Why You Need To Protect Your Cryptographic Keys

zKEYBOX™

ZIMPERIUM®

Executive Summary

In past decades, data security was identified with massive physically secure data centers and corporate controlled computing assets. Today's reality is that many software applications are running on unmanaged devices in vulnerable and targeted networks. Adversaries can easily gain physical access to many devices that need to protect internal secrets, including mobile phones, IoT devices, automobiles, set-top boxes, and medical equipment. Even in a well-secured corporate setting, the perimeter is increasingly hard to define and defend because not all devices on a corporate network are adequately managed or secure, (e.g. BYOD). Widespread malware deployments make it likely that devices, no matter how well managed, are subject to infection. Consequently, there is a high risk that adversaries can easily examine and attack these kinds of devices.

In this paper, we will focus on one particular security risk that is inevitable in today's open and insecure digital environments — namely, **the security of cryptographic keys**. As will be further explained, a cryptographic key is the cornerstone concept of most security schemes used on billions of devices all over the world. While cryptography is designed to ensure protection of confidential data, it does not automatically eliminate the risk of attacks on such data because cryptographic security relies on the security of keys. In reality, cryptography merely shifts the problem of protecting data to protecting keys.

In fact, in many situations, a single key may protect many different pieces of data, and so securing those keys is of paramount importance. Unwarranted extraction of a key from a cryptographic module essentially nullifies the entire security system. The consequences of a compromised key can include financial loss, liability, regulatory fines and impact to brand reputation.

The overview of common techniques hackers use to discover keys will be provided, such as the use of static and dynamic analysis, network eavesdropping, and side-channel attacks. In addition, the established methods for fighting these attacks will also be discussed, and the concept of white box cryptography will be explained.

Finally, this white paper will focus on Zimperium's industry-leading solution to protecting cryptographic keys in software — **zKeyBox**, a white box cryptography library that provides a secure implementation of the standard cryptographic algorithms that **completely hides the cryptographic keys** in the binary code and makes key extraction attempts extremely difficult.

The Challenge of Keeping Cryptographic Keys Safe

Cryptography is the foundation of data security in digital assets and services used by millions every day. It enables secure communication, strong authentication, and protection of confidential information. Bank cards, ATMs, Pay TV, cloud computing, online payments, and connected cars are just a few examples of modern systems that would be highly vulnerable and impractical without the use of cryptography.

At the core of cryptography lies the concept of a key — a small piece of information that determines the output of cryptographic operations (encryption, decryption, signing, verification, etc.). Having access to the right key opens the door to all the secret data protected by that particular cryptographic algorithm and that key.

While those that use cryptographic algorithms generally acknowledge the need to protect their secret data, the necessity to protect the cryptographic keys themselves is often overlooked. A misguided assumption is that the secret cryptographic keys are not accessible to the adversary; however, that is not the case. In the vast majority of cases, cryptographic algorithms expose their keys in the clear to the execution environment in one way or another. There are many ways how the keys can be obtained, as explained later in the "How Adversaries Attack Cryptographic Keys" section. Therefore, one of the main points of emphasis is that **it is absolutely critical to protect cryptographic keys**.

If hackers were to obtain cryptographic keys, they could potentially eavesdrop on secure communication, spoof a user, manipulate network transactions, and/or infiltrate the system to exfiltrate confidential information. The effects of broken cryptographic modules and stolen keys can be significant for governments, financial institutions, automotive manufacturers, healthcare organizations, and gaming distributors. Financial loss, damaged brand reputation, exposure to liability, and sometimes even loss of human life can all result from the failure to ensure adequate protection of cryptographic keys.

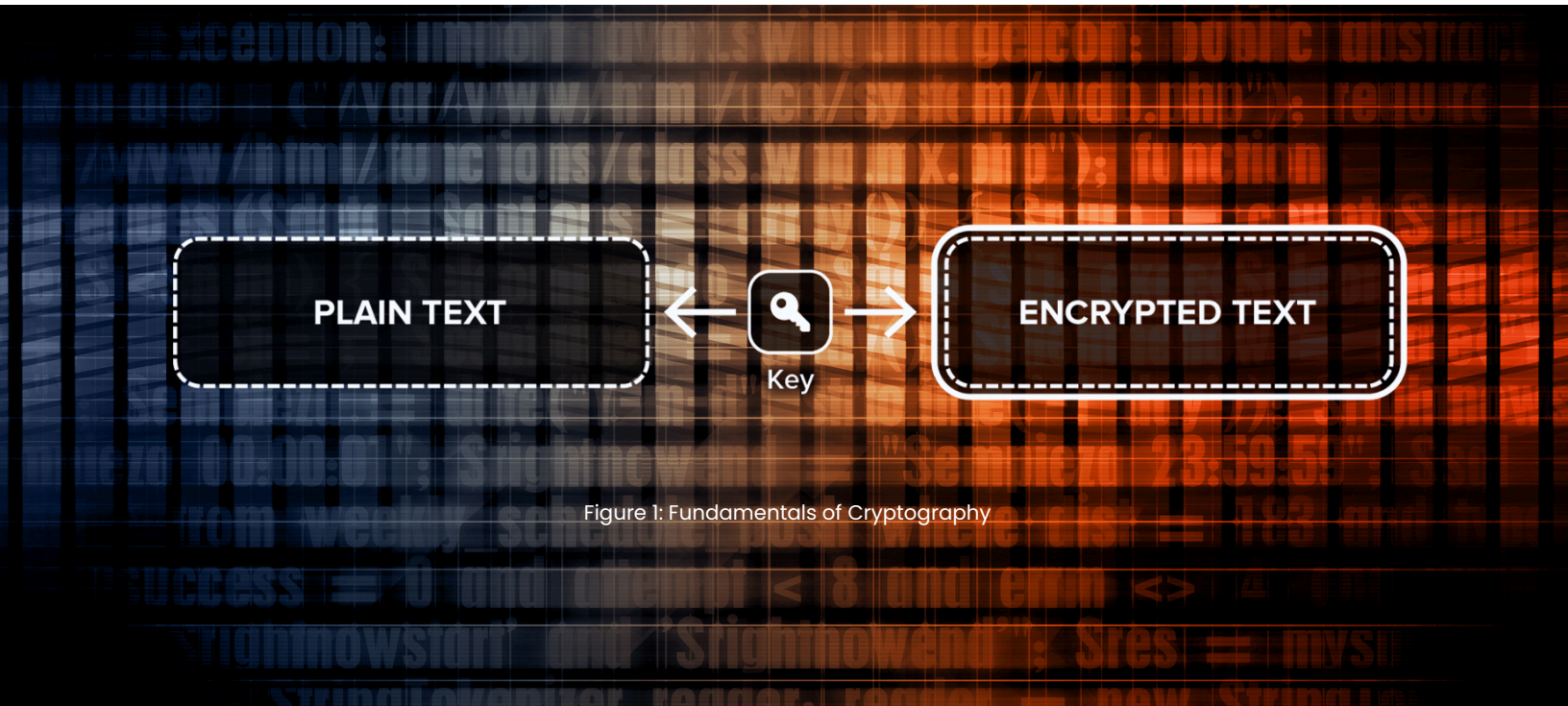

Figure 1: Fundamentals of Cryptography

Examples of Cryptographic Key Attacks

The following are some of the well-known attacks on large organizations involving discovery of cryptographic keys.

Volkswagen Remote Key System

In 2016, a team of computer scientists published a paper on a flaw that applies to practically every car Volkswagen has sold since 1995. By using an inexpensive and readily available piece of radio hardware, they could intercept signals from a victim's key fob, discover the secret keys used, and then clone the original remote.[1]

Tesla Mobile Application

In 2016, a team of security experts demonstrated a vulnerability that allowed them to gain full control over a Tesla Model S by overcoming the security measures of the Tesla mobile application. The application is authenticated using a secret key stored locally by the application. Since the key was stored in the clear, it became vulnerable to theft by malware on a mobile device. The vulnerability was executed in practice by installing a malicious version of the Tesla mobile application.

Nintendo Wii Console

In 2007, a hacker was able to obtain secret encryption keys used on the Nintendo Wii console by exploiting a bug in the signature verification algorithm and compromising the keys that were stored in the external GDDR3 RAM in unencrypted form. As a result, the anti-piracy measures of the console were broken, allowing unsanctioned software to be installed and run on the Wii hardware.[2]

How Adversaries Attack Cryptographic Keys

This section provides an overview of the common methods used by hackers to extract secret keys from various systems. The focus is that key extraction is a serious risk and safeguarding your systems against such attacks is a significant task.

Brute-Force Attack

In a brute-force attack, the attacker tries a huge number of inputs to see if one works. For example, many password-cracking algorithms (Brutus, RainbowCrack) work this way, trying millions of common passwords until one is found that works. That is why you are always asked to pick passwords with hard to remember combinations of upper- and lower-case digits, numbers, and special characters.

Usually, brute-force attacks are only effective for breaking cryptographic algorithms that deal with small key sizes. With the industry's latest standard crypto algorithms, brute-force attacks are generally unfeasible.

Theoretical Loopholes and Implementation Errors

Threat actors might attempt to find theoretical weaknesses or implementation bugs in cryptographic algorithms or protocols that would allow them to quickly bypass the security protections inherent in a particular algorithm or protocol. A classic example of this is the man-in-the middle attack against the Needham-Schroeder Public-Key Protocol.[3] This attack demonstrated a fundamental weakness in the protocol that enabled an unforeseen attack to succeed. The Wired Equivalent Privacy (WEP) protocol is another case where a theoretical vulnerability was discovered and published in a paper.[4] A more recent example that leveraged a vulnerability in the OpenSSL cryptographic[5] software library was the notorious Heartbleed vulnerability.

As demonstrated by these examples, even well-established standards and systems are subject to the risk of being attacked and compromised.

Static Analysis

By analyzing the static machine code of a software executable such as the binary image in the device storage, hackers can easily discover cryptographic keys if they are stored in the clear. Identifying potential keys in the code is made easier by the fact that cryptographic keys are random sets of bits exhibiting high entropy. In contrast, most uncompressed machine code has relatively low entropy.[6] As a result, a key is likely to stand out against the background of low-entropy non-key data. The following figure visualizes machine code in 2D, such that one pixel represents one bit, and each column represents 64 bits of sequential data (ordered left to right). A human eye can quickly identify a region characterized by high randomness, which may indicate a cryptographic key. The process of pinpointing of such regions can be easily automatized.

Static analysis is one of the most effective attacks if the hacker has access to the device storage or any channel used to deploy the executable code, and if the keys are stored as cleartext.

Static analysis is one of the most effective attacks if the hacker has access to the device storage or any channel used to deploy the executable code, and if the keys are stored as cleartext.

Figure 2: Discovering High-Entropy Key Material Within the Binary Code

Dynamic (or Memory) Analysis

While encrypting a key on a storage medium is a fairly simple procedure, hiding the key in device memory is much more complicated because at some point, the key needs to be provided to a cryptographic algorithm as valid input. In most cryptographic libraries this is the moment when the key is decrypted in the memory as plaintext and becomes susceptible to extraction. With the right set of tools, attackers can dynamically analyze the memory and hijack cryptographic secrets during execution of the software. There are automated tools which are readily available that can instantly discover secret keys in any arbitrary process running on a device.[7]

Eavesdropping on Network Communication

Secret keys should never be transferred over any network in unencrypted form, as this enables threat actors to easily exploit keys. From a security point of view, the Internet should be viewed as a completely transparent ecosystem where hackers can potentially see all the data you exchange with other endpoints. Consequently, it becomes absolutely clear that cryptographic keys and other secrets sent through the Web must always be protected. The common practice is to encrypt all secrets before they are sent over the Internet, and never expose these keys used for encrypting the secrets. There are established methods for agreeing on encryption keys on both endpoints without sending them over the Internet, such as the Diffie-Hellman key exchange algorithym.[8]

Side-Channel Attacks

In these attacks, the attacker does not attempt to access the key directly in the device, but rather attempts to reconstruct the key from indirect signals and the physiology of internal components in the device. For example, in some cases it is possible to reconstruct a key by measuring the power consumption of a chip.[9]

In another example, the attacker injects faults into the algorithm by subjecting the hardware to extreme temperature and then observes the behavior of the algorithm in order to reconstruct the key.[10]

Under certain circumstances, keys can be extracted from devices even when they are powered off. This type of side-channel attack relies on memory retention that is common in most modern devices. Even after the device is powered down, the internal memory retains its contents for seconds to minutes at normal operating temperatures, even if it is removed from a motherboard.[11] To execute the attack, a hard reboot of the device is performed and a removable disk is then immediately used to boot a lightweight operating system, or in some cases the memory modules are removed from the original system and quickly placed in a compatible machine. Further analysis can then be performed against the information that was dumped from memory to find the cryptographic keys contained in it. Automated tools are now available to perform this task for attacks against some popular encryption systems.

Methods for Protecting Cryptographic Keys

In the previous section, we explained the importance of keeping cryptographic keys hidden and safe — a fact that is often ignored even by large corporations. At the same time, we showed that ensuring good key protection is not an easy task since there are a wide array of techniques that hackers use to attack cryptographic systems and steal keys.

In this section, we outline the main categories of countermeasures against discovery of cryptographic keys.

Hardware-Based Security

To deal with key protection challenges, hardware-based security is commonly used to provide strong protection for the keys on devices. Some examples are hardware security modules (HSM), trusted platform modules (TPM), and trusted execution environments (TEE). The security of these systems relies on the fact that it is very difficult and expensive for attackers to reverse engineer a hardware module and manipulate its internal data. Generally speaking, hardware security systems can be considered "black box models" because their internal workings are essentially hidden to the observer.

Although hardware-based approaches do provide excellent security advantages, there are also significant downsides:

- Hardware-based security **adds cost** to a system. Manufacturers of platforms might choose cost sensitivity over the security risks of compromised keys — security is usually an afterthought.
- Vulnerabilities in hardware are **difficult and potentially expensive to mitigate**. Examples like Meltdown and Spectre[12] illustrate that hardware and software manufacturers might need to spend large amounts of money and resources to issue patches to fix vulnerabilities in existing deployments.
- Different devices may contain **different hardware with varying functionality** that require complex logic in applications built to run on a wide range of devices.
- As it was explained in the "Side-Channel Attacks" section, **hardware is not immune to attacks**. Clever approaches such as differential power analysis can be used to extract keys from hardware by examining indirect patterns in signals emanating from the hardware.
- There are business models which **preclude application developers from using secure hardware** on a device even when it exists. Such is the case with Apple iPhone, where although it has ARM processors with the TrustZone extension, third-party applications are generally not allowed to use that functionality.

Keystores

Most operating systems and execution platforms offer some kind of means for storing and using cryptographic keys in a secure manner. Examples of these include Android Keystore, Java Keystore, Apple Secure Enclave, and Windows Keystore. In some cases, these keystores are backed up by hardware-based security, if such technology is available on the device. Typically, keystores are used for certificate and key pair management associated with SSL communication.

While such keystores are sufficiently secure, they cannot be considered general-purpose cryptographic libraries. For instance, usually the list of supported cryptographic algorithms and operations is quite limited. Moreover, in some cases it is not possible to import an existing key into the keystore. Another important factor to consider is that such keystores are built for a particular target platform, which means that supporting the same application on multiple platforms will require re-implementing the cryptographic operations on each of them. Because of these reasons, relying on a platform-specific keystore may be impractical and expensive, depending on the use case.

White Box Cryptography

The objective of white box cryptography is to implement cryptographic primitives in such a way that, within the context of the intended application, **having full access to the cryptographic implementation does not present any advantage for an adversary** in comparison to the adversary working with the implementation as a black box.[13] In simple terms, white box cryptography is a general-purpose software implementation of cryptographic algorithms that attempts to hide keys. Since software is easily examinable if the hacker has access to the device, such software execution environment is called a "white box model".

Academic Work on White Box Cryptography

The premise of white box cryptography may seem like impossible magic, but university researchers have been studying the problem of general obfuscation since 2001. Over time, several academic derivatives of white box cryptography have emerged, such as the following:

- **Functional encryption** (since 2005) combines basic encryption with mathematically forged access control.
- **Fully homomorphic encryption** (since 2009) enables secure computing with encrypted data on an untrusted cloud server.
- **Indistinguishability obfuscation** (since 2013) achieves (as well as theoretically possible) general software obfuscation which has been called "crypto-complete" as a flood of exotic cryptographic applications can be built from indistinguishable obfuscation.

While most of these advanced cryptographic techniques are theoretically possible, they are practically infeasible as they require enormous amounts of computational resources to solve even the simplest problems. These are active areas of investigation, and researchers are making continual progress. However, it may be decades before some of these techniques are practical.

How White Box Cryptography Works

To implement white box cryptographic primitives it is necessary to provide functionality equivalent to the standard algorithms without revealing the intermediate values arising within the usual algorithms. One general technique is to encode and thereby obscure inputs, outputs, and intermediate values. Another technique is to rearrange steps into less revealing combined operations.

As illustrated in this figure, in a "regular" or unobfuscated implementation, the secret keys and execution logic are clearly distinguishable and easy to tamper with. In a white box implementation, the internal data and execution flow are obscure and inseparable — the keys cannot be easily extracted and making any modifications to the code can result in
breaking the entire executable. One way this is frequently done in white box implementations is to move computations into tables which can be easily randomized and are difficult to reverse engineer.

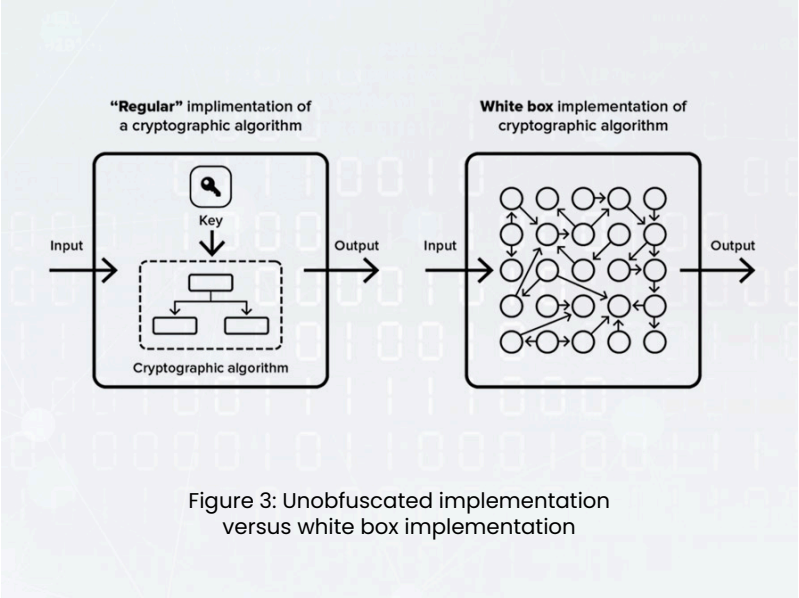

Figure 3: Unobfuscated implementation versus white box implementation

Choosing the Right Key Protection Technique

Generally speaking, software-based security cannot be considered as safe as dedicated purpose-built security hardware, and computations performed within a software white box environment will always be slower. However, the obvious advantage of white box software algorithms over their black box hardware counterparts is that they can be deployed on devices without hardware support. White box software algorithms can support the same functionality on any platform, and they can be easily and cost-effectively upgraded if vulnerabilities are found. In some cases, it may be desirable to have both software and hardware protection in place to provide defense in depth. All these factors must be carefully evaluated when choosing the desired key protection technique.

Zimperium's zKeyBox is the world's leading implementation of white box cryptography algorithms that provides a robust solution to the problem of securing keys in software and ensures protection against the vast majority of key attacks including static and dynamic analysis as well as side-channel attacks.

The subsequent part of this white paper will be focusing on the zKeyBox library and how the particular features address various threats aimed at cryptographic keys and other inner parts of cryptographic algorithms.

Zimperium's zKeyBox

zKeyBox is a cross-platform library that provides advanced white box implementation of a number of cryptographic algorithms. It allows standard cryptographic functions to be performed without the keys ever being in the clear. Because of its strong protection design, zKeyBox is extremely difficult to reverse engineer and tamper with. zKeyBox employs **patented technologies** and has successfully passed a number of third-party security audits.

In the case of existing software applications that already have cryptographic modules in place, zKeyBox can simply replace those modules in code. Therefore, the zKeyBox-protected application will be functionally equivalent to the original application and ensure robust protection of its keys.

The general procedure for applying zKeyBox protection is as follows:

1. Link the static zKeyBox library with the target application that you want to protect.
2. Change the code that uses the low-level cryptographic functions so that they employ the zKeyBox API.
3. Build and deploy your zKeyBox-protected application.

Figure 4: Applying zKeyBox protection to the target application

Main Features

zKeyBox provides white box implementation for a number of industry's standard algorithms that can be run on a vast array of target platforms as can be seen in the following tables:

Supported algorithms
Encryption
Decryption
Signing
Verification
Key generation
Key wrapping
Key unwrapping
Key agreement
Digests (hashing) of keys
Key derivation

Supported platforms
Android
iOS
tvOS
macOS
Windows
PlayStation
glibc/Linux
uClibc/Linux
musl/Linux
MinGW

The most popular ciphers such as AES, RSA, ECC, DES, and Speck are supported. Since the exact list of supported algorithms and target platforms is constantly changing, please consult the zKeyBox User Guide or contact your Zimperium account executive for the latest set of supported functions and platforms.

Security Aspects

In this section, we touch upon some of the generic security characteristics of zKeyBox.

Encrypted Domain

An encrypted domain is a part of a program where all the data is stored in encrypted form and all the operations are obfuscated. Due to the execution speed trade-off involved (since obfuscating code necessarily results in a performance penalty), an encrypted domain is typically never used for an entire program, but rather just for its crucial parts such as the cryptographic algorithms and the program code that handles the keys.

zKeyBox provides a complete encrypted domain for working with cryptographic keys. The library exposes a set of API functions to the calling application in such a way that there is no possibility (and no need) for the application or hacker to obtain the keys in plaintext. Computation in an encrypted domain is a central feature of zKeyBox.

This means that even when a cryptographic algorithm is being executed, the keys and other data it is working with are never revealed in plaintext. In addition, any attempts to tamper with the algorithm or separate keys from it will most likely result in crashing the application.

Obfuscation

zKeyBox hides the secret keys and the execution flow of the cryptographic algorithms. It is nearly impossible to reverse engineer the logic and trace the logical steps. Since the standard debugging tools yield no meaningful statistics to the threat actors analyzing the unique white box code, the traditional tampering methods are ineffective with zKeyBox.

Therefore, even if there are any theoretical weaknesses discovered in the industry's cryptographic algorithms implemented by zKeyBox, the obfuscated nature of the way the library works will greatly encumber the potential attacks or even render them impossible.

Diversification

Software diversification is a method of adding randomization to an executable binary and its input and output data so that various instances of the same software appear different in every case. Software diversification confounds an attacker's attempts to exploit information gained from one deployment to compromise other deployments. It is much harder to develop a universal cracking scheme for software instances that are diversified, i.e., each software instance must be cracked individually.

Diversification is an integral component of all Zimperium products. zKeyBox in particular, has a two-tier diversification scheme in place. First, the binary of each zKeyBox instance is generated from a random seed which ensures code diversity, meaning, the binary footprint of every application that employs the library is unique, rendering creation of universal cracking tools almost impossible. Second, every zKeyBox instance uses a different pattern for encrypting the keys it saves and loads from the storage (data diversity). This means that the hacker cannot take the zKeyBox library from one compromised application and use it to decrypt keys from other applications.

Protection against White Box Attacks

The research team behind Secure Key Box is constantly self-testing and improving the product to ensure security against known white box attacks. One example is the Billet attack, which is probably the best-known attack that can be made against certain types of AES white box implementations.[14] The attack depends on certain characteristics to be present in the particular AES white box implementation. For example, it is assumed the white box implementation looks like a sequence of S-box applications and permutations of the encoded bytes. The zKeyBox implementation of AES however, does not have the characteristics that allow the specified type of attack to be applied. Hence, the attack is rendered useless against zKeyBox.

Select Use Cases

Tokenized EMV Payment Solution

A typical use case for zKeyBox is to secure parts of a tokenized EMV payment solution on a mobile device. The main functions of such systems include device provisioning, token provisioning, storage of token data, and token processing.

Device provisioning involves establishing an identity of the mobile device and linking it to the identity of the cardholder within the payment ecosystem. During this process the device acquires a unique key that is linked with the cardholder identity (as known by the card issuer). Device provisioning may use a key agreement scheme between device and server, key derivation, a digital signature for authentication, encryption/decryption for session traffic, as well as secure and device-bound storage of the acquired key. Token provisioning is requesting and receiving the single-use tokens for later use in payment transactions. During this process, a digital signature is used for authentication, and encryption and decryption are used for session traffic and protection of token data while in transit and while at rest on the device. Token processing happens during the payment when the token is used (as a replacement for the PAN card number); it involves decrypting token data, calculating the authentication value (Retail MAC), and encrypting the modified token data.

As can be seen, a large number of cryptographic operations are involved in this use case. All of these operations are supported by zKeyBox while ensuring that the involved keys and other secrets are never revealed in the clear. This allows deployment of the payment application on devices that do not support the hardware-based security environment and on devices where such environment is not available to developers.

Digital Rights Management System

Global entertainment and media companies have increased their value through innovative global streaming services, programs, live concerts, daily behind-the-scenes interviews, live sports broadcasts and a variety of music and news events that can be viewed on mobile devices. More importantly, consumers can now view specific entertainment content on their own devices just about anywhere, including planes, taxis, and other forms of public transportation. To protect the content from being stolen, digital rights management (DRM) systems must be in place, and to protect the players' applications themselves, mobile security application solutions are a necessity.

Because DRM systems involve multiple cryptographic operations and depend on the integrity of cryptographic keys, developers should add a layer of protection to their DRM applications to prevent hackers from breaking the DRM system or stealing the secret keys. zKeyBox is an ideal tool for this purpose because it supports all the industry standard cryptographic algorithms used in DRM solutions, and never reveals cryptographic keys in the clear.

Next Steps

This white paper has presented an in depth look into Zimperium's zKeyBox. Our state-of-the art protection mechanisms will help you shield your cryptographic keys from attacks and protect the most important assets for you and your customers.

Contact us to see how Zimperium can help you protect your cryptographic keys, get a demo, start a free trial, or to learn more.

Sources

1 https://www.documentcloud.org/documents/3010178-Volkswagen-amp-HiTag2-Keyless-Entry-System.html

2 https://marcan.st/uploads/25c3_console_hacking

3 G. Lowe, "An attack on the Needham-Schroeder public key authentication protocol", Information Processing Letters, Volume 56, Issue 3, 1995

4 S. Fluhrer, I. Mantin, A. Shamir, "Weaknesses in the Key Scheduling Algorithm of RC4", Selected Areas in Cryptography. SAC 2001. Lecture Notes in Computer Science, vol 2259, 2001

5 http://heartbleed.com

6 A. Shamir, N. van Someren, "Playing Hide and Seek With Stored Keys", Financial Cryptography. FC 1999. Lecture Notes in Computer Science, vol 1648, 1998

7 https://github.com/mmozeiko/aes-finder

8 https://tools.ietf.org/html/rfc2631

9 P. Kocher, J. Jaffe, B. Jun, "Differential Power Analysis", CRYPTO '99 Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology, 1999

10 M. Hutter, J. Schmidt, "The Temperature Side Channel and Heating Fault Attacks", CARDIS, 2013

11 J. A. Halderman et al, "Lest We Remember: Cold Boot Attacks on Encryption Keys", Proc. 17th USENIX Security Symposium (Sec '08), 2008

12 https://meltdownattack.com

13 B. Wyseur, "White Box Cryptography", PhD thesis, 2009

14 O. Billet, H. Gilbert, C. Ech-Chatbi, "Cryptanalysis of a White Box AES Implementation", Selected Areas in Cryptography. SAC 2004. Lecture Notes in Computer Science, vol 3357, 2005

About Zimperium

Zimperium secures mobile devices and mobile applications so they can safely access sensitive data and systems. We are an advanced machine learning-based solution with a privacy focus, supporting iOS, Android, and ChromeOS platforms.

Zimperium's Mobile Application Protection Suite (MAPS) helps enterprises to build secure and compliant mobile applications. It is the only unified solution that combines comprehensive in-app protection with centralized threat visibility.

- Our in-app protection includes application shielding, client-side runtime application self-protection (RASP), and anti-malware techniques.
- Our visibility enables continuous application security testing (AST) during development and runtime visibility into threats and attacks.

ZIMPERIUM®

Learn more at: zimperium.com
Contact us at: 844.601.6760 | info@zimperium.com

Zimperium, Inc
4055 Valley View, Dallas, TX 75244