

AWS Cloud Security Foundations– AWS-CSF

Dr. Ibrahim Gomaa

Lecture 9: Protecting Data in your Application



Lecture 9: Protecting data in your application

Topics

- Protect data at rest
- Amazon S3 protection features
- Protection through encryption
- Protect data in transit
- Best practices to protect data in Amazon S3
- Additional data protection services

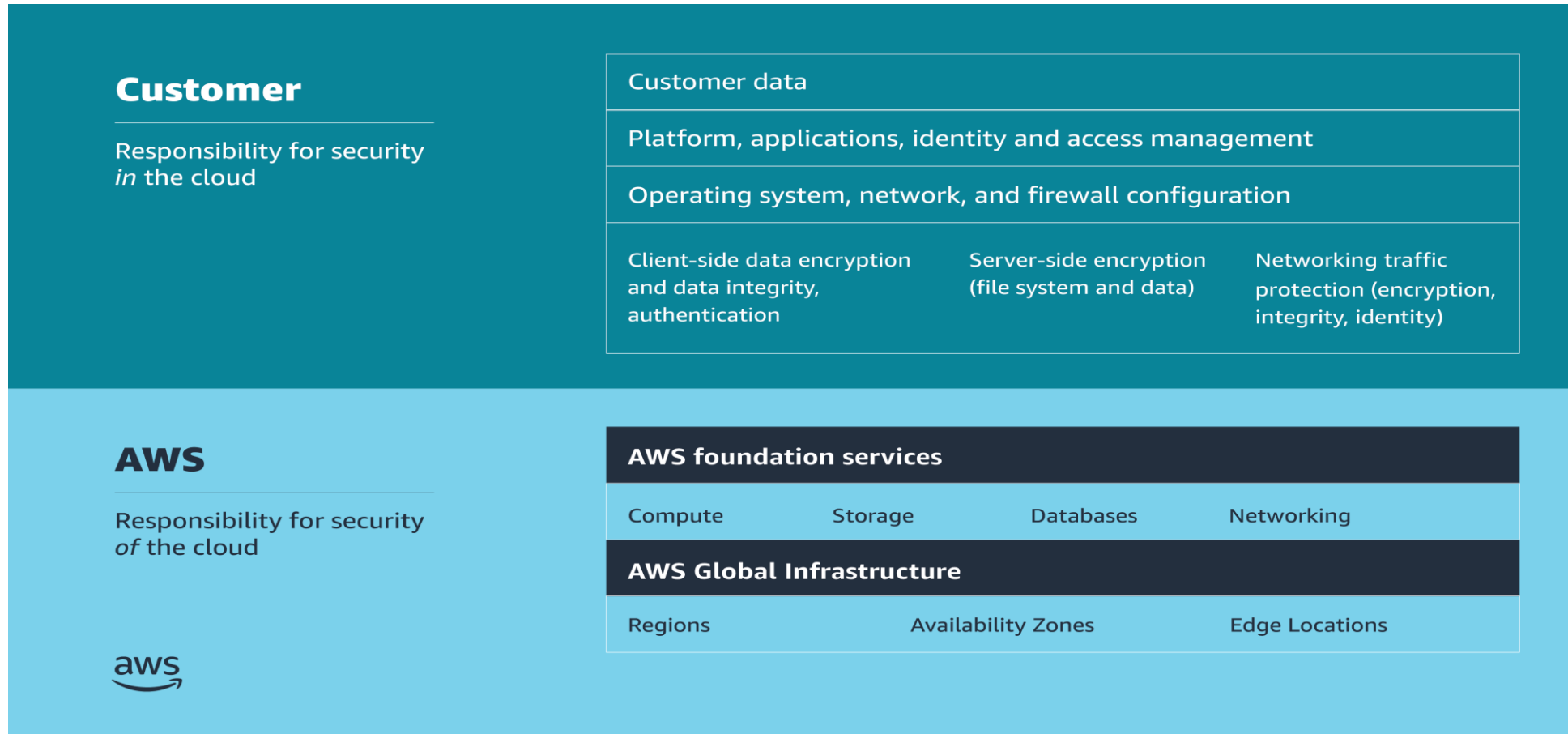
Lab

- Encrypting Data at Rest by Using AWS KMS

Protect data at rest



Shared responsibility model



Why protect data at rest?

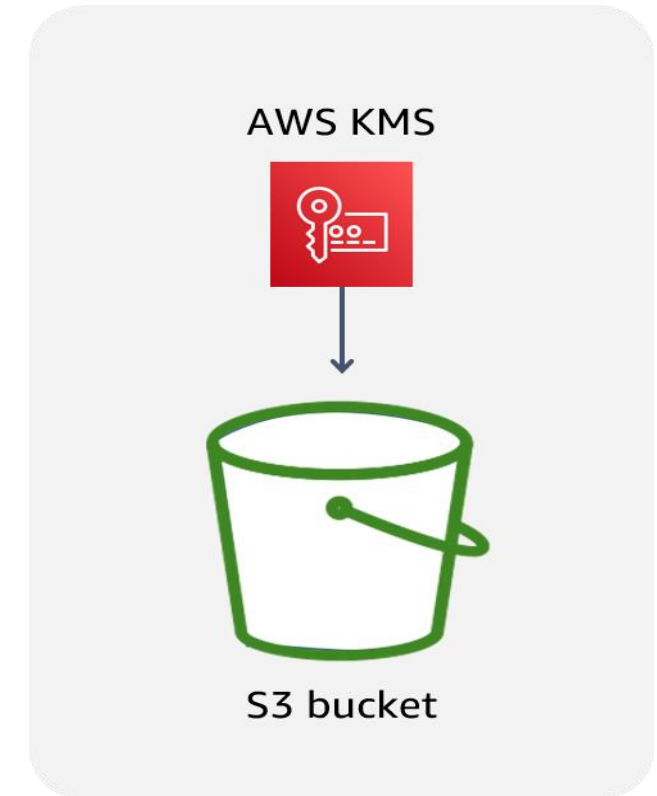
- Common scenarios
 - Information disclosure
 - Data integrity compromise
 - Accidental or malicious deletion
 - System, hardware, and software availability
- Extra layer of protection if your system is compromised



Data at rest in Amazon S3

Data stored in Amazon S3 is private by default and requires AWS credentials for access.

- Use bucket policies for granular access to objects.
- Consider encrypting data at rest.



Granting permissions

Identity based
(Attached to an *IAM principal*)

Resource	Get	Put	List
Bucket X	ALLOW	ALLOW	ALLOW
Bucket Y	N/A	N/A	ALLOW

Resource based
(Attached to an *AWS resource*)

Bucket X	User	Get	Put	List
	Bob	ALLOW	DENY	ALLOW
Bucket Y	User	Get	Put	List
	Bob	ALLOW	N/A	ALLOW

Can Bob GET, PUT, or LIST for bucket X?

Can Bob GET or LIST for bucket Y?

Amazon S3 protection features



Amazon S3 Block Public Access

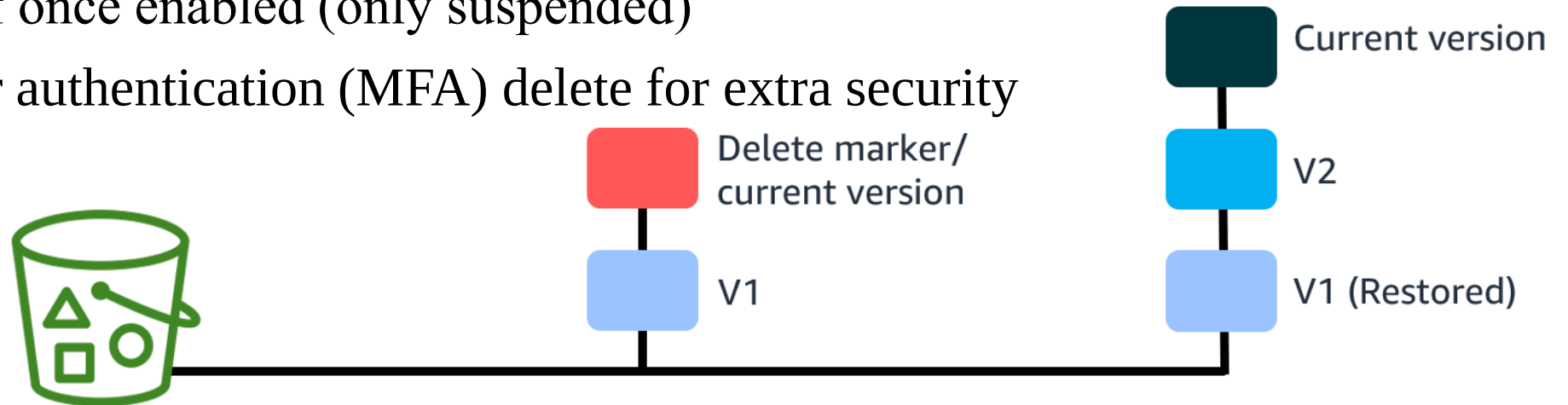
- Helps you manage public access to Amazon S3 resources
- Has four settings:
 - **Block Public Acls:** Block public access granted by new ACLs.
 - **Ignore Public Acls:** Block public access granted by any ACLs.
 - **Block Public Policy:** Block public access granted by new public bucket policies.
 - **Restrict Public Buckets:** Block public and cross-account access by any public bucket policies.



Bucket with block public access settings

Amazon S3 Versioning

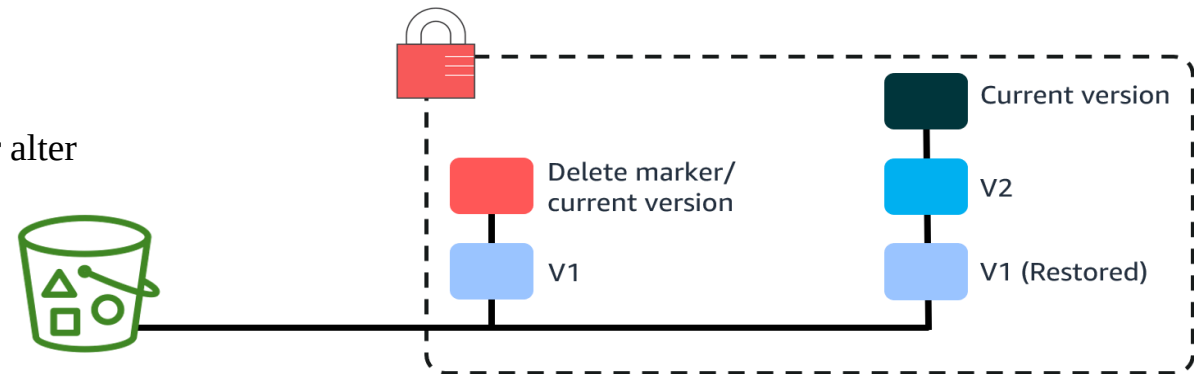
- Creates a new version with every upload
- Protects from unintended deletion
- Provides retrieval of deleted objects
- Can be used with lifecycle policies for cost savings
- Can't be turned off once enabled (only suspended)
- Offers multi-factor authentication (MFA) delete for extra security



Amazon S3 Object Lock

Object Lock prevents an object version from being deleted or overwritten for a fixed amount of time or indefinitely

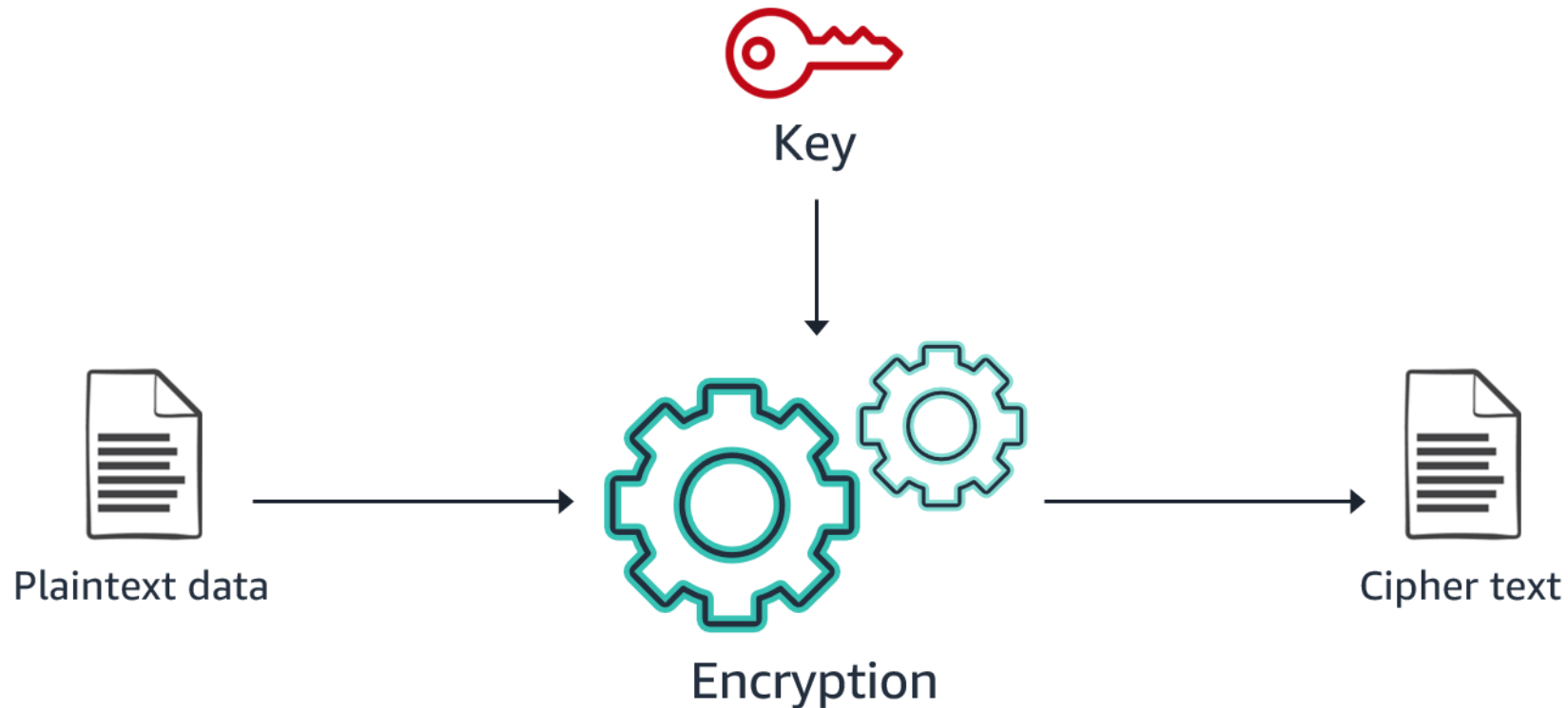
- Stores objects by using the write-once-read-many (WORM) model
- Works only in versioned buckets
- Provides the ability to manage object retention (A **retention period** specifies a fixed period during which an object remains locked. During this period, your object is WORM-protected and can't be overwritten or deleted. A **legal hold** provides the same protection as a retention period but has no expiration date. Instead, remains in place until you explicitly remove it)
- Provides two retention modes:
 - **Governance** users can't overwrite or delete an object version or alter its lock settings unless they have special permissions
 - **Compliance** a protected object version can't be overwritten or deleted by any user, including the root user in your AWS account.



Protection through encryption



Encryption: What, how, and why



Comparing client-side and server-side encryption

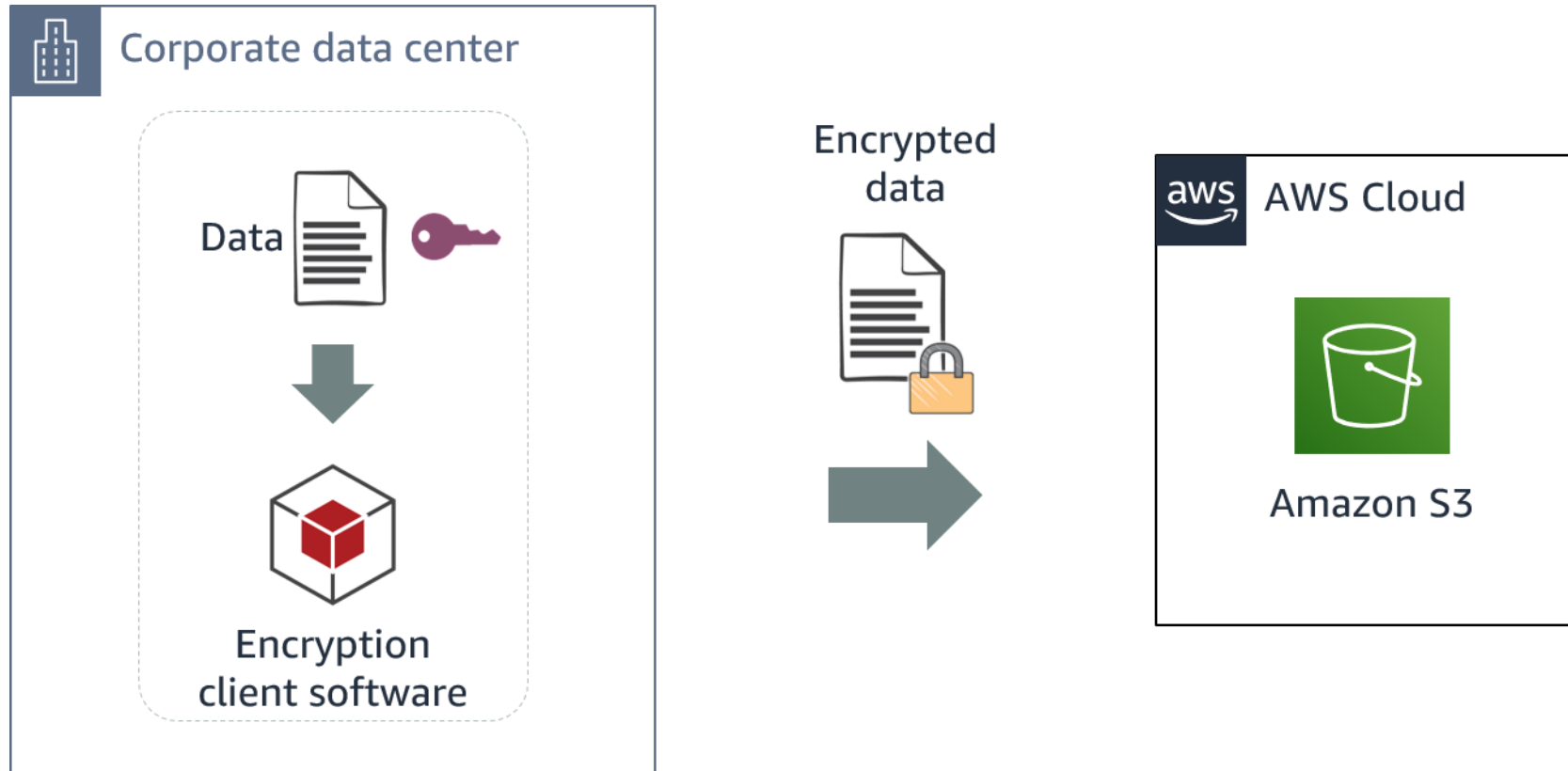
Client-side encryption (CSE)

- Your application encrypts data before sending it to AWS.
- Data is stored in its encrypted state.
- The keys and algorithms are known only to you.

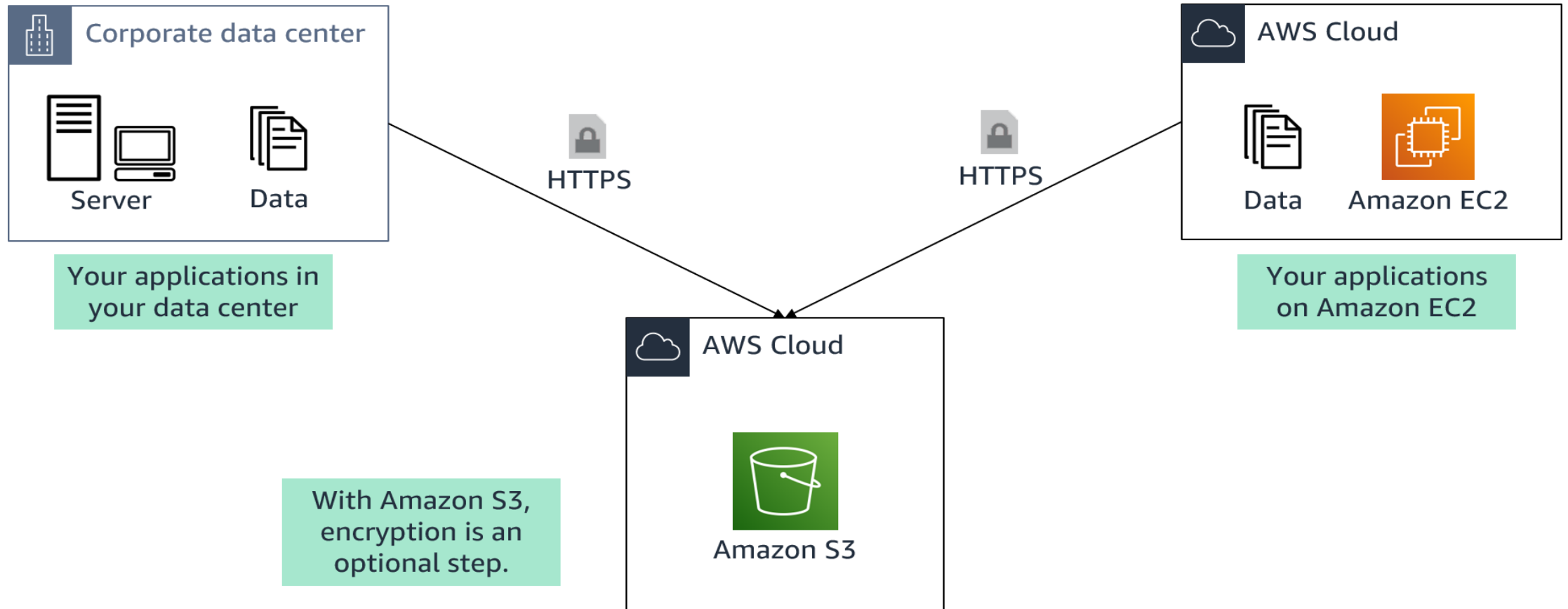
Server-side encryption (SSE)

- AWS encrypts data on your behalf after receiving it.
- The process is transparent to the user.

Client-side encryption



AWS server-side encryption



Types of Amazon S3 server-side encryption

SSE with customer-provided keys

SSE-C

- You retain control of the keys.
- Amazon S3 doesn't store the encryption keys that you provide.

SSE with Amazon S3 managed keys

SSE-S3

- AWS manages the keys.
- The encrypted data and keys are stored in separate hosts.



SSE with AWS KMS keys

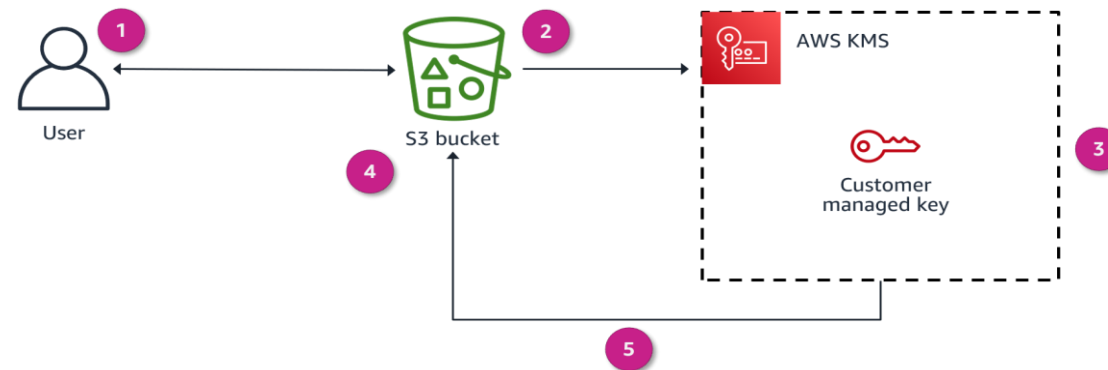
SSE-KMS

- AWS manages the keys.
- An envelope key is used for added protection.

Encryption overview

This diagram explains how encryption happens when uploading a file to Amazon S3:

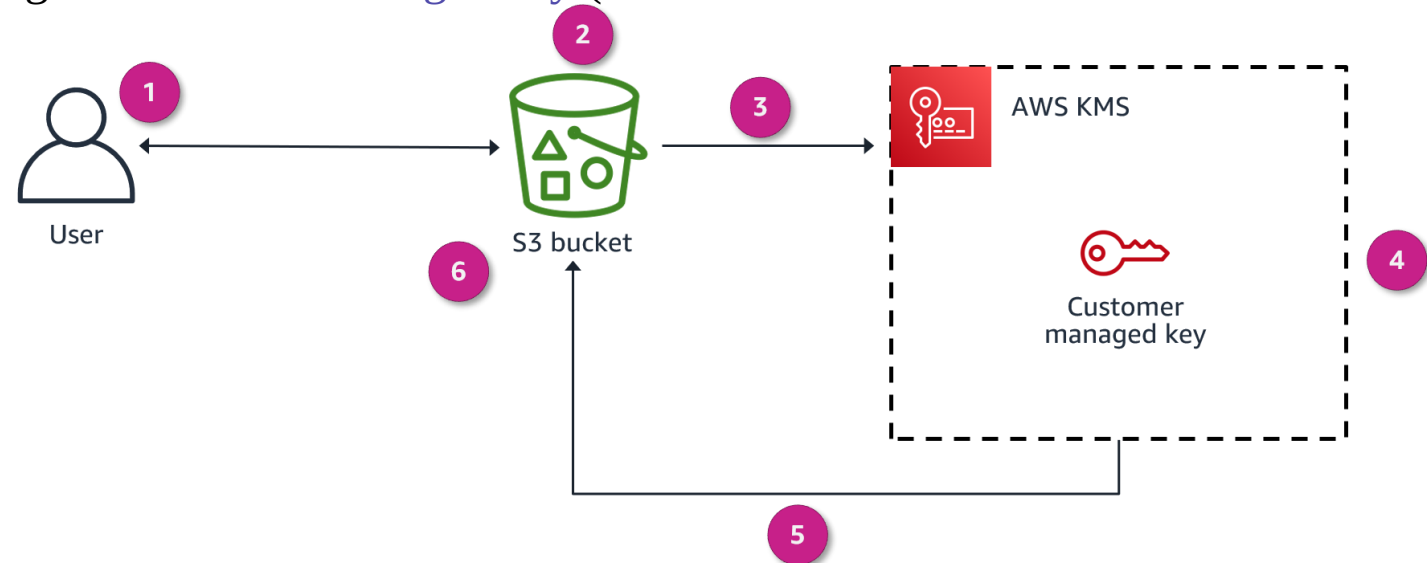
1. You request to upload a file and store it as an encrypted object in an S3 bucket.
2. Amazon S3 requests a data key from **AWS KMS** to use to encrypt the file.
3. AWS KMS generates a **plaintext data key** and **encrypts the data key** by using the **customer managed key**.
4. AWS KMS sends both copies of the data key to Amazon S3.
5. Amazon S3 encrypts the object by using the plaintext data key, stores the object, and then deletes the plaintext data key. The encrypted key is kept in the object metadata.



Decryption overview

Now, this diagram explains what happens in the same scenario (SSE-KMS) when the user requests to open an encrypted object that is stored in Amazon S3:

1. You request to open the object.
2. Amazon S3 notices that the requested object is encrypted.
3. Amazon S3 sends the **encrypted copy of the data key** that the object is encrypted with to AWS KMS.
4. AWS KMS then **decrypts the data key** by using the **customer managed key** (which never leaves the AWS KMS service).
5. AWS KMS then sends the plaintext data key back to Amazon S3.
6. Finally, Amazon S3 decrypts the ciphertext of the data object, allows you to open the object, and deletes the plaintext copy of the data key

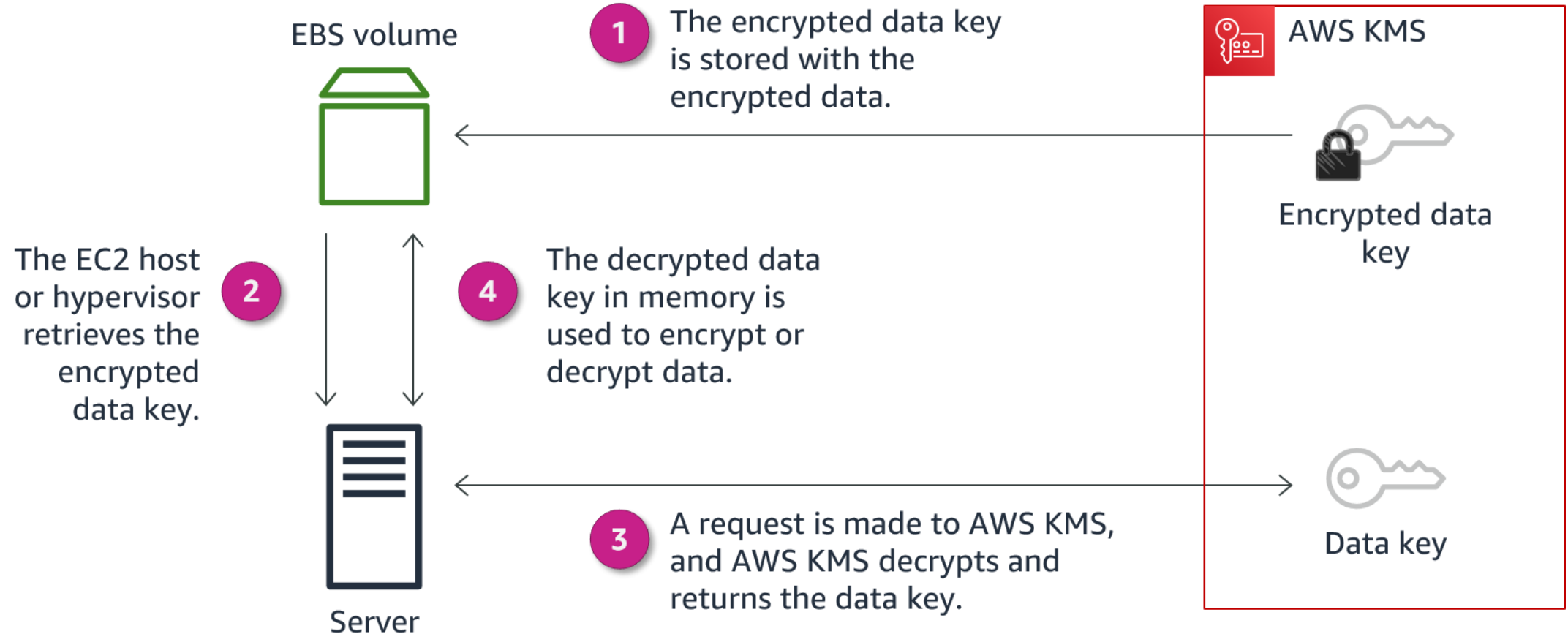


AWS Key Management Service (AWS KMS)

- Provides the ability to create and manage cryptographic keys
- Uses hardware security modules (HSMs) to protect your keys
- Is integrated with other AWS services
- Provides the ability to set usage policies to determine which users can use keys

AWS Key Management
Service (AWS KMS)

AWS KMS example with Amazon EBS



Protect data in transit

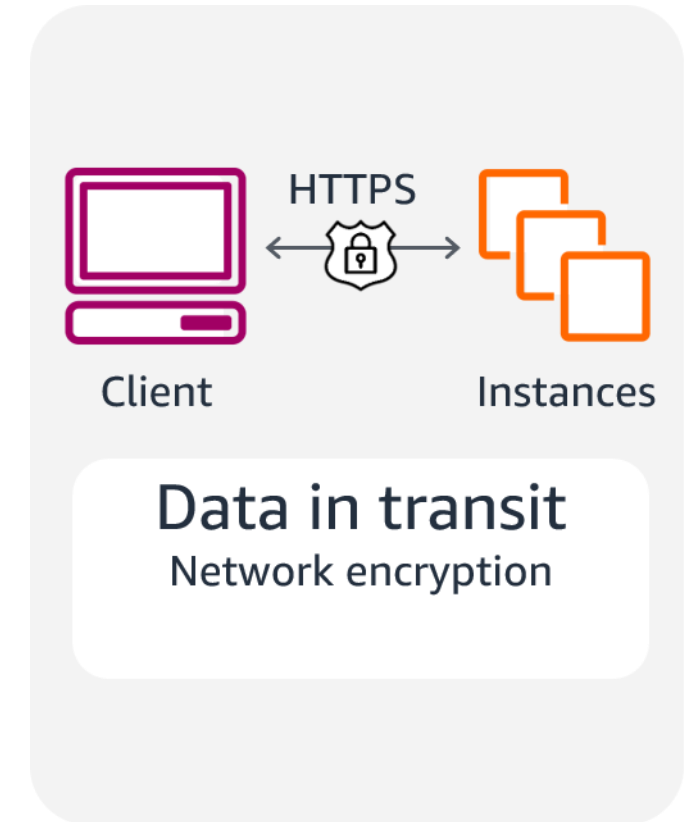


Why protect data in transit?

- Your communications might go through the public internet.
- What are the risks that data in transit is exposed to?
 - Accidental information disclosures
 - Data integrity compromises
 - Identity compromises
 - Man-in-the-middle (MITM) attacks
 - Identity spoofing

Protecting data in transit

- Use Secure Sockets Layer (SSL) endpoints over Transport Layer Security (TLS) (HTTPS).
- Use encryption.
- Use Amazon Virtual Private Cloud (Amazon VPC) endpoints to limit access to your bucket.



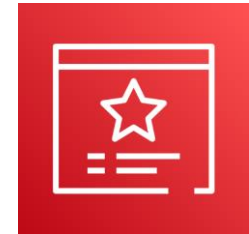
Protecting remote connections to servers

- Remote Desktop Protocol (RDP) is typically used for Windows servers.
 - RDP establishes an underlying SSL/TLS connection.
 - For better security, issue a trusted X.509 certificate.
 - Don't use the default self-signed certificates.
- Secure Shell (SSH) is typically used for Linux servers.
 - SSH establishes a secure communication channel.
 - Use tunneling to protect the application session in transit.
 - Don't allow the root user to use an SSH terminal.
 - Be sure that all users log in with an SSH key pair, and then deactivate password authentication.



AWS Certificate Manager (ACM)

- Provides a single interface to manage both public and private certificates
- Makes it easy to deploy certificates
- Protects and stores private certificates
- Minimizes downtime and outages with automatic renewals



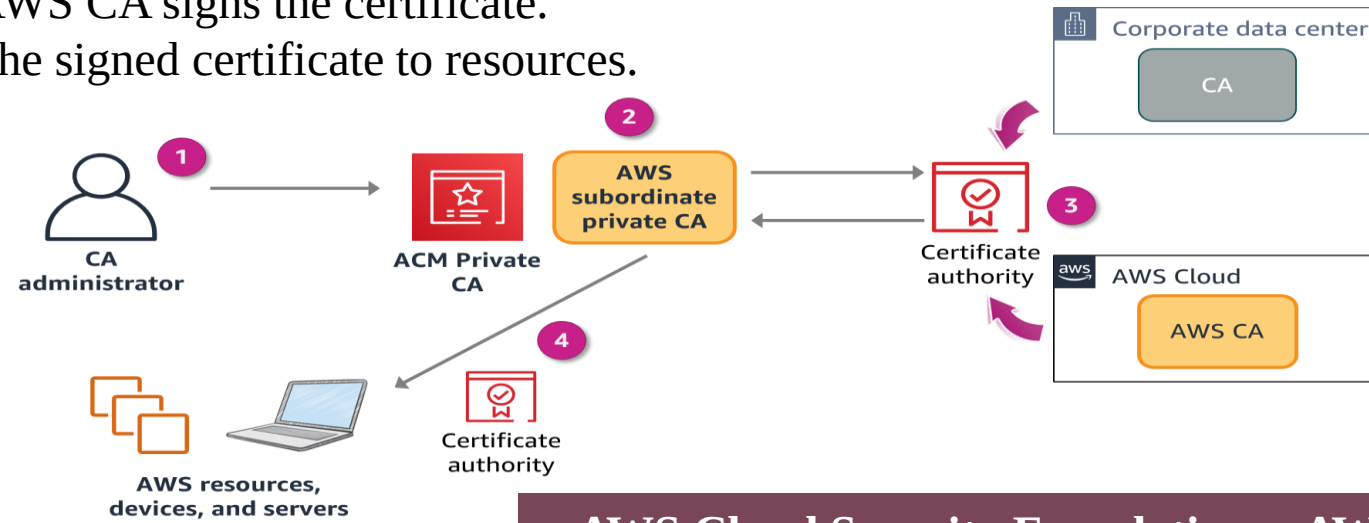
AWS Certificate
Manager (ACM)

AWS Certificate Manager Private Certificate Authority

By using **AWS Certificate Manager Private Certificate Authority**, you can create **private certificate authority (CA)** hierarchies, including **root** and **subordinate CAs**, without the investment and maintenance costs of operating an on-premises CA

The diagram shows ACM Private CA in action:

1. The CA administrator creates a subordinate CA to issue certificates.
2. The subordinate CA sends the certificate to be signed.
3. The on-premises CA or AWS CA signs the certificate.
4. ACM Private CA issues the signed certificate to resources.



ACM Private CA considerations



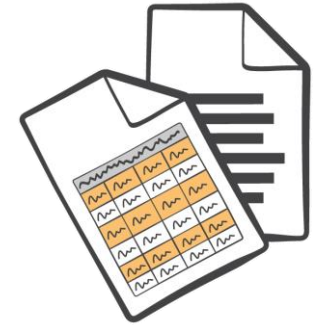
Hardware
security
modules (HSMs)



IAM policies for
access control



Certificate
revocation
list



Generated audit
reports

Best practices to protect data in Amazon S3



Presigned URLs

All objects in an S3 bucket are **private by default**. Only the object owner has permissions to access these objects. However, the **object owner** can optionally share objects with others by creating a **presigned URL**, using their own security credentials, to grant time-limited permissions to upload or download the objects

- Generate an S3 Presigned URL, and use it to upload files (objects).
- A presigned URL uses three parameters to limit user access:
 - **Bucket**: Bucket that the object is in (or will be in)
 - **Key**: Name of the object
 - **Expires**: Amount of time that the URL is valid

Security considerations and best practices (1 of 2)

- Consider encryption of data at rest, and enforce encryption of data in transit.
- Ensure that your S3 buckets use the correct policies and are not publicly accessible.
- Use the principle of least privilege.
- Enable MFA delete for buckets.
- Enforce encryption for each PUT request.
- Use presigned URLs for applications that refer to Amazon S3 objects.

Security considerations and best practices (2 of 2)

- To encrypt all objects, set default encryption on a bucket.
- If using S3 Object Lock, use the appropriate retention mode.
- Use S3 Block Public Access.
- Upload data to Amazon S3 over SSH File Transfer Protocol (SFTP) through AWS Transfer for SFTP.
- Enable S3 Versioning.

Additional data protection services



AWS Secrets Manager

AWS Secrets Manager is a service that makes it easier for you to manage 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 console, AWS CLI, or API and SDKs.

- Is a secure and scalable method for managing access to secrets
- Is a way to meet regulatory and compliance requirements
- Rotates secrets safely without breaking applications
- Audits and monitors the lifecycle of secrets
- Helps you to avoid putting secrets in code or config files

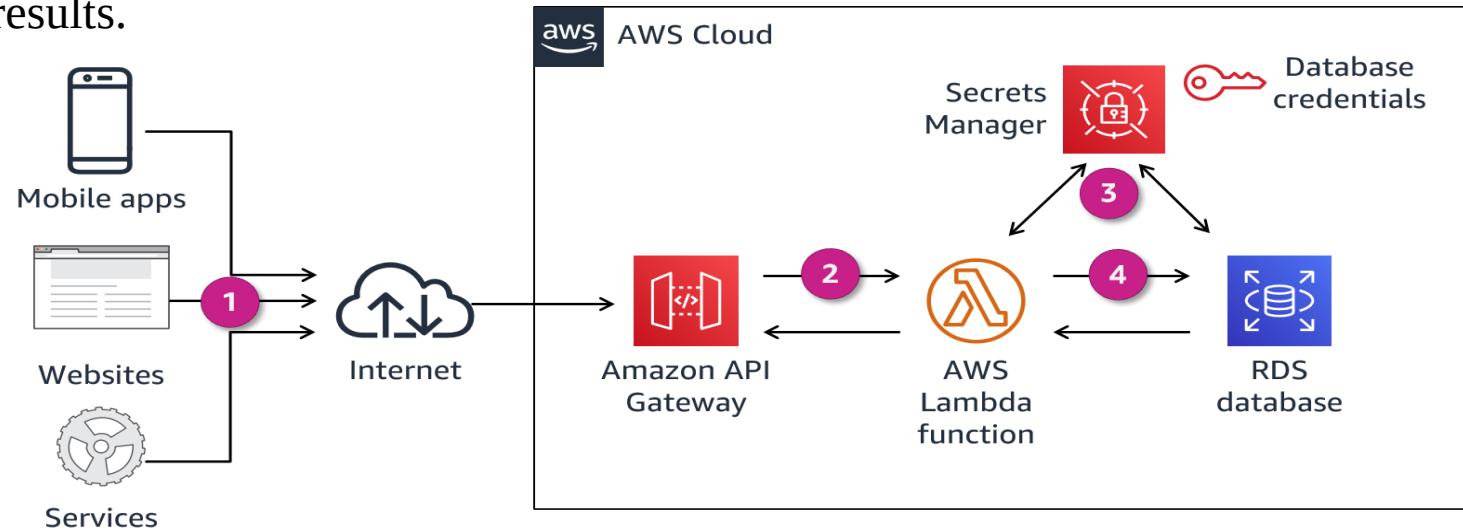


AWS Secrets Manager

Using Secrets Manager

The steps in this workflow are as follows:

1. Clients call the RESTful API hosted on Amazon API Gateway.
2. API Gateway runs the Lambda function.
3. The Lambda function retrieves the database secrets using the Secrets Manager API. Secrets Manager retrieves the secret, decrypts the protected secret text, and returns it to the function over a secured channel.
4. The Lambda function connects to the Amazon RDS database by using database secrets from Secrets Manager and returns the query results.



Rotating secrets

The **rotation function** does the work of rotating the secret. The process to rotate a secret has **four steps**, which correspond to four steps in **the Lambda rotation function**. Secrets Manager uses staging labels to label secret versions during rotation:

1. The function contacts the DB for new credentials.
2. Secrets Manager stores the new credentials with the AWSPENDING label.
3. The new credentials are tested.
4. The new credentials are made the default with the AWSCURRENT label.



Built-in Lambda function for Amazon RDS DB engine credentials
Custom Lambda function for other credentials

Amazon Macie

Amazon Macie is a security service that uses machine learning to automatically discover, classify, and protect sensitive data in AWS

- Recognizes sensitive data such as personally identifiable information (PII), financial information, encryption keys, and credentials
- Allows for custom-defined data types
- Protects data stored in Amazon S3 by monitoring resource policies and ACLs
- Provides full API coverage for management
- Integrates with AWS Organizations



Amazon Macie

Classify data and monitor its access permissions

- Macie scans and evaluates bucket objects for policy violations and sensitive data.
- Macie generates findings in real-time for the following:
 - Buckets that are public
 - Buckets that are not encrypted
 - Buckets that are shared or replicated with AWS accounts outside of a customer's organization

Macie summary

Amazon Macie X

SummaryFindingsBy bucketBy typeBy job

S3 bucketsJobs

UsageSettingsGeneralCustom data identifiersAccounts

Summary of S3 buckets

Last updated: 05-15-2020 17:32:54

Total S3 buckets38

Total storage722.8 GB

Object count63.74m

AccountEnter account

Public0%

Unencrypted87%

Shared3%

0% of buckets are publicly accessible

87% of buckets are unencrypted

3% of buckets are shared

Publicly accessible0

Not publicly accessible38

Unencrypted33

Not publicly accessible5

Shared outside1

Not shared outside37

Publicly world writeable0

Publicly world readable0

Encrypted with SSE-S31

Encrypted with SSE-KMS4

Shared inside1

Not shared36

Macie findings

Showing **791** of **791** **254** **343** **194**

Findings 

This table lists findings for your organization. Select a finding to show its details. You can also filter, group, and sort findings based on specific fields and field values.

Actions 

Saved filters/Auto-archive 

Current  Add filter criteria

☐	Finding type	Resources affected	Updated at	Count
☐	 SensitiveData:S3Object/Financial	mybucket/123456789012/us-east-1/0a2eac77...jsonl.gz	5 minutes ago	1
☐	 SensitiveData:S3Object/Financial	mybucket /123456789012/us-east-1/123ec7e3...jsonl.gz	an hour ago	1
☐	 SensitiveData:S3Object/Financial	mybucket /123456789012/us-east-1/b42eac70...jsonl.gz	2 hours ago	1
☐	 SensitiveData:S3Object/Financial	mybucket /123456789012/us-east-1/0e11ac6a...jsonl.gz	3 hours ago	1
☐	 SensitiveData:S3Object/Financial	mybucket /123456789012/us-east-1/7ae54y00...jsonl.gz	4 hours ago	1
☐	 SensitiveData:S3Object/Credentials	mybucket /123456789012/us-east-1/z45w834...jsonl.gz	6 hours ago	1

Thanks