed 10-Jan-2023].

[27] Magnet Forensics. Magnet Axiom. https://www.magnetforensics.com/products/magnet-axiom/, 2023. [Accessed 8-Jun-2023].

[28] Oxygen Forensics. Oxygen Forensic Detective. https://oxygenforensics.com/en/products/oxygen-forensic-detective/, 2023. [Accessed 1-Jun-2023].

[29] Signal Foundation. Signal Private Messenger. https://play.google.com/store/apps/details?id=org.thoughtcrime.securesms&hl=en&gl=US, 2023. [Accessed 20-Apr-2023].

[30] GCHQ. The Cyber Swiss Army Knife. https://github.com/gchq/CyberChef, 2023. [Accessed 1-Jun-2023].

[31] Privacy Geeks. LOCKED Secret Photo Vault. https://play.google.com/store/apps/details?id=com.lkd.calculator&hl=en&gl=US, 2023. [Accessed 20-Apr-2023].

[32] Genymobile. Genymotion. https://www.genymotion.com/, 2023. [Accessed 8-Jun-2023].

[33] Kasagiannis Georgios. Security evaluation of Android Keystore. https://dione.lib.unipi.gr/xmlui/handle/unipi/11252, 2022. [Accessed 6-Jun-2023].

[34] Alexander S. Gillis. Diffie-Hellman key exchange. https://www.techtarget.com/searchsecurity/definition/Diffie-Hellman-key-exchange, 2023. [Accessed 1-Jun-2023].

[35] Google. Android Keystore system. https://developer.android.com/training/articles/keystore, 2018. [Accessed 13-Jun-2023].

[36] Google. Android Open Source Project. https://source.android.com/, 2023. [Accessed 1-Jun-2023].

[37] Google. Chrome DevTools. https://developer.chrome.com/docs/devtools/, 2023. [Accessed 8-Jun-2023].

[38] Google. Locking/Unlocking the Bootloader. https://source.android.com/docs/core/architecture/bootloader/locking_unlocking, 2023. [Accessed 1-Jun-2023].

[39] Google. The Chromium Projects. https://www.chromium.org, 2023. [Accessed 13-Jun-2023].

[40] Hak5. DuckyScript. https://docs.hak5.org/hak5-usb-rubber-ducky/duckyscript-tm-quick-reference, 2023. [Accessed 10-Jan-2023].

[41] Hak5. Keystroke Reflection. https://cdn.shopify.com/s/files/1/0068/2142/files/hak5-whitepaper-keystroke-reflection.pdf?v=1659317977, 2023. [Accessed 10-Jan-2023].

[42] Ian Urbina (he New York Times). The Secret Life of Passwords. https://www.nytimes.com/2014/11/19/magazine/the-secret-life-of-passwords.html, 2023. [Accessed 8-Jun-2023].

[43] Terry Hughes. Google and Android Are Not the Same... and That's a Good Thing. https://appdevelopermagazine.com/google-and-android-are-not-the-same...-and-that's-a-good-thing/, 2023. [Accessed 1-Jun-2023].

[44] Snap Inc. Snapchat. https://play.google.com/store/apps/details?id=com.snapchat.android&hl=en&gl=US, 2023. [Accessed 20-Apr-2023].

[45] Wickr Inc. Wickr Me - Private Messenger. https://play.google.com/store/apps/details?id=com.mywickr.wickr2&hl=en&gl=US, 2023. [Accessed 20-Apr-2023].

[46] Burt Kaliski. PKCS #5: Password-Based Cryptography Specification Version 2.0. https://datatracker.ietf.org/doc/html/rfc2898#section-5.2, 2000. [Accessed 13-Jun-2023].

[47] Jihun Son; Yeong Woong Kim; Dong Bin Oh; Kyounggon Kim. Forensic analysis of instant messengers: Decrypt Signal, Wickr, and Threema. https://www.sciencedirect.com/science/article/pii/S2666281722000166?dgcid=rss_sd_all#!, 2022. [Accessed 6-Jun-2023].

[48] Kutatua. MD5, SHA1, MD5Crypt, BCrypt Password Time to Crack Calculator. https://kutatua.com/password/time-to-crack-calculator, 2000. [Accessed 13-Jun-2023].

[49] DoMobile Lab. AppLock. https://play.google.com/store/apps/details?id=com.domobile.applockwatcher&hl=en&gl=US, 2023. [Accessed 20-Apr-2023].

[50] RSA Laboratories. What are MD2, MD4, and MD5? https://web.archive.org/web/20170116172936/http://www.emc.com/emc-plus/rsa-labs/standards-initiatives/md2-md4-and-md5.htm, 2017. [Accessed 5-Jun-2023].

[51] Google LLC. Google Chrome: Fast & Secure. `https://play.google.com/store/apps/details?id=com.android.chrome&hl=en&gl=US`, 2023. [Accessed 20-Apr-2023].

[52] Squid Tooth LLC. Vaulty : Hide Pictures Videos. `https://play.google.com/store/apps/details?id=com.theronrogers.vaultyfree&hl=en&gl=US`, 2023. [Accessed 20-Apr-2023].

[53] Mega Ltd. MEGA. `https://play.google.com/store/apps/details?id=mega.privacy.android.app&hl=en&gl=US`, 2023. [Accessed 20-Apr-2023].

[54] WeChat International Pte. Ltd. WeChat. `https://play.google.com/store/apps/details?id=com.tencent.mm&hl=en&gl=US`, 2023. [Accessed 20-Apr-2023].

[55] Niels Provos; David Mazières. A Future-Adaptable Password Scheme. `https://www.usenix.org/legacy/events/usenix99/provos/provos.pdf`, 1999. [Accessed 13-Jun-2023].

[56] Meta. Introducing Conceal: Efficient storage encryption for Android. `https://engineering.fb.com/2014/02/03/android/introducing-conceal-efficient-storage-encryption-for-android/`, 2023. [Accessed 1-Jun-2023].

[57] AVG Mobile. AVG AntiVirus & Security. `https://play.google.com/store/apps/details?id=com.antivirus&hl=en&gl=US`, 2023. [Accessed 20-Apr-2023].

[58] MOBILedit. MOBILedit Forensic. `https://www.mobiledit.com/mobiledit-forensic`, 2023. [Accessed 1-Jun-2023].

[59] MSAB. XAMN - Mobile forensic data analysis software. `https://www.msab.com/product/analyze/`, 2023. [Accessed 1-Jun-2023].

[60] Ngrok. Ngrok. `https://ngrok.com/`, 2023. [Accessed 10-Jan-2023].

[61] Juraj Novak. SQLite Viewer. `https://inloop.github.io/sqlite-viewer/`, 2023. [Accessed 1-Jun-2023].

[62] Philippe Oechslin. Making a Faster Cryptanalytic Time-Memory Trade-Off. `https://link.springer.com/chapter/10.1007/978-3-540-45146-4_36`, 2023. [Accessed 13-Jun-2023].

[63] United States National Institute of Standards and Technology. Secure Hash Standard. `https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf`, 2015. [Accessed 5-Jun-2023].

[64] United States National Institute of Standards and Technology. Federal Information Processing Standards Publication 197. https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf, 2023. [Accessed 1-Jun-2023].

[65] OWASP. Manipulator-in-the-middle attack. https://owasp.org/www-community/attacks/Manipulator-in-the-middle_attack, 2023. [Accessed 7-Jun-2023].

[66] Mauricio Piacentini. DB Browser for SQLite. https://sqlitebrowser.org/, 2023. [Accessed 1-Jun-2023].

[67] The Android Playground. Apps Lock & File Encryption. https://play.google.com/store/apps/details?id=playground.develop.applocker&hl=en&gl=US, 2023. [Accessed 20-Apr-2023].

[68] Christof Paar; Jan Pelzl; Bart Preneel. *Understanding Cryptography: A Textbook for Students and Practitioners.* Springer, 2010.

[69] Haroon Q Raja. Android Partitions Explained: boot, system, recovery, data, cache & misc. https://www.addictivetips.com/mobile/android-partitions-explained-boot-system-recovery-data-cache-misc/, 2023. [Accessed 1-Jun-2023].

[70] Viber Media S.a r.l. Viber - Safe Chats And Calls. https://play.google.com/store/apps/details?id=com.viber.voip&hl=en&gl=US, 2023. [Accessed 20-Apr-2023].

[71] Samsung. What is an OTG cable and what is it used for? https://www.samsung.com/nz/support/mobile-devices/what-is-an-otg-cable-and-what-is-it-used-for/, 2023. [Accessed 10-Jan-2023].

[72] Denis Sazonov. Andriller. https://github.com/den4uk/andriller, 2023. [Accessed 1-Jun-2023].

[73] Bruce Schneier. The Legacy of DES. https://www.schneier.com/blog/archives/2004/10/the_legacy_of_d.html, 2004. [Accessed 7-Jun-2023].

[74] Bruce Schneier. Encryption Workarounds. https://www.schneier.com/wp-content/uploads/2017/03/Encryption_Workarounds-2.pdf, 2017. [Accessed 14-Jun-2023].

[75] Cornell Law School. Daubert standard. https://www.law.cornell.edu/wex/daubert_standard, 2023. [Accessed 1-Jun-2023].

[76] Cornell Law School. Frye Standard. https://www.law.cornell.edu/wex/frye_standard, 2023. [Accessed 1-Jun-2023].

[77] Brave Software. Brave Private Web Browser. `https://play.google.com/store/apps/details?id=com.brave.browser&hl=en&gl=US`, 2023. [Accessed 20-Apr-2023].

[78] SQLabs. SQLiteManager. `https://www.sqlabs.com/sqlitemanager.php`, 2023. [Accessed 1-Jun-2023].

[79] Statista. Market share of mobile operating systems in India from 2012 to 2022. `https://www.statista.com/statistics/262157/market-share-held-by-mobile-operating-systems-in-india/`, 2023. [Accessed 1-Jun-2023].

[80] Statista. Mobile operating systems' market share worldwide from 1st quarter 2009 to 1st quarter 2023. `https://www.statista.com/statistics/272698/global-market-share-held-by-mobile-operating-systems-since-2009/`, 2023. [Accessed 1-Jun-2023].

[81] Statista. Number of available applications in the Google Play Store from December 2009 to March 2023. `https://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store/`, 2023. [Accessed 1-Jun-2023].

[82] Statista. Share of active phones with enabled biometrics in North America, Western Europe & Asia Pacific from 2016 to 2020. `https://www.statista.com/statistics/1226088/north-america-western-europe-biometric-enabled-phones/`, 2023. [Accessed 10-Jan-2023].

[83] Statista. Tablet shipments market share by vendor worldwide from 2nd quarter 2011 to 1st quarter 2023. `https://www.statista.com/statistics/276635/market-share-held-by-tablet-vendors/`, 2023. [Accessed 1-Jun-2023].

[84] Christoph Wille. Storing Passwords - done right! `http://www.aspheute.com/english/20040105.asp`, 2023. [Accessed 13-Jun-2023].

[85] Rafael Alvarez; Candido Caballero-Gil; Juan Santonja; Antonio Zamora. Algorithms for Lightweight Key Exchange. `https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5551094`, 2017. [Accessed 5-Jun-2023].

[86] Zetetic. SQLCipher 3.0.0 Release. `https://www.zetetic.net/blog/2013/11/11/sqlcipher-300-release.html`, 2013. [Accessed 13-Jun-2023].

[87] Zetetic. SQLCipher 4.0.0 Release. `https://www.zetetic.net/blog/2018/11/30/sqlcipher-400-release/`, 2018. [Accessed 13-Jun-2023].