Key Management Cheat Sheet

From OWASP

Introduction

This Key Management Cheat Sheet provides developers with guidance for implementation of cryptographic key management within an application in a secure manner. It is important to document and harmonize rules and practices for:

1. key life cycle management (generation, distribution, destruction)
2. key compromise, recovery and zeroization
3. key storage
4. key agreement

General Guidelines and Considerations

Formulate a plan for the overall organization's cryptographic strategy to guide developers working on different applications and ensure that each application's cryptographic capability meets minimum requirements and best practices. Identify the cryptographic and key management requirements for your application and map all components that process or store cryptographic key material.

Key Selection

Selection of the cryptographic and key management algorithms to use within a given application should begin with an understanding of the objectives of the application. For example, if the application is required to store data securely, then the developer should select an algorithm suite that supports the objective of data at rest protection security. Applications that are required to transmit and receive data would select an algorithm suite that supports the objective of data in transit protection. We have provided recommendations on the selection of crypto suites within an application based on application and security objectives. Application developers oftentimes begin the development of crypto and key management capabilities by examining what is available in a library. However, an analysis of the real needs of the application should be conducted to determine the optimal key management approach. Begin by understanding the security objectives of the application which will then drive the selection of cryptographic protocols that are best suited. For example, the application may require:

1. Confidentiality of data at rest and confidentiality of data in transit.
2. Authenticity of the end device
3. Authenticity of data origin
4. Integrity of data in transit
5. Keys to create the data encryption keys

Once the understanding of the security needs of the application is achieved, developers can determine what protocols and algorithms are required. Once the protocols and algorithms are understood, you can you can begin to define the different types of keys that will support the application's objectives. There are a diverse set of key types and certificates to consider, for example:

1. Encryption: - Symmetric encryption keys - Asymmetric encryption keys (public and private)
2. Authentication of End Devices: - Pre-shared symmetric keys - Trusted certificates - Trust Anchors
3. Data Origin Authentication - HMAC
4. Integrity Protection - Message Authentication Codes (MACs)
5. Key Encryption Keys

Algorithms and Protocols

According to NIST SP 800-57 Part 1, many algorithms and schemes that provide a security service use a hash function as a component of the algorithm. Hash functions can be found in digital signature algorithms (FIPS186), Keyed-Hash Message Authentication Codes (HMAC) (FIPS198), key-derivation functions/methods (NIST Special Publications (SP) 800-56A, 800-56B, 800-56C and

800-108), and random number generators (NIST SP 800-90A). Approved hash functions are defined in FIPS180. NIST SP 800-57 Part 1 recognizes three basic classes of approved cryptographic algorithms: hash functions, symmetric- key algorithms and asymmetric-key algorithms. The classes are defined by the number of cryptographic keys that are used in conjunction with the algorithm.

Cryptographic hash functions

Cryptographic hash functions do not require keys. Hash functions generate a relatively small digest (hash value) from a (possibly) large input in a way that is fundamentally difficult to reverse (i.e., it is hard to find an input that will produce a given output). Hash functions are used as building blocks for key management, for example,

1. To provide data authentication and integrity services (Section 4.2.3) – the hash function is used with a key to generate a message authentication code;
2. To compress messages for digital signature generation and verification (Section 4.2.4);
3. To derive keys in key-establishment algorithms (Section 4.2.5); and
4. To generate deterministic random numbers (Section 4.2.7).

Symmetric-key algorithms

Symmetric-key algorithms (sometimes known as secret-key algorithms) transform data in a way that is fundamentally difficult to undo without knowledge of a secret key. The key is “symmetric” because the same key is used for a cryptographic operation and its inverse (e.g., encryption and decryption). Symmetric keys are often known by more than one entity; however, the key shall not be disclosed to entities that are not authorized access to the data protected by that algorithm and key. Symmetric key algorithms are used, for example,

1. To provide data confidentiality (Section 4.2.2); the same key is used to encrypt and decrypt data;
2. To provide authentication and integrity services (Section 4.2.3) in the form of Message Authentication Codes (MACs); the same key is used to generate the MAC and to validate it. MACs normally employ either a symmetric key-encryption algorithm or a cryptographic hash function as their cryptographic primitive;
3. As part of the key-establishment process (Section 4.2.5); and
4. To generate deterministic random numbers (Section 4.2.7).

Asymmetric-key algorithms

Asymmetric-key algorithms, commonly known as public-key algorithms, use two related keys (i.e., a key pair) to perform their functions: a public key and a private key. The public key may be known by anyone; the private key should1 be under the sole control of the entity that “owns” the key pair. Even though the public and private keys of a key pair are related, knowledge of the public key does not reveal the private key. Asymmetric algorithms are used, for example,

1. To compute digital signatures (Section 4.2.4);
2. To establish cryptographic keying material (Section 4.2.5); and
3. To generate random numbers (Section 4.2.7).

Message Authentication Codes (MACs)

Message Authentication Codes (MACs) provide data authentication and integrity. A MAC is a cryptographic checksum on the data that is used in order to provide assurance that the data has not changed and that the MAC was computed by the expected entity. Although message integrity is often provided using non-cryptographic techniques known as error detection codes, these codes can be altered by an adversary to effect an action to the adversary’ s benefit. The use of an approved cryptographic mechanism, such as a MAC, can alleviate this problem. In addition, the MAC can provide a recipient with assurance that the originator of the data is a key holder (i.e., an entity authorized to have the key). MACs are often used to authenticate the originator to the recipient when only those two parties share the MAC key.

Digital Signatures

Digital signatures are used to provide authentication, integrity and non-repudiation. Digital signatures are used in conjunction with hash functions and are computed on data of any length (up to a limit that is determined by the hash function). FIPS186 specifies algorithms that are approved for the computation of digital signatures.

Key Encryption Keys

Symmetric key-wrapping keys are used to encrypt other keys using symmetric-key algorithms. Key-wrapping keys are also known as key encrypting keys.

Key Strength

Review NIST SP 800-57 (Recommendation for Key Management) for recommended guidelines on key strength for specific algorithm implementations. Also, consider these best practices:

1. Establish what the application's minimum computational resistance to attack should be. Understanding the minimum computational resistance to attack should take into consideration the sophistication of your adversaries, how long data needs to be protected, where data is stored and if it is exposed. Identifying the computational resistance to attack will inform engineers as to the minimum length of the cryptographic key required to protect data over the life of that data. Consult NIST SP 800-131a for additional guidance on determining the appropriate key lengths for the algorithm of choice.
2. When encrypting keys for storage or distribution, always encrypt a cryptographic key with another key of equal or greater cryptographic strength.
3. When moving to Elliptic Curve-based algorithms, choose a key length that meets or exceeds the comparative strength of other algorithms in use within your system. Refer to NIST SP 800-57 Table 2.
4. Formulate a strategy for the overall organization's cryptographic strategy to guide developers working on different applications and ensure that each application's cryptographic capability meets minimum requirements and best practices.

Memory Management Considerations

Keys stored in memory for a long time can become “burned in”. This can be mitigated by splitting the key into components that are frequently updated. NIST SP 800.57). Loss or corruption of the memory media on which keys and/or certificates are stored, and recovery planning, according to NIST SP 800.57. Plan for the recovery from possible corruption of the memory media necessary for key or certificate generation, registration, and/or distribution systems, subsystems, or components as recommended in NIST SP 800.57.

Perfect Forward Secrecy

Ephemeral keys can provide perfect forward secrecy protection, which means a compromise of the server's long term signing key does not compromise the confidentiality of past sessions. Refer to OWASP Transport Layer Protection Cheat Sheet (https://www.owasp.org/index.php/Transport_Layer_Protection_Cheat_Sheet).

Proxy Handling

Key Usage

According to NIST, in general, a single key should be used for only one purpose (e.g., encryption, authentication, key wrapping, random number generation, or digital signatures). There are several reasons for this:

1. The use of the same key for two different cryptographic processes may weaken the security provided by one or both of the processes.
2. Limiting the use of a key limits the damage that could be done if the key is compromised.
3. Some uses of keys interfere with each other. For example, the length of time the key may be required for each use and purpose. Retention requirements of the data may differ for different data types.

Cryptographic Module Topics

According to NIST SP800-133, cryptographic modules are the set of hardware, software, and/or firmware that implements security functions (including cryptographic algorithms and key generation) and is contained within a cryptographic module boundary to provide protection of the keys.

Key Management Lifecycle Best Practices

Generation

Cryptographic keys shall be generated within cryptographic module with at least a FIPS 140-2 compliance. For explanatory purposes, consider the cryptographic module in which a key is generated to be the key-generating module. Any random value required by the key-generating module shall be generated within that module; that is, the Random Bit Generator that generates the random value shall be implemented within cryptographic module with at least a FIPS 140-2 compliance that generates the key. Hardware cryptographic modules are preferred over software cryptographic modules for protection.

Distribution

The generated keys shall be transported (when necessary) using secure channels and shall be used by their associated cryptographic algorithm within at least a FIPS 140-2 compliant cryptographic modules. For additional detail for the recommendations in this section refer to NIST Special Paper 800-133.

Storage

1. Developers must understand where cryptographic keys are stored within the application. Understand what memory devices the keys are stored on.
2. Keys must be protected on both volatile and persistent memory, ideally processed within secure cryptographic modules.
3. Keys should never be stored in plaintext format.
4. Ensure all keys are stored in cryptographic vault, such as a hardware security module (HSM) or isolated cryptographic service.
5. If you are planning on storing keys in offline devices/databases, then encrypt the keys using Key Encryption Keys (KEKs) prior to the export of the key material. KEK length (and algorithm) should be equivalent to or greater in strength than the keys being protected.
6. Ensure that keys have integrity protections applied while in storage (consider dual purpose algorithms that support encryption and Message Code Authentication (MAC)).
7. Ensure that standard application level code never reads or uses cryptographic keys in any way and use key management libraries.
8. Ensure that keys and cryptographic operation is done inside the sealed vault.
9. All work should be done in the vault (such as key access, encryption, decryption, signing, etc).

Escrow and Backup

Data that has been encrypted with lost cryptographic keys will never be recovered. Therefore, it is essential that the application incorporate a secure key backup capability, especially for applications that support data at rest encryption for long-term data stores. When backing up keys, ensure that the database that is used to store the keys is encrypted using at least a FIPS 140-2 validated module. It is sometimes useful to escrow key material for use in investigations and for re-provisioning of key material to users in the event that the key is lost or corrupted. Never escrow keys used for performing digital signatures, but consider the need to escrow keys that support encryption. Oftentimes, escrow can be performed by the Certificate Authority (CA) or key management system that provisions certificates and keys, however in some instances separate APIs must be implemented to allow the system to perform the escrow for the application.

Key Rotation

Accountability and Audit

Accountability involves the identification of those that have access to, or control of, cryptographic keys throughout their lifecycles. Accountability can be an effective tool to help prevent key compromises and to reduce the impact of compromises once they are detected. Although it is preferred that no humans are able to view keys, as a minimum, the key management system should account for all individuals who are able to view plaintext cryptographic keys. In addition, more sophisticated key-management systems may account for all individuals authorized to access or control any cryptographic keys, whether in plaintext or ciphertext form. Accountability provides three significant advantages:

1. It aids in the determination of when the compromise could have occurred and what individuals could have been involved,
2. It tends to protect against compromise, because individuals with access to the key know that their access to the key is known, and
3. It is very useful in recovering from a detected key compromise to know where the key was used and what data or other keys were protected by the compromised key.

Certain principles have been found to be useful in enforcing the accountability of cryptographic keys. These principles might not apply to all systems or all types of keys. Some of the principles that apply to long-term keys controlled by humans include:

1. Uniquely identifying keys,
2. Identifying the key user,
3. Identifying the dates and times of key use, along with the data that is protected, and
4. Identifying other keys that are protected by a symmetric or private key.

Two types of audit should be performed on key management systems:

1. The security plan and the procedures that are developed to support the plan should be periodically audited to ensure that they continue to support the Key Management Policy (NIST SP 800-57 Part 2).
2. The protective mechanisms employed should be periodically reassessed with respect to the level of security that they provide and are expected to provide in the future, and that the mechanisms correctly and effectively support the appropriate policies.

New technology developments and attacks should be taken into consideration. On a more frequent basis, the actions of the humans that use, operate and maintain the system should be reviewed to verify that the humans continue to follow established security procedures. Strong cryptographic systems can be compromised by lax and inappropriate human actions. Highly unusual events should be noted and reviewed as possible indicators of attempted attacks on the system.

Key Compromise and Recovery

The compromise of a key has the following implications:

1. In general, the unauthorized disclosure of a key used to provide confidentiality protection (i.e., via encryption) means that all information encrypted by that key could be expose or

known by unauthorized entities. The disclosure of a Certificate of Authorities' s private signature key means that an adversary can create fraudulent certificates and Certificate Revocation Lists (CRLs).
2. A compromise of the integrity of a key means that the key is incorrect - either that the key has been modified (either deliberately or accidentally), or that another key has been substituted; this includes a deletion (non-availability) of the key. The substitution or modification of a key used to provide integrity calls into question the integrity of all information protected by the key. This information could have been provided by, or changed by, an unauthorized entity that knows the key. The substitution of a public or secret key that will be used (at a later time) to encrypt data could allow an unauthorized entity (who knows the decryption key) to decrypt data that was encrypted using the encryption key.
3. A compromise of a key' s usage or application association means that the key could be used for the wrong purpose (e.g., for key establishment instead of digital signatures) or for the wrong application, and could result in the compromise of information protected by the key.
4. A compromise of a key' s association with the owner or other entity means that the identity of the other entity cannot be assured (i.e., one does not know who the other entity really is) or that information cannot be processed correctly (e.g., decrypted with the correct key).
5. A compromise of a key' s association with other information means that there is no association at all, or the association is with the wrong "information". This could cause the cryptographic services to fail, information to be lost, or the security of the information to be compromised. Certain protective measures may be taken in order to minimize the likelihood or consequences of a key compromise. Similar affect as ransomware, except that you can' t pay the ransom and get the key back.

The following procedures are usually involved:

1. Limiting the amount of time a symmetric or private key is in plaintext form.
2. Preventing humans from viewing plaintext symmetric and private keys.
3. Restricting plaintext symmetric and private keys to physically protected containers. This includes key generators, key-transport devices, key loaders, cryptographic modules, and key-storage devices.
4. Using integrity checks to ensure that the integrity of a key or its association with other data has not been compromised. For example, keys may be wrapped (i.e., encrypted) in such a manner that unauthorized modifications to the wrapping or to the associations will be detected.
5. Employing key confirmation (see NIST SP 800-57 Part 1 Section 4.2.5.5) to help ensure that the proper key was, in fact, established.
6. Establishing an accountability system that keeps track of each access to symmetric and private keys in plaintext form.
7. Providing a cryptographic integrity check on the key (e.g., using a MAC or a digital signature).
8. The use of trusted timestamps for signed data. i. Destroying keys as soon as they are no longer needed.
9. Creating a compromise-recovery plan, especially in the case of a CA compromise.

A compromise-recovery plan is essential for restoring cryptographic security services in the event of a key compromise. A compromise-recovery plan shall be documented and easily accessible. The compromise-recovery plan should contain:

1. The identification and contact info of the personnel to notify,
2. The identification and contact info of the personnel to perform the recovery actions,
3. The re-key method,
4. An inventory of all cryptographic keys and their use (e.g., the location of all certificates in a system),
5. The education of all appropriate personnel on the recovery procedures,
6. An identification and contact info of all personnel needed to support the recovery procedures,
7. Policies that key-revocation checking be enforced (to minimize the effect of a compromise),
8. The monitoring of the re-keying operations (to ensure that all required operations are performed for all affected keys), and
9. Any other recovery procedures, which may include:
    1. Physical inspection of the equipment,
    2. Identification of all information that may be compromised as a result of the incident,
    3. Identification of all signatures that may be invalid, due to the compromise of a signing key, and
    4. Distribution of new keying material, if required.

Trust Stores

1. Design controls to secure the trust store against injection of 3rd party root certificates. The access controls are managed and enforced on an entity and application basis.
2. Implement integrity controls on objects stored in the trust store.
3. Do not allow for export of keys held within the trust store without authentication and authorization.
4. Setup strict policies and procedures for exporting key material from applications to network applications and other components.
5. Implement a secure process for updating the trust store.

Cryptographic Key Management Libraries

Use only reputable crypto libraries that are well maintained and updated, as well as tested and validated by 3rd party organizations (e.g., NIST/FIPS)

Authors and Primary Editors

Brian Russell - russellbri[at]leidos.com
Drew Van Duren - drew.f.van.duren[at]leidos.com
Vanessa Amador - vanessa.c.amador[at]leidos.com
Susanna Bezold – BezoldCISSP[at]aol.com

Other Cheatsheets

OWASP Cheat Sheets Project Homepage

- OWASP Cheat Sheet Series

V - T - E (https://www.owasp.org/index.php?title=Key_Management_Cheat_Sheet&action=edit) **Cheat Sheets** [Collapse]

Developer / Builder	3rd Party Javascript Management • Access Control • AJAX Security Cheat Sheet • Authentication (ES) • Bean Validation Cheat Sheet • Choosing and Using Security Questions • Clickjacking Defense • Credential Stuffing Prevention Cheat Sheet • Cross-Site Request Forgery (CSRF) Prevention • Cryptographic Storage • C-Based Toolchain Hardening • Deserialization • DOM based XSS Prevention • Forgot Password • HTML5 Security • HTTP Strict Transport Security • Injection Prevention Cheat Sheet • Injection Prevention Cheat Sheet in Java • JSON Web Token (JWT) Cheat Sheet for Java • Input Validation • Insecure Direct Object Reference Prevention • JAAS • **Key Management** • LDAP Injection Prevention • Logging • Mass Assignment Cheat Sheet • .NET Security • OS Command Injection Defense Cheat Sheet • OWASP Top Ten • Password Storage • Pinning • Query Parameterization • REST Security • Ruby on Rails • Session Management • SAML Security • SQL Injection Prevention • Transaction Authorization • Transport Layer Protection • Unvalidated Redirects and Forwards • User Privacy Protection • Web Service Security • XSS (Cross Site Scripting) Prevention • XML External Entity (XXE) Prevention Cheat Sheet
Assessment / Breaker	Attack Surface Analysis • REST Assessment • Web Application Security Testing • XML Security Cheat Sheet • XSS Filter Evasion
Mobile	Android Testing • IOS Developer • Mobile Jailbreaking
OpSec / Defender	Virtual Patching • Vulnerability Disclosure
Draft and Beta	Application Security Architecture • Business Logic Security • Content Security Policy • Denial of Service Cheat Sheet • Grails Secure Code Review • IOS Application Security Testing • PHP Security • Regular Expression Security Cheatsheet • Secure Coding • Secure SDLC • Threat Modeling

All Pages In This Category

This project is part of the OWASP Builders community.
Feel free to browse other projects within the Defenders, Builders, and Breakers communities.

Retrieved from "https://www.owasp.org/index.php?title=Key_Management_Cheat_Sheet&oldid=241393"

Categories: Cheatsheets | OWASP Builders

- This page was last modified on 18 June 2018, at 20:53.
-