Cryptographic Storage Cheat Sheet

From OWASP

Contents

Introduction

This article provides a simple model to follow when implementing solutions for data at rest.

Architectural Decision

An architectural decision must be made to determine the appropriate method to protect data at rest. There are such wide variety of products, methods and mechanisms for cryptographic storage that this cheat sheet will only focus on low-level guidelines for developers and architects who are implementing cryptographic solutions. We will not address specific vendor solutions, nor will we address the design of cryptographic algorithms.

Providing Cryptographic Functionality

Benefits

Basic Requirements

Secure Cryptographic Storage Design

Rule - Only store sensitive data that you need

Many eCommerce businesses use payment providers that store the credit card for recurring billing. This offloads the burden of keeping credit card numbers safe.

Rule - Only use strong cryptographic algorithms

Only use approved public algorithms such as AES, RSA public key

cryptography, and SHA-256 or better for hashing. Do not use weak algorithms, such as MD5 / SHA1/ Favor safer. The definition of a "strong" cryptographic algorithms change over time.

http://csrc.nist.gov/groups/STM/cavp/index.html

This is a good default if one doesn't have AES and one of the authenticated encryption modes that provide confidentiality and authenticity (i.e., data origin authentication) such as CCM, EAX, CMAC, etc. For Java, if you are using SunJCE that will be the case. The cipher modes supported in JDK 1.5 and later are CBC, CFB, CFBx, CTR, CTS, ECB, OFB, OFBx, PCBC, None of these cipher modes are authentication encryption modes. (That's why I added it explicitly.) If you are using an alternate JCE provider such as Bouncy Castle, RSA JSafe, IAIK, etc then some of these authentication encryption modes probably should be preferred.

Use salts when appropriate. Note that integrity should use a MAC such as HMAC-SHA1 or even better, HMAC-SHA256.

Rule - Ensure that random numbers are cryptographically strong

Ensure that all random numbers, random file names, random GUIDs, and random strings are generated in a cryptographically strong fashion. Also ensure that random algorithms are seeded with sufficient entropy.

Rule - Only use widely accepted implementations of cryptographic algorithms

Do not implement an existing cryptographic algorithm on your own, no matter how easy it appears. Use widely accepted algorithms and widely accepted implementations only. Ensure that an implementation has, at least, had some cryptography experts involved in its creation. If possible, use an implementation that is FIPS 140-2 certified.

Rule - Prefer authenticated encryption modes

Use cryptographic cipher modes that offer both confidentiality and authenticity. Recommended modes include counter with CBC-MAC (CCM) mode and Galois/counter mode (GCM). These authenticated encryption (AE) modes resist padding oracle attacks which can be used in many cases to recover plaintext from ciphertext without knowledge of the encryption key.

If an authenticated encryption mode is not available, then the best option is to consult with a professional cryptographer. Ask the cryptographer to review and customize the system to counter padding oracle attacks. This customization will

typically involve the use of a message authentication code (MAC) that must be carefully employed in order to be effective.

In the worst case, where neither an AE mode nor a professional cryptographer is available, then the cryptography is definitely vulnerable to a padding oracle attack. The best that can be done in this situation is to design the overall system so that the decryption function is only given ciphertext retrieved directly from a canonical source which must itself be protected by other integrity controls. While this technique will help reduce the risk of a padding oracle attack, it does not eliminate the risk, and it should be treated as a temporary workaround until an AE-based solution can be implemented.

When selecting a cryptographic mode for this worse case, cipher-block chaining (CBC) mode is a strong choice as it has been carefully studied, is well understood, and is readily available in many cryptographic APIs. In all cases avoid the electronic codebook (ECB) mode. If using cipher modes that create streaming ciphers from block ciphers, such as the output feedback (OFB) or cipher feedback (CFB) modes, be careful not to repeat the IV for the same encryption key.

See Authenticated Encryption (http://www.cryptopp.com /wiki/Authenticated_Encryption) on the Crypto++ wiki for a more complete discussion of the technique. NIST's CRSC Current Modes (http://csrc.nist.gov /groups/ST/toolkit/BCM/current_modes.html) and Modes Development (http://csrc.nist.gov/groups/ST/toolkit/BCM/modes_development.html) documents offer an abbreviated list of cipher modes. Additional block cipher modes can be found in Wikipedia's Block cipher modes of operation (http://en.wikipedia.org/wiki/Block_cipher_modes_of_operation) article.

Rule - Store the hashed and salted value of passwords

Store the salted hashed value of the password. Salt each hash. Use a different random salt for each password hash. Never store the clear text password or an encrypted version of the password.

Cryptographic framework for password hashing is described in PKCS #5 v2.1: Password-Based Cryptography Standard (http://www.rsa.com/rsalabs /node.asp?id=2127) . Specific secure password hashing algorithms exist such as bcrypt (http://www.usenix.org/events/usenix99/provos/provos_html/node1.html) , scrypt (http://www.tarsnap.com/scrypt/scrypt.pdf) . Implementations of secure password hashing exist for PHP (phpass (http://www.openwall.com/phpass/)), ASP.NET (ASP.NET 2.0 Security Practices (http://msdn.microsoft.com/en-us /library/ms998372.aspx#pagpractices0001_sensitivedata)), Java (OWASP Hashing Java (http://www.owasp.org/index.php/Hashing_Java)).

Rule - Ensure that the cryptographic protection remains secure even if access controls fail

This rule supports the principle of defense in depth. Access controls (usernames, passwords, privileges, etc.) are one layer of protection. Storage encryption should add an additional layer of protection that will continue protecting the data even if an attacker subverts the database access control layer.

Rule - Ensure that any secret key is protected from unauthorized access

Rule - Define a key lifecycle

The key lifecycle details the various states that a key will move through during its life. The lifecycle will specify when a key should no longer be used for encryption, when a key should no longer be used for decryption (these are not necessarily coincident), whether data must be rekeyed when a new key is introduced, and when a key should be removed from use all together.

Rule - Store unencrypted keys away from the encrypted data

If the keys are stored with the data then any compromise of the data will easily compromise the keys as well. Unencrypted keys should never reside on the same machine or cluster as the data.

Rule - Use independent keys when multiple keys are required

Ensure that key material is independent. That is, do not choose a second key which is easily related to the first (or any preceeding) keys.

Rule - Protect keys in a key vault

Keys should remain in a protected key vault at all times. In particular, ensure that there is a gap between the threat vectors with direct access to the data and the threat vectors with direct access to the keys. This implies that keys should not be stored on the application or web server (assuming that application attackers are part of the relevant threat model).

Rule - Document concrete procedures for managing keys through the lifecycle

These procedures must be written down and the key custodians must be adequately trained.

Rule - Build support for changing keys periodically

Key rotation is a must as all good keys do come to an end either through expiration or revocation. So a developer will have to deal with rotating keys at some point -- better to have a system in place now rather than scrambling later. (From Bil Cory as a starting point).

Rule - Document concrete procedures to handle a key compromise

Rule - Rekey data at least every one to three years

Rekeying refers to the process of decrypting data and then re-encrypting it with a new key. Periodically rekeying data helps protect it from undetected compromises of older keys. The appropriate rekeying period depends on the security of the keys. Data protected by keys secured in dedicated hardware security modules might only need rekeying every three years. Data protected by keys that are split and stored on two application servers might need rekeying every year.

Rule - Follow applicable regulations on use of cryptography

Rule - Under PCI DSS requirement 3, you must protect cardholder data

The Payment Card Industry (PCI) Data Security Standard (DSS) was developed to encourage and enhance cardholder data security and facilitate the broad adoption of consistent data security measures globally. The standard was introduced in 2005 and replaced individual compliance standards from Visa, Mastercard, Amex, JCB and Diners. The current version of the standard was introduced in 2008 and an update to the standard is expected in 2010.

PCI DSS requirement 3 covers secure storage of credit card data. This requirement covers several aspects of secure storage including the data you must never store but we are covering Cryptographic Storage which is covered in requirements 3.4, 3.5 and 3.6 as you can see below:

3.4 Render PAN (Primary Account Number), at minimum, unreadable anywhere it is stored

Compliance with requirement 3.4 can be met by implementing any of the four types of secure storage described in the standard which includes encrypting and hashing data. These two approaches will often be the most popular choices

from the list of options. The standard doesn't refer to any specific algorithms but it mandates the use of **Strong Cryptography**. The glossary document from the PCI council defines **Strong Cryptography** as:

Cryptography based on industry-tested and accepted algorithms, along with strong key lengths and proper key-management practices. Cryptography is a method to protect data and includes both encryption (which is reversible) and hashing (which is not reversible, or "one way"). SHA-1 is an example of an industry-tested and accepted hashing algorithm. Examples of industry-tested and accepted standards and algorithms for encryption include AES (128 bits and higher), TDES (minimum double-length keys), RSA (1024 bits and higher), ECC (160 bits and higher), and ElGamal (1024 bits and higher).

If you have implemented the second rule in this cheat sheet you will have implemented a strong cryptographic algorithm which is compliant with or stronger than the requirements of PCI DSS requirement 3.4 You need to ensure that you identify all locations that card data could be stored including logs and apply the appropriate level of protection. This could range from encrypting the data to replacing the card number in logs.

This requirement can also be met by implementing disk encryption rather than file or column level encryption. The requirements for **Strong Cryptography** are the same for disk encryption and backup media. The card data should never be stored in the clear and by following the guidance in this cheat sheet you will be able to securely store your data in a manner which is compliant with PCI DSS requirement 3.4

3.5 Protect cryptographic keys used for encryption of cardholder data against both disclosure and misuse

As the requirement name above indicates we are required to securely store the encryption keys themselves. This will mean implementing strong access control, auditing and logging for your keys. The keys must be stored in a location which is both secure and "away" from the encrypted data, this means key data shouldn't be stored on web servers, database servers etc

Access to the keys must be restricted to the smallest amount of users possible. This group of users will ideally be users who are highly trusted and trained to perform Key Custodian duties. There will obviously be a requirement for system/service accounts to access the key data to perform encryption/decryption of data.

The keys themselves shouldn't be stored in the clear but encrypted with a KEK (Key Encrypting Key). The KEK must not be stored in the same location as the encryption keys it is encrypting.

3.6 Fully document and implement all key-management processes and procedures for cryptographic keys used for encryption of cardholder data

Requirement 3.6 mandates that key management processes within a PCI compliant company cover 8 specific key lifecycle steps:

3.6.1 Generation of strong cryptographic keys

As we have previously described in this cheat sheet we need to use algorithms which offer high levels of data security. We must also generate strong keys so that the security of the data isn't undermined by weak cryptographic keys. A strong key is generated by using a key length which is sufficient for your data security requirements and compliant with the PCI DSS. The key size alone isn't a measure of the strength of a key. The data used to generate the key must be sufficiently random ("sufficient" often being determined by your data security requirements) and the entropy of the key data itself must be high.

3.6.2 Secure cryptographic key distribution

The method used to distribute keys must be secure to prevent the theft of keys in transit. The use of a protocol such as Diffie Hellman can help secure the distribution of keys, the use of secure transport such as SSLv3, TLS and SSHv2 can also secure the keys in transit.

3.6.3 Secure cryptographic key storage

The secure storage of encryption keys including KEK's has been touched on in our description of requirement 3.5 (see above).

3.6.4 Periodic cryptographic key changes

The PCI DSS standard mandates that keys used for encryption must be rotated at least annually. The key rotation process must remove an old key from the encryption/decryption process and replace it with a new key. All new data entering the system must encrypted with the new key. While it is recommended that existing data be rekeyed with the new key, as per the Rekey data at least every one to three years rule above, it is not clear that the PCI DSS requires this.

3.6.5 Retirement or replacement of old or suspected compromised cryptographic keys

The key management processes must cater for archived, retired or compromised keys. The process of securely storing and replacing these keys will more than likely be covered by your processes for requirements 3.6.2, 3.6.3 and 3.6.4

3.6.6 Split knowledge and establishment of dual control of cryptographic

keys

The requirement for split knowledge and/or dual control for key management prevents an individual user performing key management tasks such as key rotation or deletion. The system should require two individual users to perform an action (i.e. entering a value from their own OTP) which creates to separate values which are concatenated to create the final key data.

3.6.7 Prevention of unauthorized substitution of cryptographic keys

The system put in place to comply with requirement 3.6.6 can go a long way to preventing unauthorised substitution of key data. In addition to the dual control process you should implement strong access control, auditing and logging for key data so that unauthorised access attempts are prevented and logged.

3.6.8 Requirement for cryptographic key custodians to sign a form stating that they understand and accept their key-custodian responsibilities

To perform the strong key management functions we have seen in requirement 3.6 we must have highly trusted and trained key custodians who understand how to perform key management duties. The key custodians must also sign a form stating they understand the responsibilities that come with this role.

Related Articles

OWASP - Testing for SSL-TLS, and OWASP Guide to Cryptography

OWASP – Application Security Verification Standard (ASVS) – Communication Security Verification Requirements (V10) (http://www.owasp.org/index.php /ASVS)

Other Articles in the OWASP Prevention Cheat Sheet Series

- Authentication Cheat Sheet
- Cross-Site Request Forgery (CSRF) Prevention Cheat Sheet
- Forgot Password Cheat Sheet
- **Cryptographic Storage Cheat Sheet**
- SQL Injection Prevention Cheat Sheet
- Transport Layer Protection Cheat Sheet
- XSS (Cross Site Scripting) Prevention Cheat Sheet
- DOM based XSS Prevention Cheat Sheet

Authors and Primary Editors

Kevin Kenan - kevin[at]k2dd.com

David Rook - david.a.rook[at]gmail.com

Kevin Wall - kevin.w.wall[at]gmail.com

Jim Manico - jim[at]manico.net