Top 5 Cryptographic Key Protection Best Practices

Cryptographic keys are a foundational element of modern cybersecurity. They serve to keep data safely encrypted and help maintain secure networks for client-server communication. Unfortunately, this makes them a prime target for hackers. A single compromised key can give access to a goldmine of personal data and valuable IP, as well as enable other malicious actions such as unauthorized system access or signing digital certificates. Yet, despite its importance, many software developers still do not prioritize cryptographic key protection.
Recent cryptographic key attacks illustrate the extent and consequences of the problem:

- The theft of encryption keys from the world's largest SIM card manufacturer, which produces over two billion SIM cards a year. This access would hypothetically enable hackers to access and decrypt communications from mobile devices and modems that were supposed to be secure.
- Side-channel attacks that use "leaked" signals from devices to steal crypto keys, including a device created by researchers that could read electromagnetic frequencies while hidden in a piece of pita bread.
- A bank in South Africa stored its master encryption key in plain text at a data center, which was subsequently stolen by some employees who used it to commit fraud. The bank eventually had to replace 12 million bank cards.
- The massive Marriott hotel hack, which saw half a billion customer records stolen, would not have been so damaging if the hackers hadn't also managed to steal the cryptographic keys, which were stored on the same server as the encrypted customer data.

Most of these attacks could have been mitigated with basic crypto key protection strategies. So how can development teams keep keys safer? Here's a look at five best practices to implement.

Cryptographic key protection best practices

1. Never hard code keys in your software

This may seem self-evident, but it still occurs a shocking number of times, even by cybersecurity vendors. The use of a hardcoded cryptographic key greatly increases the risk that encrypted data may be recovered. This type of vulnerability is also notoriously difficult to fix, requiring a software update patch to remediate. For example, the cybersecurity vendor above took 18 months to remove hardcoded encryption keys from all of their software.

In addition, secure coding practices dictate that variables containing cryptographic keys should be overwritten after each use. This prevents compromise if that memory location is later accessed by untrusted code.

2. Limit keys to a single, specific purpose

Each key should be used for one application and purpose only, whether that is encryption, authentication, key wrapping, random number generation, or digital signature. Keys should be created with the appropriate key strength for their intended purpose—using it for a different process may not provide the necessary level of security. Reusing a key also can lead to greater damage in the event the key is compromised.

Care especially must be taken with key-wrapping keys, also known as key-encryption-keys (KEKs). These are keys used to protect other cryptographic keys. KEKs should always be of equal or greater strength than the cryptographic key they are wrapping and should never be used for an additional purpose, such as encrypting data or communications.

3. Use hardware-backed security when possible

Hardware security modules (HSMs) provide highly effective cryptographic key protection and may be mandated in certain use cases, like securing root keys in PKI. This physical device can perform cryptographic functions such as encryption, decryption, and key generation. Using an HSM removes the burden of secure key storage from a software's logic and reduces the chance that hackers will get access to data and the keys they need to decrypt it in one place. Other hardware-backed solutions include trusted platform modules (TPM) and trusted execution environments (TEE), which provide hardware isolated systems to perform cryptographic operations.

However, hardware-backed solutions can be prohibitive both in terms of cost and physical space. They are also vulnerable to side-channel attacks that measure the unintentional signals transmitted by physical devices (such as heat, sound, or time taken to perform an action).

4. Take advantage of white-box cryptography for key protection gaps

For high-value applications and data, and when hardware-backed key security is not feasible or sufficient, include software-based key protection. Applications cannot be certain that the mobile and desktop devices they run on will have the hardware capability needed to perform cryptographic operations securely and ensure crypto key protection. This is especially the case with web apps as browsers do not have access to underlying hardware security support, even if available. In these cases, white-box cryptography is recommended.

White-box cryptography uses multiple protection techniques to create a secure execution and storage environment for cryptographic functions in software and apps. zKeyBox, Zimperium's industry-leading white-box cryptography solution, uses patented technology to protect major crypto algorithms (including DES, AES, RSA, SPECK, ECC, ECDSA, DH, ECDH, and SHA) and has no dependency on any hardware based mechanisms provided by the platforms (ex: Keystores, Secure Enclave, Trusted Execution Environment (TEE) on Android).

Software-based key protection solutions are easier to implement than their hardware counterparts as they provide identical functionality across devices. This becomes especially critical for applications in regulated industries that must meet strict security standards. Strong white-box cryptography has the added advantage of protecting applications from speculative execution and other side-channel attacks.

5. Put robust key management in place

Key management involves creating a number of policies to ensure that cryptographic keys are not put in danger through ignorance or carelessness. Effective key management policies focus on:

- Key lifecycle: This includes secure handling of everything from key generation, distribution, and normal use, to replacement, expiration, archival, and destruction.
- Key storage and backup: As discussed earlier, cryptographic key protection depends on secure storage, such as hardware-based security devices or using white-box cryptography. Keys stored in offline devices/databases should be encrypted using KEKs before exporting and storing. Applications must include the ability to securely backup keys as data encrypted with a lost cryptographic key cannot be recovered.
- Access protections and restrictions: Access to cryptographic keys throughout their lifecycle must be tightly controlled, with users and level of access able to be identified. Such accountability is critical to both keep cryptographic keys safe and reduce the impact of any compromise that does occur.

How to get started

Fortunately, when it comes to the security of crypto keys, you don’t have to reinvent the wheel. A number of recommended standards and protocols, such as NIST 800-57, provide extensive guidelines on how to best protect secret keys and to what standard. The OWASP Key Management Cheat Sheet is an excellent resource on cryptographic key protection and management for developers.

Cryptographic key protection is an essential element of any cybersecurity strategy. Zimperium's zKeyBox is designed to keep cryptographic keys safe in some of the most vulnerable situations. To learn more about how the industry-leading zKeyBox solution works, contact us and talk to one of our security experts.