AWS Key Management Services Overview

With Hands-on

Agenda

- Encryption Primer
  - Symmetric, Asymmetric Encryption
  - Hashing, Digital Signature & Digital Certificates
- KMS Services Overview
  - Different key types, Key hierarchy
  - Customer Master Key
  - AWS vs Customer Managed Key
  - Client side vs Server side Encryption
- KMS Demo

Symmetric Key Encryption

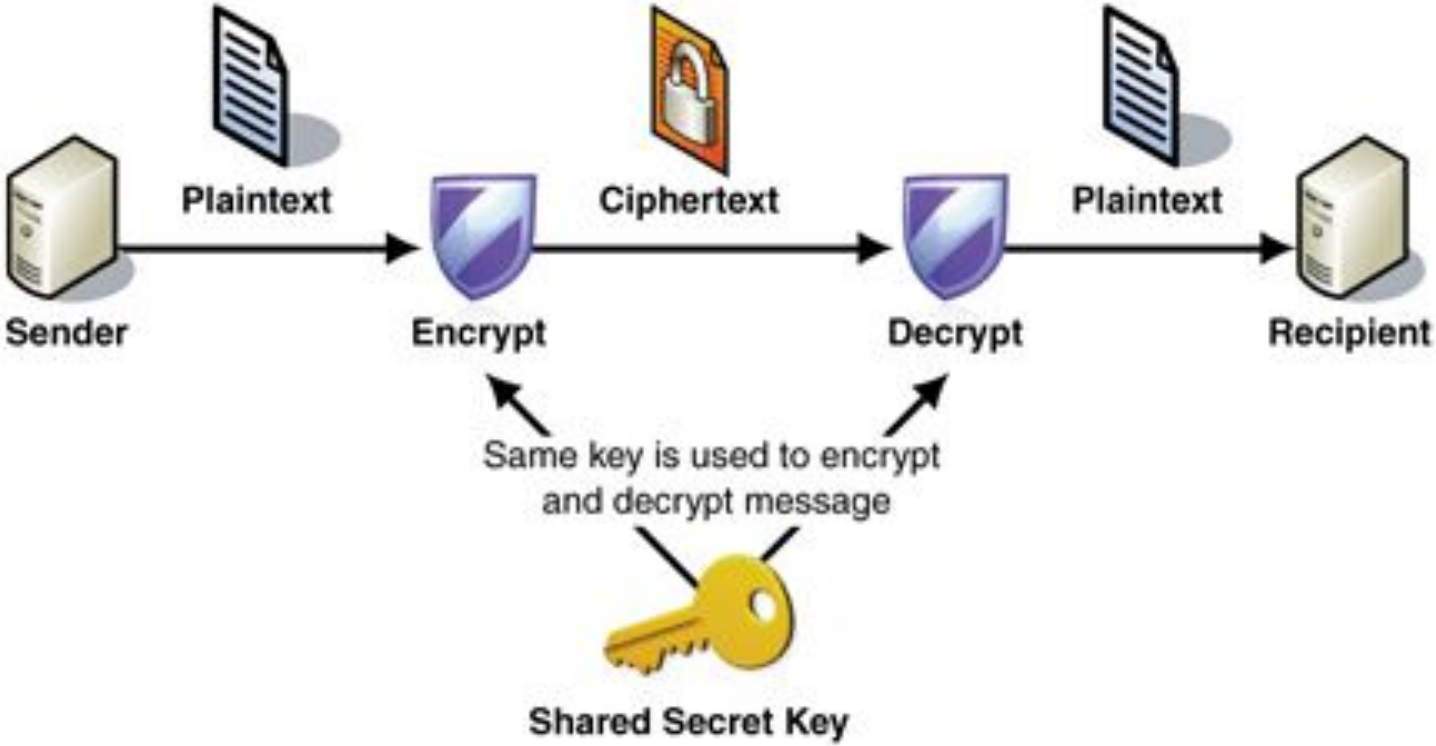


Diagram source: http://jrruethe.github.io/blog/2014/10/25/cryptography-primer/

Symmetric Key Encryption Summary

- Symmetric Key Encryption uses same key for Encryption & decryption
- Encryption Algorithms are not held as secret
- Some popular Symmetric Key Encryption algorithms are AES, 3DES, RCx, Blowfish etc
- Know the difference between Algorithms vs Keys
- Key exchange has to be done out of band
- Symmetric Key Encryption typically uses smaller key sizes and is fast
- But prone to Man in the middle attack (if the key exchange is done in an insecure way)

Man-in-the Middle Attack

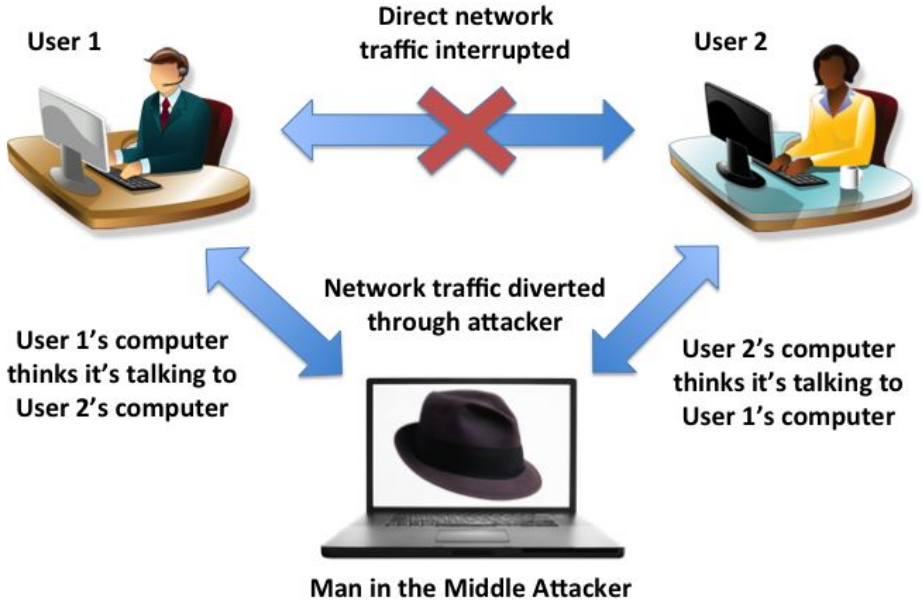


Diagram source: http://jrruethe.github.io/blog/2014/10/25/cryptography-primer/

Man-in-the Middle Attack Summary

- A man in the middle (MiTM) attack happens when someone is intercepting/tampering your communication line by placing themselves in-line in between you and the other person.
- In MiTM, an attacker is able to tamper with the communication line without either party knowing; Compare this with Passive sniffing or eavesdropping, where only “bad person” is collecting information
- In the previous slide, User1 is thinking that he/she is talking to User2 and vice versa
- But the “bad person/attacker” is performing an active MiTM by impersonating as User2 with User1 and vice versa
- This situation can be alleviated if both users encrypt their communications using “key” shared between them using a secure channel (and ensure the man in the middle doesn’t gets hold of it)
- Not a good idea to use the same key all the time with Symmetric key encryption;
    - “Session key” - Unique key for each transaction/flow
    - Challenge is to securely transfer them between trusted parties
    - Can Asymmetric key encryption help?

Asymmetric Key Encryption

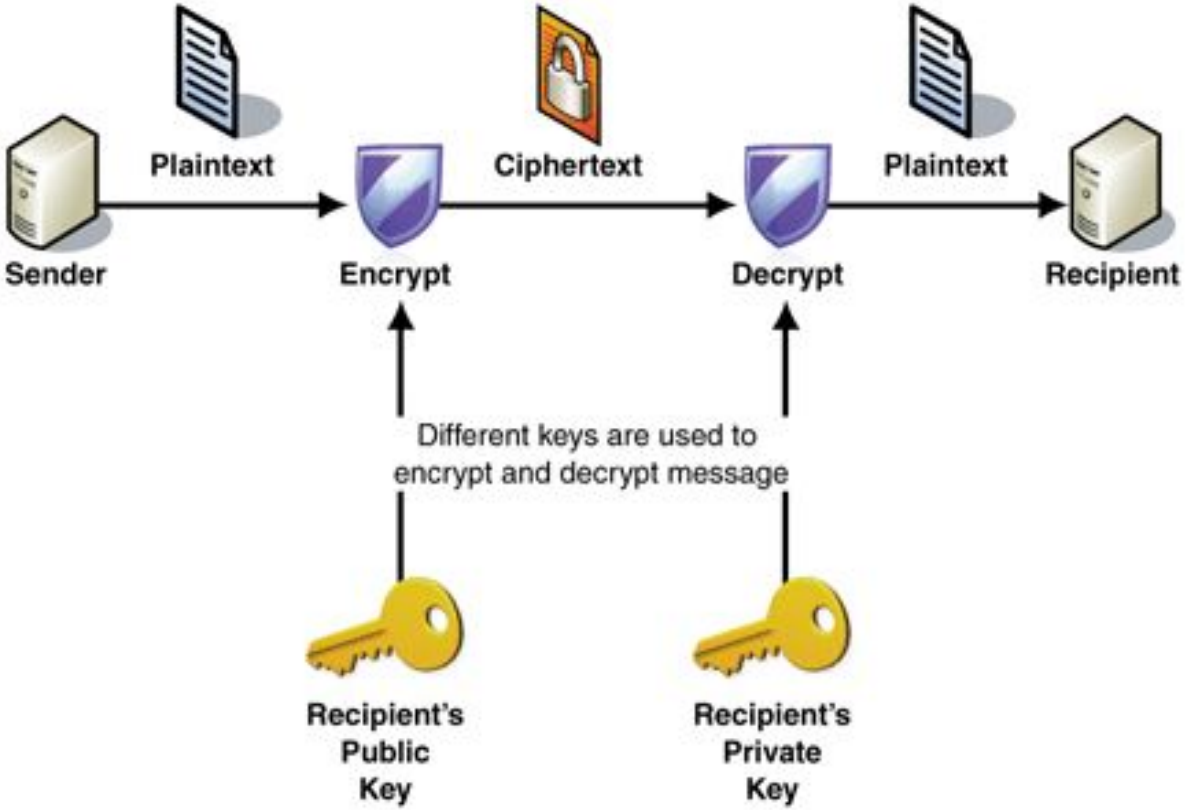


Diagram source: http://jrruethe.github.io/blog/2014/10/25/cryptography-primer/

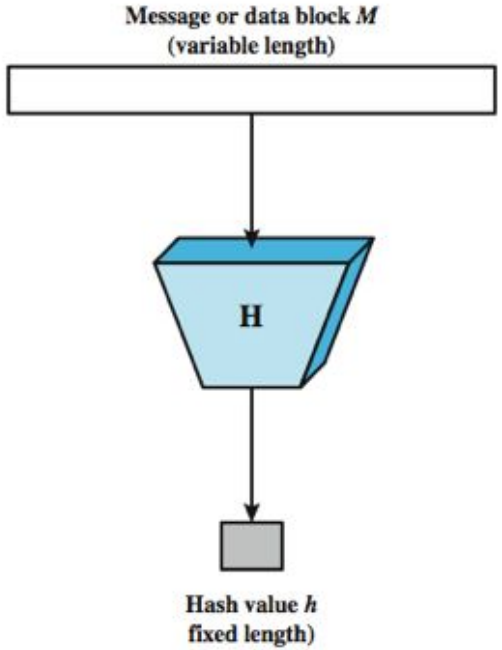
Asymmetric Key Encryption Summary

- Asymmetric key encryption algorithms use two keys to perform crypto operations
- Two keys are called Private/secure & Public key, they are mathematically connected
- Each user would be using a pair of key to perform encryption/decryption operation
- Private/secure should never be shared with anyone
- Public key can be shared with others

Asymmetric Key Encryption Summary

- How to communicate securely using Asymmetric key encryption?
- Users generate a pair of keys for themselves using well known asymmetric key encryption algorithms
- Users will safe their own private/secure key and share public with others they want to communicate with
- If user1 wants to communicate with user2,
  - User1 uses publicly available user2's public key to encrypt the communication
  - Sends the encrypted data to user2
  - User2 uses his/her private key to decrypt the data
  - Data encrypted with user2's public key can only be decrypted using user's private key
- RSA, ECC, DSA, DH key exchange are some well known Asymmetric key encryption algorithms (uses typically large key sizes)
- What if user2 is an impersonator and tricks user1 with his public key?

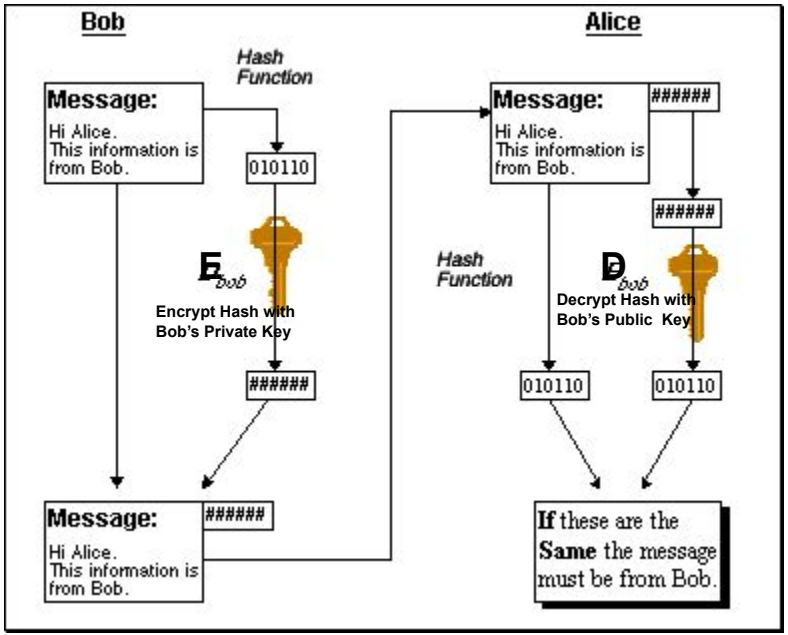
Hashing



- Hashes are used to generate digital fingerprints of a message
- Hashes don’t encrypt data
- Hashing algorithms take variable length data and provides a fixed length data
- Two messages that are identical will produce identical hash
- Even if there is a single bit of difference between two messages, hash won’t match
- Used for checksum calculations, storing passwords

Diagram source: http://jrruethe.github.io/blog/2014/10/25/cryptography-primer/

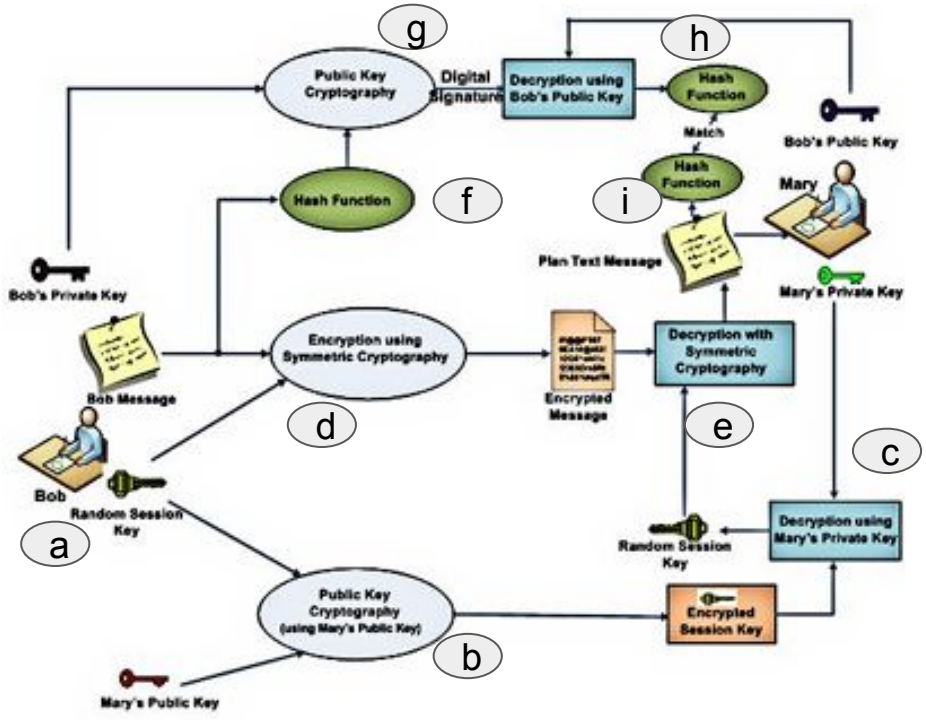
Digital Signature



- Provides assurance that a message indeed came from a person
- If a user encrypts a message with his/her own private key, it can be decrypted by anyone (using the public key pair)
- But one can assume that the message came from the owner of the private key
- When someone encrypts a message using their private key it's called Digital signature
- Since Asymmetric key encryption uses large keys, it's recommended to create a hash of the message and then use that to encrypt
- Send both the original message and hash
- Receiver can calculate the hash from the rxed message and compare the decrypted hash rxed from the sender

Diagram source: http://jrruethe.github.io/blog/2014/10/25/cryptography-primer/

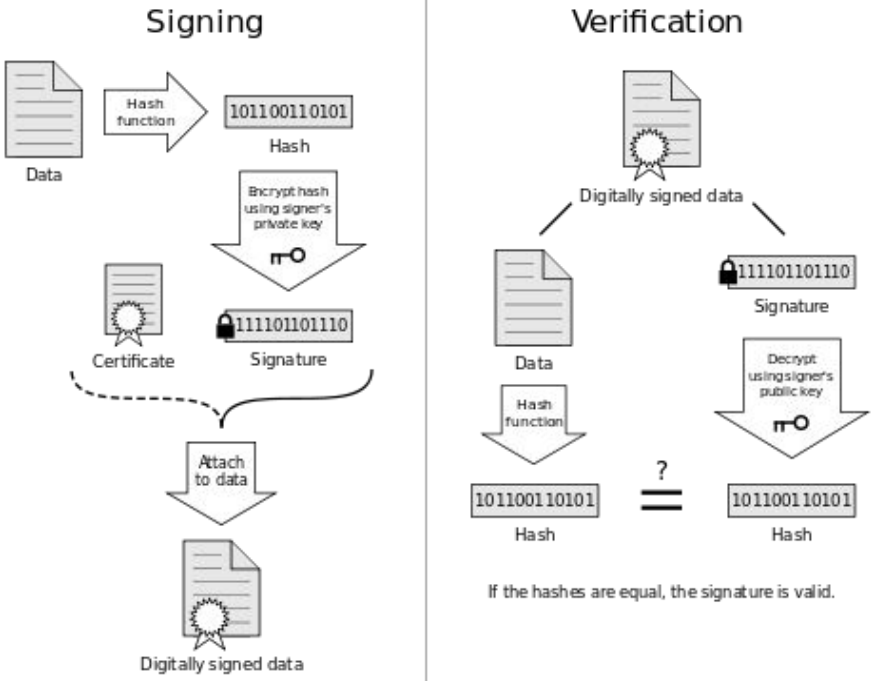
End to End Secure Communication/Digital Signature



- Putting all these concepts together, we get what's called as self-signed certificate
  a. Bob generates a session key
  b. Uses Mary's pub key to encrypt and send it to her
  c. Mary decrypts the message (using her own private key) to get the session key
  d. Bob encrypts his public key using session key and sends the encrypted message to Mary
  e. Mary decrypts the message using session key to get Bob's public key
  f. Bob creates a hash of his public key
  g. Bob encrypts the hash using his private key and send it to Mary
  h. Mary decrypts the hash using Bob's pub key
  i. Mary calculates the hash of Bob's public key and compares with what Bob has sent

Diagram source: http://jrruethe.github.io/blog/2014/10/25/cryptography-primer/

Digital Certificates



- Almost similar to the process outlines in the previous slide
- But the public key of an user or entity will be signed by a trusted 3rd party, aka CA or Certificate Authority

Diagram source: http://jrruethe.github.io/blog/2014/10/25/cryptography-primer/

AWS Key Management Service Overview

- AWS Key Management Service (AWS KMS) is a managed service that provides an easy way for customers to create and control customer master keys (CMKs), the encryption keys used to encrypt your data.
- Customer master keys are the primary resources in AWS KMS.
- The CMK contains the key material used to encrypt and decrypt data.
- The CMK also includes metadata, such as the key ID, creation date, description, and key state.
- AWS KMS supports symmetric and asymmetric CMKs.
- A symmetric CMK represents a 256-bit key that is used for encryption and decryption.
- An asymmetric CMK represents an RSA key pair that is used for encryption and decryption or signing and verification (but not both), or an elliptic curve (ECC) key pair that is used for signing and verification. For detailed information about symmetric and asymmetric CMKs
- AWS KMS CMKs are protected by hardware security modules (HSMs) that are validated by the FIPS-140-2 program (except in China)

Slide content adopted source: https://docs.aws.amazon.com/kms/latest/developerguide/concepts.html

AWS KMS Overview

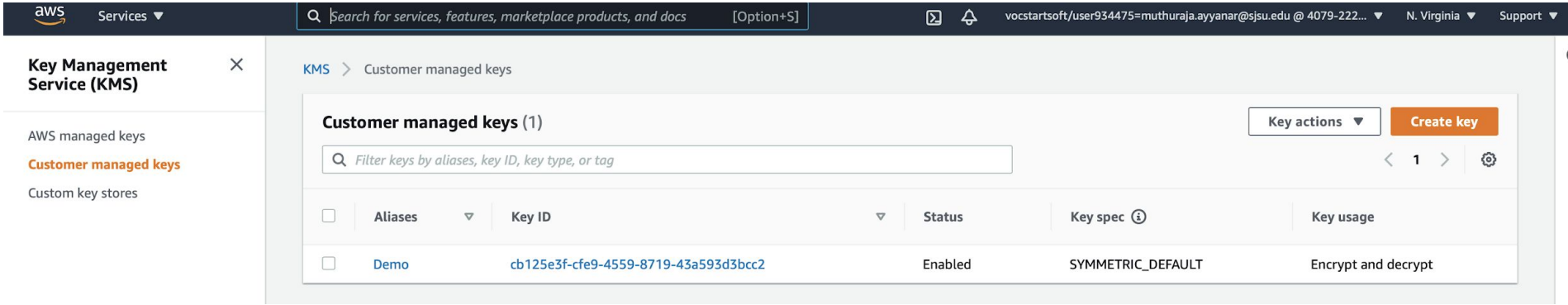
- CMKs are created in AWS KMS.
- Symmetric CMKs and the private keys of asymmetric CMKs never leave AWS KMS unencrypted.
- To manage CMK, both the AWS Management Console or the AWS KMS API be used.
- To use a CMK in crypto operations/applications, use the AWS KMS API.
- AWS KMS does not store, manage, or track your data keys. You must use them outside of AWS KMS.
- By default, AWS KMS creates the key material for a CMK. Customers cannot extract, export, view, or manage this key material. Also, customers cannot delete this key material; but must delete the CMK.

AWS KMS Supported Customer Master Key

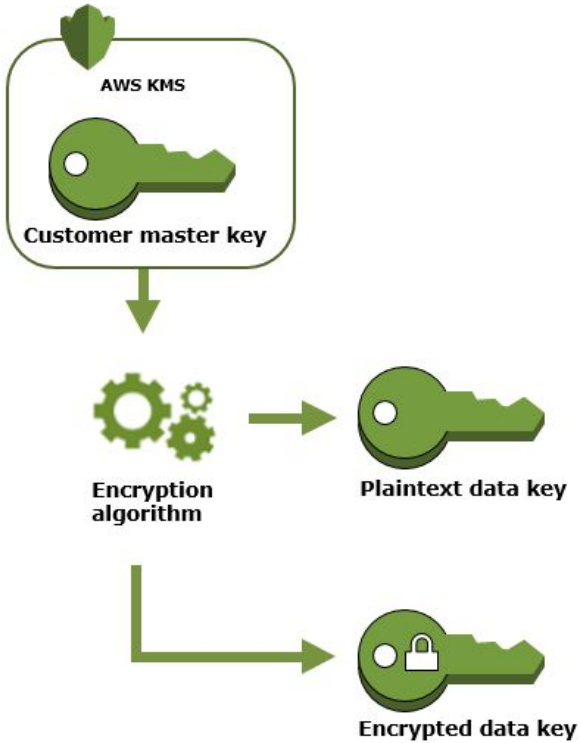
Customer Managed CMK	AWS Managed CMK	AWS Owned CMK
AWS Account owner/customer create, own, and manage CMK. Customer managed CMK can be used in crypto operations/applications by the customer	AWS managed CMKs are created, managed, and used on your behalf **by an AWS service integrated with KMS**	AWS owned keys are common keys that may be used across many AWS accounts or may be specific to an AWS service that's used across all AWS accounts
It's customer's responsibility to enable/disable/rotate key, define policy to grant/deny access to CMK	Customer can neither manage these keys (AWS KMS manages access policy, rotation etc) nor use them in crypto operations	Customers can neither view nor manage these keys
Many AWS services provides an option to integrate customer managed CMK	AWS managed CMKs by their aliases, which have the format aws/service-name, such as aws/redshift	
Customer Managed CMK is listed in the AWS console under "Customer Managed Keys"	AWS managed CMKs appear on the AWS managed keys page of the AWS Management Console for AWS KMS	
Customer Managed CMK is charged after the free tier allowance is exceeded	Customer incurs no charge with AWS managed CMK	Customer incurs no charge with AWS managed CMK

AWS KMS Overview

- CMK - Customer Master Key
- Use AWS console to create customer managed CMK
- Can be done using AWS CLI/SDK
- Use to create “data-keys” that can be used for crypto operations
- Never leaves AWS



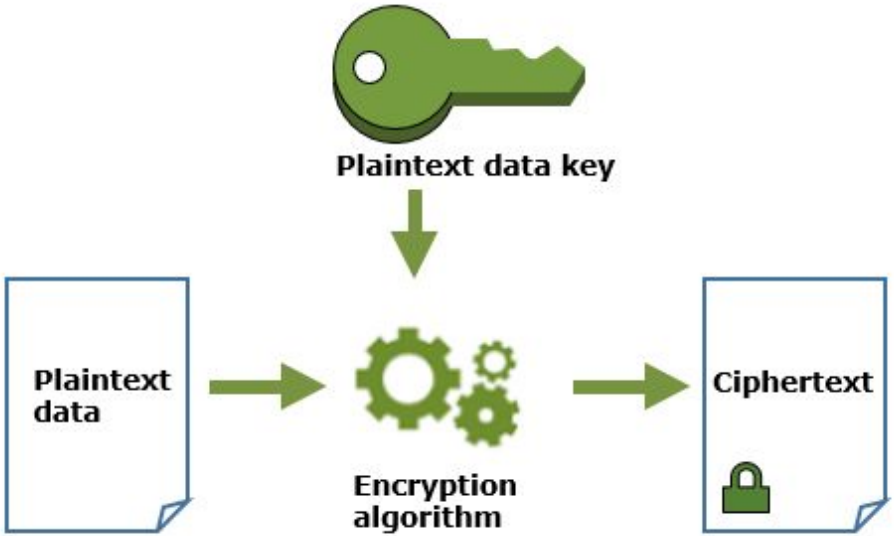
Data Key Creation



- Create Data Key using AWS CLI or SDK
- AWS KMS provides both plaintext & encrypted blob of the data key
- We will see this in the demo

Diagram source: https://docs.aws.amazon.com/kms/latest/developerguide/concepts.html

Encrypt Data



- Use the datakey obtained in the previous step to encrypt
- After this step both the plaintext data & datakey can be discarded
- We will see this in the demo

Diagram source: https://docs.aws.amazon.com/kms/latest/developerguide/concepts.html

Decrypt Data

- Use the encrypted Data Key to obtain the “plaintext data key” from AWS KMS using AWS CLI or SDK
- Use the plaintext data key along with the encrypted/ciphertext to decrypt
- We will see this in the demo

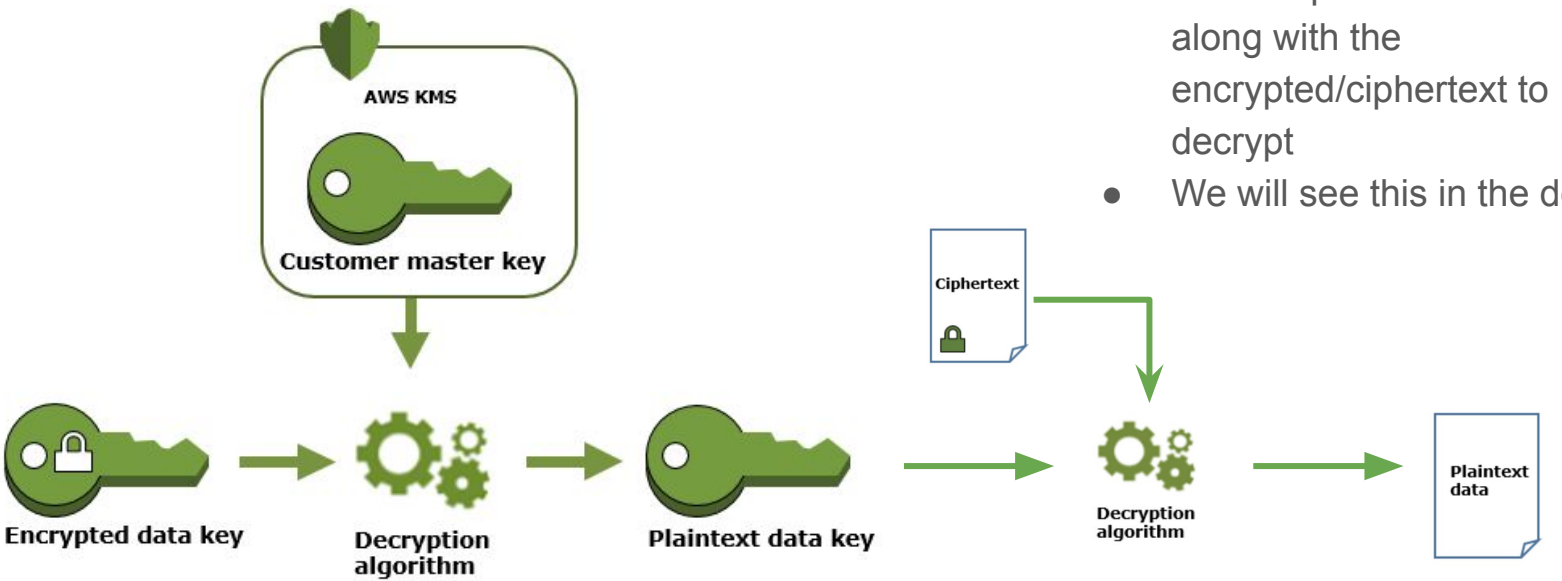
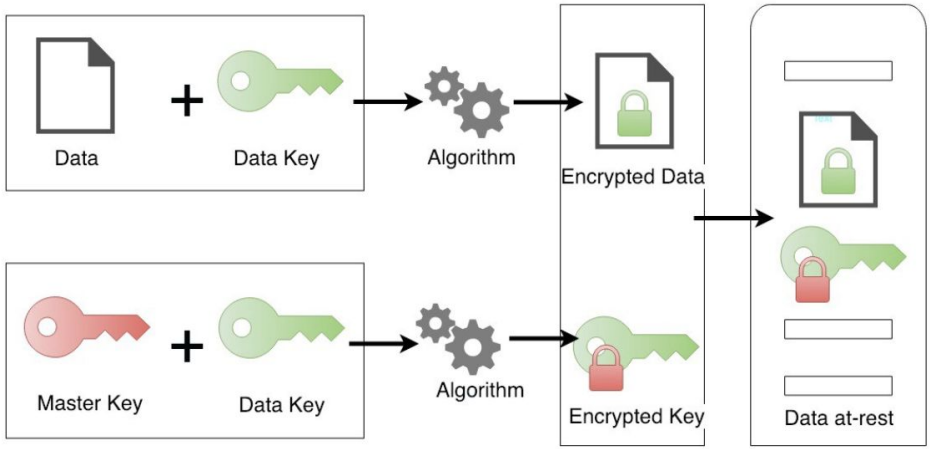


Diagram source: https://docs.aws.amazon.com/kms/latest/developerguide/concepts.html

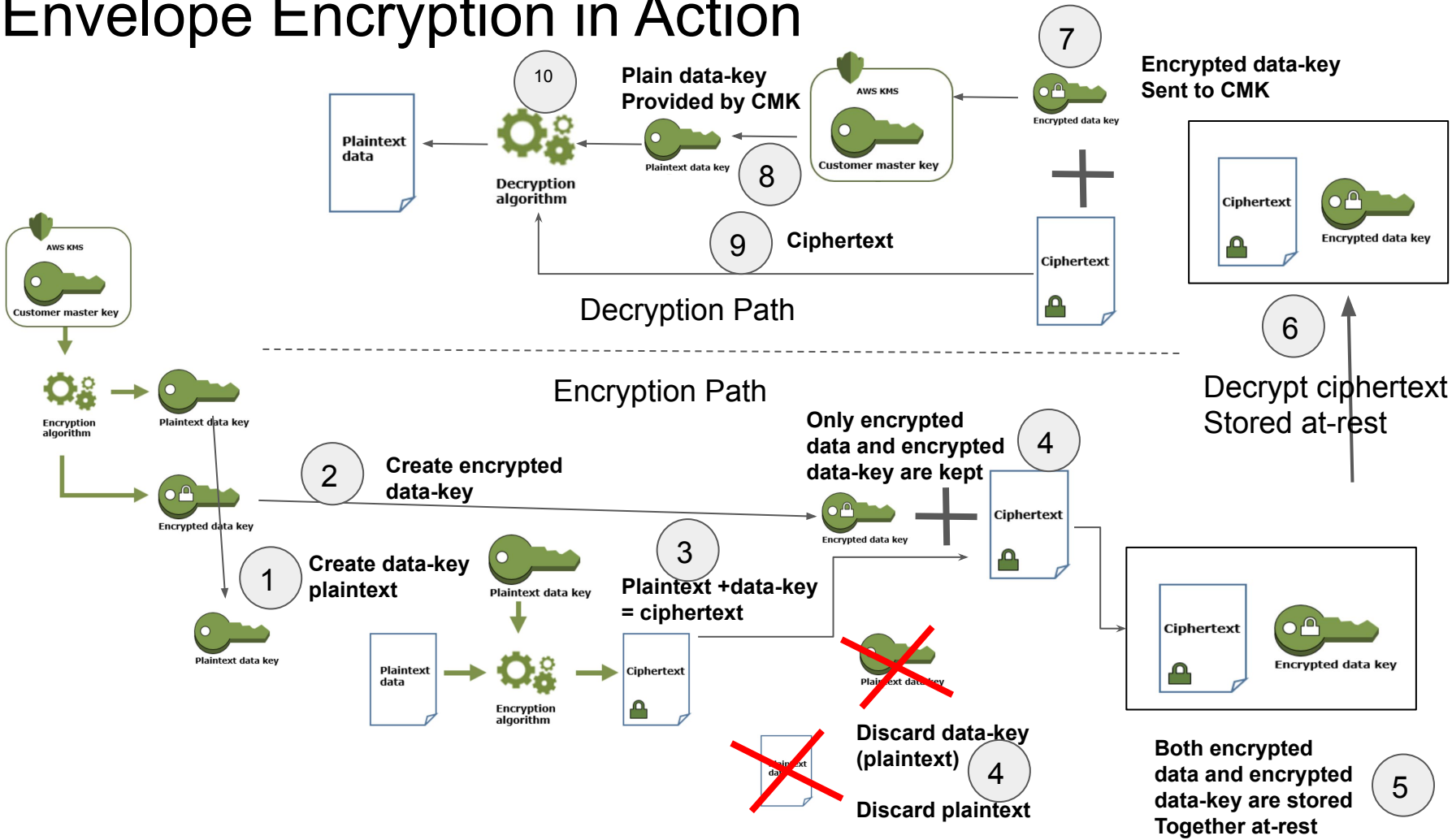
Envelope Encryption



- Master key is used to create data key and encrypted data-key
- Plaintext data is encrypted with data-key
- This results in ciphertext
- Discard data-key & plaintext and store only the ciphertext and encrypted data-key
- Master key itself can be encrypted with a hierarchy of keys
- At the time of decryption, encrypted data-key is provided to Master key, which provides the original data-key
- Now the data-key will be used to decrypt the encrypted data/ciphertext

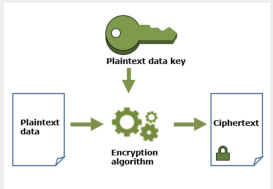
Diagram source: AWS

Envelope Encryption in Action



Client Vs Server Side Encryption

Client side App performs the crypto operation



Client Side

Client side App generates plaintext data

Ciphertext

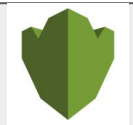
HTTP or TLS

Plaintext over TLS

HTTPS

AWS Storage services (S3, EBS, RDS etc)

Server Side



AWS KMS Service

AWS Storage services (S3, EBS, RDS etc) integrated with KMS

AWS KMS Demo

- Create Customer Managed CMK
- Use AWS CLI to generate data-key & encrypted data-key & save them locally
- Use openssl library to encrypt a file using data-key and save the encrypted/cipher file
- Discard data-key and plain-text
- Use AWS CLI to get the data-key using the encrypted data-key & save the data-key
- Use Openssl library to decrypt the encrypted file
- Verify that decrypt operation yields with the original plaintext
- Commands used for the demo: Assumes AWS CLI tool is installed
- aws kms generate-data-key --key-id alias/Demo --key-spec AES_256 --region us-east-1
- echo <unencrypted data-key> | base64 --decode > datakey
- echo <encrypted data-key> | base64 --decode > encrypted-datakey
- openssl enc -in ./data-file.txt -out ./encrypted-data-file.txt -e -aes256 -k fileb://./datakey -iter 1000
- aws kms decrypt --ciphertext-blob fileb://./encrypted-datakey --region us-east-1
- echo <unencrypted data-key> | base64 --decode > datakey
- openssl enc -in ./encrypted-data-file.txt -out ./unencrypted-data-file.txt1 -d -aes256 -k fileb://./datakey -iter 1000