AWS cryptography services

AWS cryptographic services and tools guide

AWS cryptography services: AWS cryptographic services and tools guide

Copyright © 2020 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon's trademarks and trade dress may not be used in connection with any product or service that is not Amazon's, in any manner that is likely to cause confusion among customers, or in any manner that disparages or discredits Amazon. All other trademarks not owned by Amazon are the property of their respective owners, who may or may not be affiliated with, connected to, or sponsored by Amazon.

Table of Contents

AWS cryptographic services and tools 1
What is cryptography? 1
Cryptography concepts 1
Cryptographic algorithms 7
Cryptographic services and tools 9
AWS CloudHSM 9
AWS KMS 10
AWS Encryption SDK 11
DynamoDB Encryption Client 12
AWS Secrets Manager 12
Other AWS services 13
How to choose an encryption tool or service 13
When to use AWS KMS 14
When to use AWS CloudHSM 14
When to use AWS Encryption SDK 15
When to use DynamoDB Encryption Client 15
AWS PKI services 17
What is PKI? 17
PKI concepts 17
Available services (PKI) 21
AWS Certificate Manager 21
ACM Private CA 21
Other AWS services 22
How to choose a PKI service 22
When to use ACM 22
When to use ACM PCA 23
Document history 24

AWS cryptographic services and tools

AWS provides multiple services to help you protect your data at rest or in transit. This section provides an overview of cryptographic concepts and introduces the primary cryptographic services offered by AWS. For detailed explanations of individual services, see their respective documentation sets.

Topics

- What is cryptography? (p. 1)
- Cryptography concepts (p. 1)
- AWS cryptographic services and tools (p. 9)
- How to choose an encryption tool or service (p. 13)

See also: *AWS PKI services* (p. 17)

What is cryptography?

Cryptography is the practice of protecting information through the use of coded algorithms, hashes, and signatures. The information can be at rest such as a file on a hard drive. The information can also be in transit such as electronic communication exchanged between two or more parties. Cryptography has four primary goals:

- **Confidentiality** – Makes information available to only authorized users.
- **Data Integrity** – Ensures that information has not been manipulated.
- **Authentication** – Confirms the authenticity of information or the identity of a user.
- **Nonrepudiation** – Prevents a user from denying prior commitments or actions.

Cryptography uses a number of tools, typically called *primitives*, to provide information security. A *primitive* is a cryptographic algorithm. This includes encryption algorithms, digital signature algorithms, hashes, and other functions. AWS uses only well established, peer-reviewed primitives.

> **Note**
> Cryptography relies extensively on mathematics. This includes basic function theory, permutations, probability, information theory, complexity theory, number theory, and more. The math underlying cryptography is beyond the scope of this documentation, but printed and online sources are readily available.

To learn more about the terms and concepts used in cryptography, see Cryptography concepts (p. 1).

Cryptography concepts

As you work with cryptographic tools and services, you are likely to encounter a number of basic concepts.

Topics

- additional authenticated data (AAD) (p. 2)

- asymmetric and symmetric encryption (p. 2)
- authenticated encryption (p. 2)
- authentication (p. 3)
- block cipher (p. 3)
- ciphertext (p. 3)
- client-side and server-side encryption (p. 3)
- data key (p. 3)
- decryption (p. 4)
- encryption (p. 4)
- encryption algorithm (p. 4)
- encryption context (p. 4)
- envelope encryption (p. 5)
- hardware security module (HSM) (p. 6)
- key encryption key (p. 6)
- master key (p. 6)
- plaintext (p. 7)
- private key (p. 7)
- public key (p. 7)
- stream cipher (p. 7)

additional authenticated data (AAD)

Nonsecret data that is provided to encryption (p. 4) and decryption (p. 4) operations to add an additional integrity and authenticity check on the encrypted data. Typically, the decrypt operation fails if the AAD provided to the encrypt operation does not match the AAD provided to the decrypt operation.

AWS Key Management Service (p. 10) (AWS KMS) and the AWS Encryption SDK (p. 11) both support AAD by using an encryption context (p. 4).

See also: authenticated encryption (p. 2)

authenticated encryption

Authenticated encryption uses additional authenticated data (p. 2) (AAD) to provide confidentiality, data integrity, and authenticity assurances on encrypted data.

For example, the AWS Key Management Service (AWS KMS) `Encrypt` API and the encryption methods in the AWS Encryption SDK take an encryption context (p. 4) that represents additional authenticated data (AAD). The encryption context is cryptographically bound to the encrypted data so that the same encryption context is required to decrypt the data. To learn how to use encryption context to protect the integrity of encrypted data, see How to Protect the Integrity of Your Encrypted Data by Using AWS Key Management Service and EncryptionContext in the AWS Security Blog.

asymmetric and symmetric encryption

Symmetric encryption (p. 7) uses the same secret key to perform both the encryption (p. 4) and decryption (p. 4) processes.

Asymmetric encryption (p. 9), also known as *public-key encryption*, uses two keys, a public key (p. 7) for encryption and a corresponding private key (p. 7) for decryption. The public key and private key are mathematically related so that when the public key is used for encryption, the corresponding private key must be used for decryption. Encryption algorithms (p. 4) are either symmetric or asymmetric.

For more information, see Cryptographic algorithms (p. 7).

authentication

The process of verifying identity, that is, determining whether an entity is who it claims to be and that the authentication information has not been manipulated by unauthorized entities.

block cipher

An algorithm that operates on fixed-length blocks of data, one block at a time, rather than encrypting one bit at a time as in stream ciphers (p. 7).

ciphertext

The encrypted data. Ciphertext is typically the output of an encryption algorithm (p. 4) operating on plaintext (p. 7). Ciphertext is unreadable without knowledge of the algorithm and a secret key.

client-side and server-side encryption

Client-side encryption is encrypting data at or close to its source, such as encrypting data in the application or service that generates it.

Server-side encryption is encrypting data at its destination, that is, the application or service that receives it.

The method that you choose depends on the sensitivity of your data and the security requirements of your application. Client-side and server-side encryption differ in when, where, and who encrypts and decrypts the data. They do not necessarily define how the data is encrypted and might use the same process. In addition, they are not exclusive. You can often use client-side and server-side encryption on the same data.

AWS supports both client-side and server-side encryption. Most AWS services that store or manage customer data offer a server-side encryption option or perform server-side encryption of your data by default. These services transparently encrypt your data before writing it to disk and transparently decrypt it when you access it. Most AWS services that support server-side encryption are integrated with AWS Key Management Service (p. 10) (AWS KMS) to protect the encryption keys that protect your data. For a list of integrated services, see AWS Service Integration.

AWS also supports client-side encryption libraries, such as the AWS Encryption SDK (p. 11), the DynamoDB Encryption Client (p. 12), and Amazon S3 client-side encryption. For help choosing the library that best meets your needs, see the section called “How to choose a PKI service” (p. 22).

data key

In envelope encryption (p. 5), a *data key* or *data encryption key* is an encryption key that is used to protect data. Data keys differ from master keys (p. 6) and key encryption keys (p. 6), which are typically used to encrypt other encryption keys.

The term *data key* usually refers to how the key is used, not how it is constructed. Like all encryption keys, a data key is typically implemented as a byte array that meets the requirements of the encryption algorithm that uses it. As such, data keys can be used to encrypt data or other data keys.

Often a tool or service generates unique data key for each data element, such as a database item, email message, or other resource. Then, it encrypts all of the data keys under the same master key.

Several AWS tools and services provide data keys.

- The HSMs in a AWS CloudHSM (p. 9) cluster generate encryption keys that can be used as data keys, key encryption keys, or master keys.
- You can ask AWS Key Management Service (p. 10) (AWS KMS) to generate a data key. It returns a plaintext key and a copy of that key that is encrypted under the customer master keys that you specify.

decryption

The process of turning ciphertext (p. 3) back into plaintext (p. 7). Decryption algorithms typically require an encryption key and can require other inputs, such as initialization vectors (IVs) and additional authenticated data (AAD) (p. 2).

encryption

The process of converting plaintext (p. 7) readable data to an unreadable form, known as ciphertext (p. 3), to protect it. The formula used to encrypt the data, known as an encryption algorithm (p. 4), must be almost impossible (using current and anticipated technology) to reverse without knowledge of the inputs to the algorithm. These inputs can include an encryption key and other random and determined data.

All of the cryptographic services and tools (p. 9) that AWS supports provide methods for you to encrypt and decrypt your data. Other AWS services automatically and transparently encrypt the data that they store and manage for you.

encryption algorithm

A procedure or ordered set of instructions that specifies precisely how plaintext (p. 7) data is transformed into encrypted data or ciphertext (p. 3). The input to an encryption algorithms includes the plaintext data and a encryption key. The output includes the ciphertext.

For example, AWS Key Management Service (p. 10) (AWS KMS) uses the Advanced Encryption Standard (AES) symmetric (p. 2) algorithm in Galois/Counter Mode (GCM), known as AES-GCM. AWS CloudHSM (p. 9) supports keys for multiple encryption algorithms.

encryption context

A type of additional authenticated data (AAD) (p. 2). It typically consists of nonsecret, arbitrary, name–value pairs. In most cases, you can provide an encryption context when you encrypt data. The same encryption context must be provided to decrypt the data. The encryption context is usually optional but recommended.

The term *encryption context* has different meanings in various AWS services and tools. This can be confusing, so be sure to understand how your tool or service interprets this term.

The following tools and services support an encryption context.

- In AWS Key Management Service (p. 10) (AWS KMS), an encryption context is a collection of nonsecret name–value pairs. When you provide an encryption context to an encryption (p. 4) operation, AWS KMS binds it cryptographically to the ciphertext (p. 3). To decrypt the data, you must provide an exact, case-sensitive match for the encryption context.

  AWS KMS includes the encryption context in AWS CloudTrail logs of cryptographic operations. As such, you can use a well-designed encryption context to help you track and audit the use of your encryption keys for particular projects or types of data.

  AWS KMS also lets you use all or part of the encryption context as the condition for a permission in a policy or grant. For example, you can allow a user to use a master key to decrypt data only when the encryption context includes a particular value.

  For details, see Encryption Context in the AWS Key Management Service Developer Guide.

- The AWS Encryption SDK (p. 11) also supports an optional encryption context in all cryptographic operations.

  However, you do not provide the encryption context to the decryption (p. 4) operation. Instead, when it encrypts data, the SDK saves the encryption context (in plaintext (p. 7)) along with the ciphertext in the encrypted message that it returns. When you ask the SDK to decrypt the encrypted message, the SDK uses the encryption context that it saved.

You can still use the encryption context to provide an additional verification of your data. When you decrypt data, you can get and examine the encryption context and return the decrypted data only after verifying that the encryption context has the expected value.

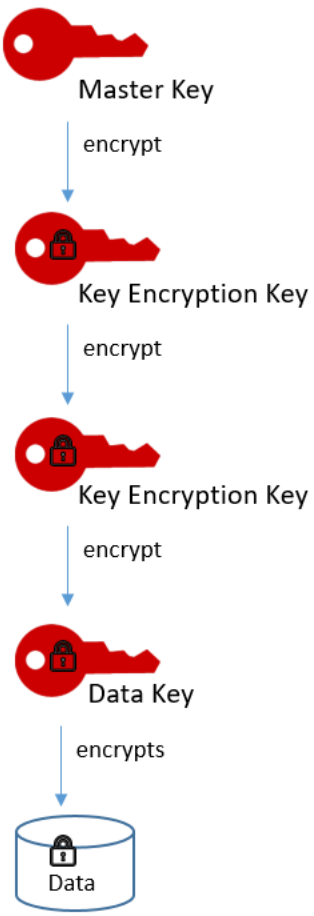
- The DynamoDB Encryption Client (p. 12) uses *encryption context* to mean something different from its use in AWS KMS or the AWS Encryption SDK. The *DynamoDB encryption context* is a collection of information about the table and table item that you pass to a cryptographic materials provider (CMP). It is not related to AAD.

envelope encryption

A strategy for protecting the encryption keys that you use to encrypt your data. First, you encrypt plaintext (p. 7) data with a data key (p. 3). Then, to protect the data key, you encrypt it under another key, known as a key encryption key (p. 6).

Encrypting the data key is more efficient than reencrypting the data under the new key because it is quicker and produces a much smaller ciphertext (p. 3).

You can even encrypt the data encryption key under another encryption key and encrypt that encryption key under still another encryption key. But, eventually, one key must remain in plaintext so you can decrypt the keys and your data. This top-level plaintext key encryption key is known as the master key (p. 6), as shown in the following diagram.



Several AWS cryptographic tools and services (p. 9) support envelope encryption. AWS Key Management Service (p. 10) (AWS KMS) protects the master key that must remain in plaintext. It supplies master keys that never leave the service unencrypted. AWS KMS supports operations that generate data keys that are encrypted under your master key. You can use the data keys to encrypt your data outside of AWS KMS.

The AWS Encryption SDK (p. 11) automatically encrypts your data with a data key that is encrypted by a master key that you specify. The DynamoDB Encryption Client (p. 12) supports many encryption (p. 4) strategies, including envelope encryption with an AWS KMS customer master key or with keys that you provide.

hardware security module (HSM)

A computing device that performs cryptographic operations and provides secure storage for cryptographic keys. Many HSMs have features that make them resistant to tampering or provide reliable tamper detection.

AWS CloudHSM (p. 9) lets you create, manage, and control your own HSMs in the cloud. AWS Key Management Service (p. 10) (AWS KMS) generates and protects the customer master keys (CMKs) that it provides in FIPS 140-2 validated HSMs that it manages for you. AWS KMS also lets you create your CMKs in a custom key store backed by an AWS CloudHSM cluster that you own and manage.

key encryption key

In envelope encryption (p. 5), a *key encryption key* is an encryption key that is used to encrypt a data key (p. 3) or another key encryption key. To protect the key encryption key, it is encrypted by using a master key (p. 6).

The term *key encryption key* refers to how the key is used, not how it is constructed. Like all encryption keys, a key encryption key is typically implemented as a byte array that meets the requirements of the encryption algorithm (p. 4) that uses it.

Several AWS services provide key encryption keys.

- The HSMs in a AWS CloudHSM (p. 9) cluster generate encryption keys that can be used as data keys, key encryption keys, or master keys.
- You can ask AWS Key Management Service (p. 10) (AWS KMS) to generate a data key, then use that key as a key encryption key outside of AWS KMS.

master key

In envelope encryption (p. 5), a master key is an encryption key that is used to encrypt other encryption keys, such as data keys (p. 3) and key encryption keys (p. 6). Unlike data keys and key encryption keys, master keys must be kept in plaintext (p. 7) so they can be used to decrypt the keys that they encrypted.

The term *master key* usually refers to how the key is used, not how it is constructed. Like all encryption keys, a master key is typically implemented as a byte array that meets the requirements of the encryption algorithm (p. 4) that uses it.

AWS Key Management Service (p. 10) (AWS KMS) generates and protect master keys. Its customer master keys (CMKs) are created, managed, used, and deleted entirely within AWS KMS.

Several AWS services provide master keys.

- The HSMs in a AWS CloudHSM (p. 9) cluster generate encryption keys that can be used as data keys, key encryption keys, or master keys.
- AWS Key Management Service (p. 10) (AWS KMS) generates and protects master keys. Its customer master keys (CMKs) are created, managed, used, and deleted entirely within AWS KMS.

plaintext

Information or data in an unencrypted, unprotected, or human-readable form.

See also: ciphertext (p. 3).

private key

One of two keys, along with public keys (p. 7), used to protect data in an asymmetric encryption (p. 2) scheme. Public and private keys are algorithmically generated in tandem: the public key is distributed to multiple trusted entities, and one of its paired private keys is distributed to a single entity. This way, a message can be *authenticated* because the public key signature proves that a trusted entity encrypted and sent it. The message contents can also be *secured* so that only a private key holder can decrypt it.

public key

One of two keys, along with private keys (p. 7), used to protect data in an asymmetric encryption (p. 2) scheme. Public and private keys are algorithmically generated in tandem: the public key is distributed to multiple trusted entities, and one of its paired private keys is distributed to a single entity. This way, a message can be *authenticated* because the public key signature proves that a trusted entity encrypted and sent it. The message contents can also be *secured* so that only a private key holder can decrypt it.

stream cipher

An algorithm that operates one bit of a data at a time rather than encrypting one block of data at a time as in block ciphers (p. 3).

Cryptographic algorithms

An *encryption algorithm* is a formula or instructions series that converts a plaintext readable message into an unreadable ciphertext. Algorithms use advanced mathematics and one or more encryption keys to make it relatively easy to encode a message but virtually impossible to decode without knowing the keys. Algorithms often involve multiple layers of encryption and can require additional random and sequential one-time values that prevent attacks and duplicate results.

AWS cryptography tools and services that encrypt data support secure algorithms that are publicly vetted. Some tools and services, like AWS Key Management Service (AWS KMS) use a particular algorithm. AWS KMS uses the Advanced Encryption Standard (AES) algorithm in Galois/Counter Mode (GCM) with 256-bit secret keys. Other tools and services offer multiple algorithms and key sizes but recommend a secure default choice.

This section describes some of the symmetric and asymmetric algorithms that AWS tools and services support.

Topics

- Symmetric algorithms (p. 7)
- Asymmetric algorithms (p. 9)

Symmetric algorithms

AWS cryptographic tools and services commonly support two widely used symmetric algorithms. Both are known as block ciphers.

- **AES** – Advanced Encryption Standard (AES) with 128-, 192-, or 256-bit keys. AES is often combined with Galois/Counter Mode (GCM) and known as AES-GCM.

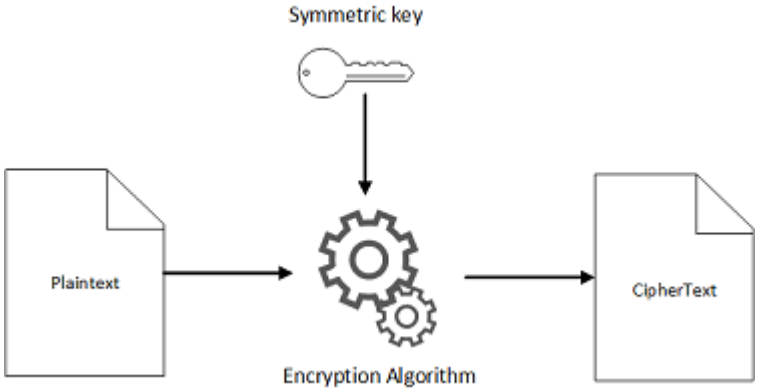
- **Triple DES** – Triple DES (3DES) uses three 56-bit keys. The scheme works on a block of data by splitting it in two and iteratively applying arbitrary round functions derived from an initial function. Triple DES uses 48 rounds to encrypt a block of data.

An encryption scheme is called *symmetric* if it uses the same key to both encrypt and decrypt plaintext. Technically, the encryption key *e* and decryption key *d* don't have to be exactly the same. All that's required is that it's computationally trivial to determine *d* when you know *e* and *e* when you know *d*. However, in most practical symmetric encryption schemes, *e* and *d* are the same.

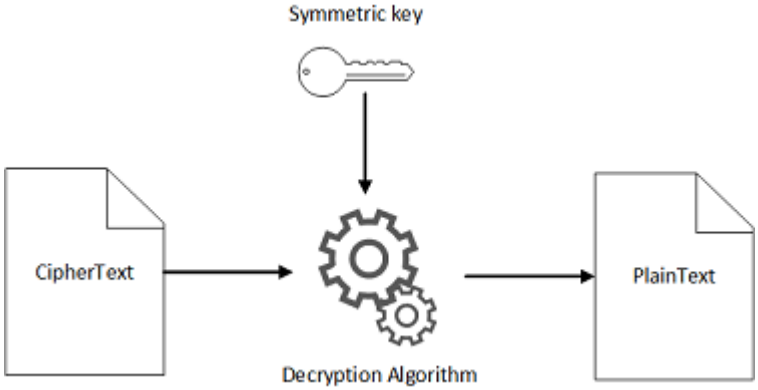
> **Note**
> Symmetric encryption is also called *shared key*, *shared secret*, and *secret key* encryption. It is not called *private key* encryption. Convention reserves the term *private key* for asymmetric cryptography, which centers around the idea of a private key and a corresponding public key.

Symmetric key encryption requires that all intended message recipients have access to the shared key. Therefore, a secure communication channel must be established among the participants so that the key can be transmitted to each along with the ciphertext. This presents practical problems and limits the use of direct symmetric key exchange.

The following illustrations show how encryption and decryption work with symmetric keys and algorithms. In the first illustration, a symmetric key and algorithm are used to convert a plaintext message into ciphertext.



The following illustration shows the same secret key and symmetric algorithm being used to turn ciphertext back into plaintext.



There are two types of symmetric key ciphers, block ciphers and stream ciphers. A **block** cipher divides the plaintext messaged into fixed-length strings called blocks and encrypts one block at a time. Block ciphers are typically considered to be more powerful and practical primitives than stream ciphers,

but they're also slower. **Stream** ciphers encrypt each unit of plaintext, one unit at a time, with a corresponding unit of a key stream. The result is a single unit of ciphertext.

Asymmetric algorithms

AWS services typically support RSA and Elliptic Curve Cryptography (ECC) asymmetric algorithms.

An encryption scheme is called *asymmetric* if it uses one key to encrypt and a different, but mathematically related, key to decrypt. It must be computationally infeasible to determine one key if the only thing one knows is the other key. Therefore, one key can be distributed publicly while the related key is kept secret and secure. Together the keys are referred to as a *key pair*. The key that's publicly distributed is called the *public key* and the key that's kept secret is called the *private key*.

Another more common name for asymmetric encryption is *public-key* cryptography. Public key cryptography is typically based on mathematical problems that are relatively easy to perform but cannot be easily reversed. These include factoring a large integer back into its component prime numbers and solving the elliptic curve discrete logarithm function. The RSA algorithm is based on the practical difficulty of factoring the product of two large prime numbers. Elliptic curve cryptography is based on the difficulty of finding the discrete logarithm of a random point on an elliptic curve given a publicly known point.

AWS cryptographic services and tools

AWS's cryptographic services utilize a wide range of encryption and storage technologies that can assure the integrity of your data at rest or in transit. AWS offers four primary tools for cryptographic operations:

- **AWS CloudHSM (p. 9)** provides hardware security modules (HSMs) (p. 6) that can securely store a variety of cryptographic keys, including master keys (p. 6) and data keys (p. 3).
- **AWS Key Management Service (KMS) (p. 10)** provides tools for generating master keys (p. 6) and other data keys (p. 3). AWS KMS also interacts with many other AWS services to encrypt their service-specific data.
- **AWS Encryption SDK (p. 11)** provides a client-side encryption library for implementing encryption and decryption operations on *all* types of data.
- **Amazon DynamoDB Encryption Client (p. 12)** provides a client-side encryption library for encrypting data tables before sending them to a database service, such as Amazon DynamoDB.
- **AWS Secrets Manager (p. 12)** provides encryption and rotation of encrypted secrets used with AWS-supported datatbases.

Many AWS services rely on these cryptographic services during data transfer or storage. For a list of such services and an overview of how they use cryptographic practices, see Other AWS Services (p. 13).

AWS cryptographic services comply with a wide range of cryptographic security standards, making it easy for you to protect your data without worrying about governmental or professional regulations. For a full list of AWS data security standard compliances, see AWS Compliance Programs.

AWS CloudHSM

AWS CloudHSM is a cryptographic service for creating and maintaining hardware security modules (HSMs) in your AWS environment. HSMs are computing devices that process cryptographic operations and provide secure storage for cryptographic keys. You can use AWS CloudHSM to offload SSL/TLS processing for web servers, protect private keys linked to an issuing certificate authority (CA), or enable Transparent Data Encryption (TDE) for Oracle databases.

When you use an HSM from AWS CloudHSM, you can perform a variety of cryptographic tasks:

- Generate, store, import, export, and manage cryptographic keys, including symmetric keys and asymmetric key pairs.
- Use symmetric and asymmetric algorithms to encrypt and decrypt data.
- Use cryptographic hash functions to compute message digests and hash-based message authentication codes (HMACs).
- Cryptographically sign data (including code signing) and verify signatures.
- Generate cryptographically secure random data.

AWS CloudHSM organizes HSMs in *clusters*, which are automatically synchronized collections of HSMs within a given Availability Zone (AZ). By adding more HSMs to a cluster and distributing clusters across AZs, you can load balance the cryptographic operations being performed within your cloud environment and provide redundancy and high availability in case of AZ failure. Additionally, AWS CloudHSM periodically generates and stores *backups* of your clusters, making CloudHSM data recovery secure and simple.

The keys that you generate in AWS KMS are protected by FIPS 140-2 validated cryptographic modules. If you want a managed service for creating and controlling encryption keys, but do not want or need to operate your own HSM, consider using AWS Key Management Service (p. 10).

To learn more about what you can do with AWS CloudHSM, see the AWS CloudHSM User Guide.

AWS Key Management Service

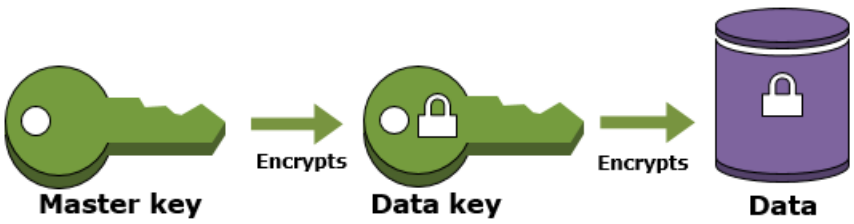
AWS Key Management Service (AWS KMS) is an AWS service that makes it easy for you to create and control the encryption keys that are used to encrypt your data. The master keys that you create in AWS KMS are protected by FIPS 140-2 validated cryptographic modules. They never leave AWS KMS unencrypted. To use or manage your master keys, you interact with AWS KMS.

Many AWS services are integrated with AWS KMS so they encrypt your data with master keys in your AWS account. AWS KMS is also integrated with AWS CloudTrail to deliver detailed logs of all cryptographic operations that use your master keys and management operations that change their configuration. This detailed logging helps you fulfill your auditing, regulatory and compliance requirements.

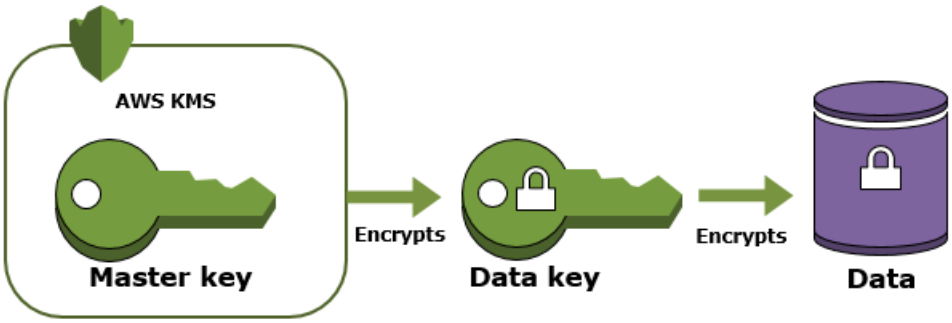
Why use AWS KMS?

AWS KMS protects the *master keys* that protect your data.

In the classic scenario, you encrypt your data using data key A. But you need to protect data key A, so you encrypt data key A by using data key B. Now data key B is vulnerable, so you encrypt it by using data key C. And, so on. This encryption technique, which is called *envelope encryption*, always leaves one last encryption key unencrypted so you can decrypt your encryption keys and data. That last unencrypted (or plaintext) key is called a *master key*.



AWS KMS protects your master keys. The *customer master keys* (CMKs) that KMS supports are created, managed, used, and deleted entirely within KMS. They never leave the service unencrypted. To use or manage your CMKs, you call KMS.



Using and managing AWS KMS customer master keys

AWS KMS customer master keys (CMKs) are 256-bit Advanced Encryption Standard (AES) symmetric keys that are not exportable. They spend their entire lifecycle entirely within AWS KMS.

You can create, view, and manage the AWS KMS customer master keys (CMKs) in your AWS account from the AWS Management Console and AWS KMS API operations. You have full control over your CMKs:

- Establish policies that determine who can use and manage CMKs.
- Enable and disable CMKs.
- Rotate CMK key material.
- Schedule deletion of CMKs when you are finished using them.

You can also use your CMKs in cryptographic operations. You can encrypt and decrypt small amounts of data directly under the CMK. But CMKs are typically used to generate, encrypt, decrypt, and reencrypt exportable data keys that protect your data outside of AWS KMS. You can also give other AWS services permission to use your CMKs on your behalf to encrypt the data that the service stores and manages for you.

More resources and information

You can read about AWS Key Management Service in the AWS Key Management Service Developer Guide and the AWS Key Management Service API Reference. If you have questions, read and post on the AWS KMS Discussion Forum.

If you are required to control and manage the hardware security modules that generate and store your encryption keys, learn about AWS CloudHSM (p. 9).

If you need help using encryption keys to encrypt your data, such as the data keys that AWS KMS returns, learn about the AWS Encryption SDK (p. 11).

AWS Encryption SDK

The AWS Encryption SDK is a client-side encryption library to help you implement best-practice encryption and decryption in any application even if you're not a cryptography expert.

The AWS Encryption SDK works on all types of data. Every successful call to encrypt returns a single portable, formatted encrypted message that contains metadata and the message ciphertext.

The AWS Encryption SDK offers advanced data protection features, including envelope encryption and additional authenticated data (AAD). It also offers secure, authenticated, symmetric key algorithm suites, such as 256-bit AES-GCM with key derivation and signing.

The AWS Encryption SDK is developed as an open source project. It is available in multiple programming languages, including a command line interface that is supported on Linux, macOS, and Windows. All implementations are interoperable. For example, you can encrypt your data with the Java library and decrypt it with the Python library. Or you can encrypt data with the C library and decrypt it with the CLI.

For informations about the AWS Encryption SDK, see the AWS Encryption SDK Developer Guide.

DynamoDB Encryption Client

The Amazon DynamoDB Encryption Client is a client-side encryption library that helps you to protect your table data before you send it to Amazon DynamoDB. Encrypting your sensitive data in transit and at rest helps ensure that your plaintext data isn't available to any third party, including AWS.

The DynamoDB Encryption Client is designed especially for DynamoDB applications. It encrypts the attribute values in each table item using a unique encryption key. It then signs the item to protect it against unauthorized changes, such as adding or deleting attributes or swapping encrypted values. After you create and configure the required components, the DynamoDB Encryption Client transparently encrypts and signs your table items when you add them to a table. It also verifies and decrypts them when you retrieve them.

The DynamoDB Encryption Client is developed in open source. It is available in Java and Python and the language implementations are interoperable. For example, you can encrypt your data with the Java library and decrypt it with the Python library.

For information about the DynamoDB Encryption Client, see the DynamoDB Encryption Client Developer Guide.

AWS Secrets Manager

AWS provides the service AWS Secrets Manager for easier management of secrets. *Secrets* can be database credentials, passwords, third-party API keys, and even arbitrary text. You can store and control access to these secrets centrally by using the Secrets Manager console, the Secrets Manager command line interface (CLI), or the Secrets Manager API and SDKs.

In the past, when you created a custom application to retrieve information from a database, you typically embedded the credentials,the secret, for accessing the database directly in the application. When the time came to rotate the credentials, you had to do more than just create new credentials. You had to invest time to update the application to use the new credentials. Then you distributed the updated application. If you had multiple applications with shared credentials and you missed updating one of them, the application failed. Because of this risk, many customers have chosen not to regularly rotate credentials, which effectively substitutes one risk for another.

Secrets Manager enables you to replace hardcoded credentials in your code, including passwords, with an API call to Secrets Manager to retrieve the secret programmatically. This helps ensure the secret can't be compromised by someone examining your code, because the secret no longer exists in the code. Also, you can configure Secrets Manager to automatically rotate the secret for you according to a specified schedule. This enables you to replace long-term secrets with short-term ones, significantly reducing the risk of compromise.

Secrets Manager encrypts the protected text of a secret by using AWS Key Management Service (AWS KMS). Many AWS services use AWS KMS for key storage and encryption. AWS KMS ensures secure encryption of your secret when at rest. Secrets Manager associates every secret with an AWS KMS CMK. It can be either the default CMK for Secrets Manager for the account, or a customer-created CMK.

To learn more about what you can do with Secrets Manager, see the AWS Secrets Manager User Guide.

Other AWS services that use cryptography

The following AWS services also use cryptography to solve specialized problems.

Service	Description	Topic
Code Signing for AWS IoT	Cryptographically sign code used by IoT devices in FreeRTOS and AWS IoT.	What is Code Signing for AWS IoT?
Amazon Cognito	Use secure identity pools and user pools to authenticate users through third-party identity providers (IdP).	Features of Amazon Cognito
Amazon Managed Blockchain	Creates and manage blockchain networks on AWS.	Key Concepts: Blockchain Networks, Members, and Peer Nodes
Amazon Quantum Ledger Database (QLDB)	Establish immutable, cryptographically viable ledgers to track data changes in an application.	Amazon QLDB Features

How to choose an encryption tool or service

AWS offers several different cryptographic tools and services. This section is designed to help you learn about them and decide which tools and services you should use for your projects.

Most AWS services that store and manage your data support *server-side encryption*, where the service that stores and manages your data also transparently encrypts and decrypts it for you. AWS also supports *client-side encryption* libraries that you can include in your applications. These libraries make it easier to include best-practice encryption in your application, even if you are not a cryptography expert.

Before selecting your cryptographic tools and services, decide if you prefer client-side encryption, server-side encryption, or both. Your decision depends on the design of your application, the sensitivity of your data, and the security requirements of your organization. We try to make our client-side encryption libraries easy to use, but for most applications it's much easier to have an AWS service manage encryption transparently.

What do you need to protect your data?

- Do you need to create and manage the hardware security modules that store your encryption keys? Consider the AWS CloudHSM (p. 14) service.
- Would you benefit from an AWS service that protects your encryption keys for you? Consider AWS Key Management Service (p. 14) (AWS KMS).
- Do you need to protect your data before you send it to AWS? Use a client-side encryption library, like the AWS Encryption SDK (p. 15), the DynamoDB Encryption Client (p. 15), or Amazon S3 client-side encryption.

What type of data do you need to protect?

- To protect DynamoDB table items before you send them to DynamoDB, use the DynamoDB Encryption Client (p. 15). But be sure that you need it. DynamoDB encryption at rest automatically protects all DynamoDB tables whenever they are written to disk.
- To protect Amazon S3 objects before you send them to an Amazon S3 bucket, use Amazon S3 client-side encryption. Amazon S3 also offers server-side encryption.
- To protect all other types of data at their source, use the AWS Encryption SDK (p. 15).

When choosing an SDK or an encryption client library, remember that they are not compatible. You cannot use one library to encrypt data and a different library to decrypt the data.

Topics

- When to use AWS Key Management Service (AWS KMS) (p. 14)
- When to use AWS CloudHSM (p. 14)
- When to use AWS Encryption SDK (p. 15)
- When to use DynamoDB Encryption Client (p. 15)

When to use AWS Key Management Service (AWS KMS)

When you encrypt data, you need to protect your encryption key. If you encrypt your key, you need to protect its encryption key. Eventually, you must protect the highest level encryption key (known as a *master key*) in the hierarchy that protects your data. That's where AWS KMS comes in.

AWS Key Management Service (AWS KMS) lets you create, store, and manage customer master keys (CMKs) securely. Your CMKs never leave AWS KMS unencrypted. To use a CMK in a cryptographic operation, you call KMS.

Additionally, you can create and manage key policiesin AWS KMS, ensuring that only trusted users have access to CMKs.

When Do I Use It?

- Use AWS KMS to create and manage master keys (CMKs). You can establish policies that determine who can use your CMKs and how they can use them. You can track their use in transaction and audit logs, such as AWS CloudTrail.
- You can use your CMKs to encrypt small amounts of data (up to 4096 bytes). However, CMKs are typically used to generate, encrypt, and decrypt the data keys that encrypt your data. Unlike CMKs, data keys can encrypt data of any size and format, including streamed data.

When Do I Use Something Else?

- AWS KMS does not store or manage data keys, and you cannot use KMS to encrypt or decrypt with data keys. To use data keys to encrypt and decrypt, use the AWS Encryption SDK.
- AWS KMS CMKs are backed by FIPS-validated hardware service modules (HSMs) that KMS manages. To manage your own HSMs, use AWS CloudHSM (p. 14).

When to use AWS CloudHSM

AWS CloudHSM is a service for creating and managing cloud-based hardware security modules. A *hardware security module* (HSM) is a specialized security device that generates and stores cryptographic keys.

When Do I Use It?

- Use AWS CloudHSM when you need to manage the HSMs that generate and store your encryption keys. In AWS CloudHSM, you create and manage HSMs, including creating users and setting their permissions. You also create the symmetric keys and asymmetric key pairs that the HSM stores.

When Do I Use Something Else?

- If you need to secure your encryption keys in a service backed by FIPS-validated HSMs, but you do not need to manage the HSM, try AWS Key Management Service (p. 14).

When to use AWS Encryption SDK

The AWS Encryption SDK is a client-side encryption library that makes it easier to encrypt and decrypt data of any type in your application. The Encryption SDK is available in several programming languages, including a command-line interface.

You can use the AWS Encryption SDK to encrypt your data before you send it to an AWS service. You can also use it with customer master keys in AWS Key Management Service (AWS KMS). However, the library does not require any AWS service.

When you encrypt data, the SDK returns a single, portable encrypted message that includes the encrypted data and encrypted data keys. This object is designed to work in many different types of applications. You can specify many of the encryption options, including selecting an encryption and signing algorithm.

When Do I Use It?

- Use the AWS Encryption SDK to encrypt and decrypt data in a script or application. You can use it with AWS Key Management Service (p. 10) or any compatible master key provider.

When Do I Use Something Else?

- Many AWS services optionally encrypt the data that they store and manage for you. (This is known as *server-side encryption*.) Many of these services are integrated with AWS KMS. For details, see the information about encryption options in the service documentation.
- You might want to use a client-side encryption library that includes special features for your data, such as Amazon S3 client-side encryption or the DynamoDB Encryption Client (p. 12).

  When you choose an SDK or encryption client library, remember that libraries are not compatible with one another. That is, you cannot use one library to encrypt data and a different library to decrypt the data. Unless you need a feature provided only by a different library, use the AWS Encryption SDK.

When to use DynamoDB Encryption Client

The Amazon DynamoDB Encryption Client is a client-side encryption library designed especially for data that you store in Amazon DynamoDB (DynamoDB). It encrypts the attribute values in each table item using a unique encryption key and then signs the item to protect it against unauthorized changes. Unauthorized changes include adding or deleting attributes or swapping encrypted values.

You can use the DynamoDB Encryption Client to encrypt and sign your table items before you send them to DynamoDB. It is compatible with the encryption at rest server-side encryption feature that DynamoDB provides for all tables. For a detailed comparison of the DynamoDB Encryption Client and DynamoDB encryption at rest, see the Client-Side and Server-Side Encryption topic in the *Amazon DynamoDB Encryption Client Developer Guide*.

When Do I Use It?

- If you need to encrypt and sign DynamoDB table items before you send them to DynamoDB, use the DynamoDB Encryption Client.
- You can also use the DynamoDB Encryption Client to encrypt and sign data that is structured like DynamoDB table items—with attributes and values—even if it is not destined for DynamoDB. The DynamoDB Encryption Client does not require any AWS service.

When Do I Use Something Else?

- You can rely on the server-side encryption at rest feature that Amazon DynamoDB provides. DynamoDB transparently encrypts all tables before writing them to disk and transparently decrypts the tables when you get them. Encryption at rest is provided by default, and you cannot disable it. However, if your data security standards require it, you can use both the DynamoDB Encryption Client and encryption at rest on your table data.
- With the DynamoDB Encryption Client, you can specify which attribute values you encrypt and which attributes are included in the item signature. However, if you have unusual data protection requirements that the DynamoDB Encryption Client cannot satisfy, you might be able to use the AWS Encryption SDK (p. 11) to protect your data.

When choosing an SDK or encryption client library, remember that they are not compatible. You cannot use one library to encrypt data and a different library to decrypt the data.

AWS PKI services

AWS provides multiple services that you can use to establish trust across the data transit process. These services are introduced in AWS Public Key Infrastructure (p. 21) and detailed thoroughly in their respective documentation sets.

Topics

- What is public key infrastructure? (p. 17)
- PKI concepts (p. 17)
- AWS public key infrastructure (PKI) services and tools (p. 21)
- How to choose a PKI service (p. 22)

See also: *AWS cryptographic services and tools* (p. 1)

What is public key infrastructure?

Public key infrastructure (PKI) (p. 20) is a system of hardware, software, people, policies, documents, and procedures. It includes the creation, issuance, management, distribution, usage, storage, and revocation of digital certificates. These certificates are then used to authenticate the identities of various actors across the data transfer process. They also assure that the data being moved between these actors is secured and encrypted in a way that both parties can decrypt. This way, information is only being sent to and received from known and trusted (p. 20) sources, and both parties are assured of the information's integrity. To learn more about the encryption and decryption processes that are used in data transfer and storage, see the What is Cryptography? (p. 1) section of this guide.

PKI trust is established by a *certificate authority* (p. 18), which is an organization or governing body that can issue certificates and verify the identity of the certificate requestor. AWS offers multiple PKI certificate authority services that can help you easily and securely manage your certificate infrastructure.

To learn more about the terms and concepts used in PKI, see PKI Concepts. (p. 20)

PKI concepts

As you work with AWS PKI tools and services, you are likely to encounter a number of basic concepts.

Topics

- asymmetric key cryptography (p. 18)
- certificate authority (CA) (p. 18)
- certificate authority certificate (p. 18)
- certificate signature (p. 18)
- domain name (p. 18)
- Domain Name System (DNS) (p. 19)
- HTTPS (p. 19)
- private certificate (p. 19)
- public certificate (p. 20)
- public key infrastructure (PKI) (p. 20)

- root certificate (p. 20)
- Secure Sockets Layer (SSL) and Transport Layer Security (TLS) (p. 20)
- SSL server certificates (p. 20)
- symmetric key cryptography (p. 20)
- TCP (p. 20)
- trust (p. 20)

asymmetric key cryptography

The use of different but mathematically related keys to encrypt and decrypt content. One of the keys is public and is typically made available in an X.509 version 3 certificate. The other key is private and is stored securely. The X.509 certificate binds the identity of a user, computer, or other resource (the certificate subject) to the public key. See also: symmetric key cryptography (p. 20).

certificate authority (CA)

A trusted third party that issues (and, if necessary, revokes) digital certificates. The most common type of certificate is based on the ISO X.509 standard. An X.509 certificate affirms the identity of the certificate subject and binds that identity to a public key. The subject can be a user, an application, a computer, or other device. The CA signs a certificate (p. 18) by hashing the contents and then encrypting the hash with the private key related to the public key in the certificate. A client application such as a web browser that needs to affirm the identity of a subject uses the public key to decrypt the certificate signature. It then hashes the certificate contents and compares the hashed value to the decrypted signature to determine whether they match.

For information about certificate signing, see certificate signature (p. 18).

certificate authority certificate

A certificate that affirms the identity of the certificate authority (CA) (p. 18) and binds it to the public key that is contained in the certificate.

certificate signature

An encrypted hash over a certificate that affirms the integrity of the certificate data. The encrypted hash is known as a digital signature. Your private CA creates a signature by using a hash function (such as SHA256) over the variable-sized certificate content to produce an irreversible fixed-size data string. The fixed data is called a hash. The CA then encrypts the hash value with its private key and concatenates the encrypted hash with the certificate.

To validate a signed certificate, a client application uses the CA public key to decrypt the signature. The client then uses the same signing algorithm that the CA used to compute a hash over the rest of the certificate. Note that the signing algorithm used by the CA is listed in the certificate. If the computed hash value is the same as the decrypted hash value, the certificate has not been tampered with.

domain name

A text string such as `www.example.com` that can be translated by the Domain Name System (DNS) into an IP address. Computer networks, including the internet, use IP addresses rather than text names. A domain name consists of distinct labels separated by periods:

TLD

The rightmost label of a domain name, or top-level domain. Common examples include `.com`, `.net`, and `.edu`. Also, the TLD for entities that are registered in some countries is an abbreviation of the country name and is called a country code. Examples include `.uk` for the United Kingdom, `.ru` for Russia, and `.fr` for France. When country codes are used, a second-level hierarchy for the TLD is often introduced to identify the type of the registered entity. For example, the `.co.uk` TLD identifies commercial enterprises in the United Kingdom.

apex domain

The portion of a domain name that includes and expands on the top-level domain. For domain names that include a country code, the apex domain includes the code and the labels, if any, that identify the type of the registered entity. The apex domain does not include subdomains (see the following paragraph). In `www.example.com`, the name of the apex domain is `example.com`. In `www.example.co.uk`, the name of the apex domain is `example.co.uk`. Other names that are often used instead of apex include *base*, *bare*, *root*, *root apex*, or *zone apex*.

subdomain

The portion of a domain name (if present) that precedes the apex domain name and is separated from it and from other subdomains by a period. The most common subdomain name is `www`, but any name is possible. Also, subdomain names can have multiple levels. For example, in `jake.dog.animals.example.com`, the subdomains are `jake`, `dog`, and `animals` in that order.

FQDN

Fully qualified domain name. The FQDN is the complete DNS name for a computer, website, or other resource that is connected to a network or to the internet. For example `aws.amazon.com` is the FQDN for Amazon Web Services. An FQDN includes all domains up to the top-level domain. For example, $[subdomain_1].[subdomain_2]...[subdomain_n].[apex\ domain].[top\text{-}level\ domain]$ represents the general format of an FQDN.

PQDN

Partially qualified domain name. A PQDN is not fully qualified and is ambiguous. A name such as $[subdomain_1.subdomain_2.]$ is a PQDN because the root domain cannot be determined.

registration

The right to use a domain name that is delegated by domain name registrars. Registrars are typically accredited by the Internet Corporation for Assigned Names and Numbers (ICANN). In addition, other organizations called registries maintain the TLD databases. When you request a domain name, the registrar sends your information to the appropriate TLD registry. The registry assigns a domain name, updates the TLD database, and publishes your information to WHOIS. Typically, domain names must be purchased.

Domain Name System

A hierarchical distributed naming system for computers and other resources connected to the internet or a private network. DNS is primarily used to translate textual domain names, such as `aws.amazon.com`, into numerical IP (internet protocol) addresses of the form `111.222.333.444`. The DNS database for your domain, however, contains a number of records that can be used for other purposes. For example, with AWS Certificate Manager (p. 21) you can use a CNAME record to validate that you own or control a domain when you request a certificate.

HTTPS

The abbreviation for *HTTP over SSL/TLS (p. 20)*, a secure form of Hypertext Transfer Protocol (HTTP) that is supported by all major browsers and servers. All HTTP requests and responses are encrypted before being sent across a network. HTTPS combines the HTTP protocol with symmetric (p. 20), asymmetric (p. 18), and X.509 certificate-based cryptographic techniques. HTTPS works by inserting a cryptographic security layer below the HTTP application layer and above the TCP (p. 20) transport layer in the Open Systems Interconnection (OSI) model. The security layer uses the Secure Sockets Layer (SSL) protocol or the Transport Layer Security (TLS) protocol.

private certificate

An SSL/TLS (p. 20) certificate that verifies the ownership of a private key. Use them to identify resources such as clients, servers, applications, services, devices, and users. When establishing a

secure encrypted communications channel, each resource uses a certificate as well as cryptographic techniques to prove its identity to another resource. Internal API endpoints, web servers, VPN users, IoT devices, and many other applications use private certificates to establish encrypted communication channels that are necessary for their secure operation. By default, private certificates are not publicly trusted. An internal administrator must explicitly configure applications to trust private certificates and distribute the certificates.

See also public certificate (p. 20).

public certificate

An SSL/TLS (p. 20) certificate that verifies the ownership of a public key and is used to initiate a secure, encrypted connection between a web server and a client. Public certificate trust (p. 20) is built upon the integrity of an issuing entity, called a certificate authority (CA) (p. 18), which authenticates the identity of a certificate through the use of a certificate signature (p. 18).

See also private certificate (p. 19).

public key infrastructure (PKI)

A comprehensive system that enables the creation, issuance, management, distribution, use, storage, and revocation of digital certificates. A PKI consists of people, hardware, software, policies, documents, and procedures.

root certificate

The certificate issued by the certificate authority (CA) (p. 18) that is at the top of the CA hierarchy. A certificate authority typically exists within a hierarchical structure that contains multiple other CAs with clearly defined parent-child relationships between them. Child or subordinate CAs are certified by their parent CAs, creating a certificate chain. The CA at the top of the hierarchy is referred to as the root CA, and its certificate is called the root certificate. This certificate is typically self-signed.

Secure Sockets Layer (SSL) and Transport Layer Security (TLS)

Cryptographic protocols that provide communication security over a computer network. TLS is the successor of SSL. They both use X.509 certificates to authenticate the server. Both protocols negotiate a symmetric key between the client and the server that is used to encrypt data flowing between the two entities.

SSL server certificates

An X.509 version 3 data structure that binds the public key in the certificate to the subject of the certificate and is signed by a certificate authority (CA) (p. 18). An SSL/TLS certificate contains the name of the server, the validity period, the public key, the signature algorithm, and more. Server certificates are required for HTTPS transactions to authenticate a server.

symmetric key cryptography

The practice of using the same key to both encrypt and decrypt data. See also: asymmetric key cryptography (p. 18).

Transmission Control Protocol (TCP)

As part of the TCP/IP stack, TCP is one of the main sets of rules used when sending data between networks. By verifying the order in which information is received, TCP assures the quality of transmissions across hosts and servers. TCP standards are regulated by the Internet Engineering Task Force (IETF).

trust

The reliability of a website's identity as established by verifying the website's certificate. BrowsersC trust only a small number of certificates known as CA root certificates (p. 20). A trusted third party, known as a certificate authority (CA) (p. 18), validates the identity of the website and issues a signed digital certificate to the website's operator. The browser can then check the digital

signature to validate the identity of the website. If validation is successful, the browser displays a lock icon in the address bar.

AWS public key infrastructure (PKI) services and tools

AWS offers multiple PKI services that can help you easily and securely manage your certificate infrastructure. The primary AWS offerings for PKI are tightly linked:

- **AWS Certificate Manager (ACM) (p. 21)** is used to generate, issue, and manage public and private SSL/TLS certificates (p. 20) for use with your AWS based websites and applications.
- **AWS Certificate Manager Private Certificate Authority (ACM PCA) (p. 21)** is a managed private certificate authority (CA) service with which you can manage your CA infrastructure and your private certificates (p. 19).

Many AWS services rely on these PKI services to authenticate the actors involved in a data transfer process. For a list of such services and an overview of how they use PKI practices, see Other AWS Services That Use SSL/TLS Certificates (p. 22).

AWS PKI services comply with a wide range of security standards, making it easy for you to protect your data without worrying about governmental or professional regulations. For a full list of AWS data security standard compliances, see AWS Compliance Programs.

AWS Certificate Manager

AWS Certificate Manager is a PKI (p. 20) service that handles the complexity of creating and managing public SSL/TLS certificates (p. 20) for your AWS based websites and applications. These public certificates verify the identity and authenticity of your web server and the ownership of your public keys. In doing so, public certificates initiate a trusted (p. 20), encrypted (p. 4) connection between you and your users. You can also use ACM in conjunction with AWS Certificate Manager Private Certificate Authority (p. 21) to create and manage private certificates, which can be used to authenticate the identity of internal organization entities. Certificates in ACM, whether generated in ACM or imported from a third-party authority, can secure multiple domain names and multiple names within a domain. You can also use ACM to create wildcard SSL certificates that can protect an unlimited number of subdomains.

You can use ACM to perform a variety of PKI-related tasks, including the generation, validation, renewal, maintenance, and deletion of public and private certificates. For more information about ACM, see the AWS ACM User Guide.

AWS Certificate Manager Private Certificate Authority

AWS Certificate Manager Private Certificate Authority (ACM PCA) is a private certificate authority (CA) (p. 18) service with which you can easily and securely manage your PKI CA infrastructure and your private certificates (p. 19). ACM PCA provides a highly available private CA service without the investment and maintenance costs of operating your own subordinate CA. By extending the scope of AWS Certificate Manager ACM (p. 21) to private certificates, ACM PCA helps you manage both public and private certificates from within one PKI environment.

You can use ACM PCA to perform a variety of PKI-related tasks, including the creation of private CAs and the generation, validation, renewal, maintenance, and deletion of private certificates. For more information about ACM PCA, see the ACM PCA User Guide.

Other AWS services that use X.509 public key certificates

The following AWS services also include features that allow you to manage and implement X.509 public key certificates for use as SSL/TLS certificates (p. 20) or for code signing.

Service	Description	Topic
Amazon API Gateway	Establish a custom API and manage the API domain information.	Set Up Custom Domain Name for an API Gateway
AWS CloudFormation	Automatically provision AWS resources, including ACM certificates.	Request an ACM Certificate
Amazon CloudFront	Distribute dynamic and static web content to end users as efficiently as possible.	Choosing How CloudFront Serves HTTPS Requests
Code Signing for AWS IoT	Cryptographically sign code using a certificate such as those provisioned by ACM.	What is Code Signing for AWS IoT?
Elastic Beanstalk	Easily deploy applications in the AWS Cloud with automated infrastructure provisioning.	Configure Your Load Balancer to Terminate HTTPS
Elastic Load Balancing	Distribute incoming application traffic across multiple Amazon EC2 instances as efficiently as possible.	HTTPS Listener for Your Application Load Balancer

How to choose a PKI service

AWS offers two primary PKI services, ACM (p. 21) and ACM PCA (p. 21). Use the guidance here to help you decide which service to use for a given scenario.

When to use ACM

A public SSL/TLS (p. 20) certificate is required to authenticate the identity of your web server and establish a secure connection with any trustworthy host it might interact with. With ACM, you can easily create and manage public and private SSL/TLS certificates or import an external public certificate (p. 20) into your AWS environment.

When Do I Use It?

Use ACM when you need to create a new public certificate, renew a public certificate created with ACM, or import an existing public certificate into your AWS environment.

Use ACM to generate a private certificate and manage it within the same environment as your public certificates. You must first use ACM PCA to establish a private CA from which private certificates can be validated. Private certificates created in ACM are bound by the following restrictions:

- They must use RSA-2048 keys and SHA-256 hashing.
- They must be renewed after 13 months.
- Their subject must be a DNS (p. 19) name.

When to use ACM PCA

Private certificates are issued by a private CA (p. 18) and are exclusively used for authentication between entities within your organization. As a result, private certificates cannot be publically trusted. ACM PCA lets you establish a private CA and use it to create and manage private certificates under its authority. Private certificates can be managed by ACM PCA as a standalone service or in conjunction with ACM.

- Use ACM PCA if you need to create an internal CA for further authentication operations.
- Use ACM PCA if you need to generate a private certificate for internal entity authentication.

  Note
  ACM can also generate private certificates once a private CA has been established. But ACM PCA gives you more control over the management and encryption protocols of those private certificates.

Document history for AWS cryptography services

update-history-change	update-history-description	update-history-date
Moved Secrets Manager to a new page.	Description of Secrets Manager was incorrect.	March 9, 2020
Initial release	Initial release of this documentation.	December 1, 2018