Amazon S3 - Security

Amazon S3 – Object Encryption

You can encrypt objects in S3 buckets using one of 4 methods:

Server-Side Encryption (SSE)

- **Server-Side Encryption with Amazon S3-Managed Keys (SSE-S3)** – *Enabled by default*
  - Encrypts S3 objects using keys handled, managed, and owned by AWS
- **Server-Side Encryption with KMS Keys stored in AWS KMS (SSE-KMS)**
  - Leverages AWS Key Management Service (AWS KMS) to manage encryption keys
- **Server-Side Encryption with Customer-Provided Keys (SSE-C)**
  - Use when you want to manage your own encryption keys

Client-Side Encryption

- Encryption is handled by the client before uploading the data to S3

It's important to understand which encryption method to use in different situations for the exam.

Amazon S3 Encryption – SSE-S3

- Encryption using keys handled, managed, and owned by AWS
- Object is encrypted server-side
- Encryption type is **AES-256**
- Must set header:
  `"x-amz-server-side-encryption": "AES256"`
- **Enabled by default** for new buckets and new objects

How it works (diagram explanation)

1. The user uploads an object to Amazon S3 using HTTP(S), optionally including the encryption header.
2. Amazon S3 encrypts the object on the server side using an encryption key that is owned and managed by S3.
3. The encrypted object is stored in the S3 bucket.

Amazon S3 Encryption – SSE-KMS

- Encryption using keys handled and managed by AWS KMS (Key Management Service)
- KMS advantages: user control and ability to audit key usage with CloudTrail
- Object is encrypted server-side
- Must set header:
  `"x-amz-server-side-encryption": "aws:kms"`

How it works (diagram explanation)

1. The user uploads an object to Amazon S3 using HTTP(S), including the encryption header.
2. Amazon S3 uses a KMS key managed in AWS Key Management Service to encrypt the object.
3. The encrypted object is stored in the S3 bucket.

SSE-KMS Limitation

- If you use SSE-KMS, you may be impacted by the **KMS limits**
- When you upload an object, S3 calls the **GenerateDataKey** API from KMS
- When you download an object, S3 calls the **Decrypt** API from KMS
- These operations **count towards the KMS quota per second**, which varies by region:
  - Examples: 5500, 10000, 30000 requests per second
- You can request a **quota increase** using the **Service Quotas Console**

How it works (diagram explanation)

1. Users upload or download objects encrypted with SSE-KMS.
2. For uploads, S3 calls the `GenerateDataKey` API to obtain a data encryption key.
3. For downloads, S3 calls the `Decrypt` API to decrypt the data key.
4. Both actions result in API calls to AWS KMS, consuming KMS API quota.

Amazon S3 Encryption – SSE-C

- Server-Side Encryption using keys **fully managed by the customer**, outside of AWS
- Amazon S3 **does NOT store** the encryption key you provide
- **HTTPS must be used**
- Encryption key must be provided in **HTTP headers for every request**

How it works (diagram explanation)

1. The user uploads an object using HTTPS and includes the encryption key in the request headers.
2. Amazon S3 uses the client-provided key to encrypt the object server-side.
3. The encrypted object is stored in the S3 bucket.
4. Every subsequent request (e.g., download) must also include the same key in the headers.

> *Note: If you lose the encryption key, you lose access to the object. AWS does not store or manage the key.*

Amazon S3 Encryption – Client-Side Encryption

- Use client libraries such as **Amazon S3 Client-Side Encryption Library**
- Clients must **encrypt data themselves before sending** it to Amazon S3
- Clients must **decrypt data themselves** when retrieving from Amazon S3
- The customer fully manages the encryption keys and the encryption/decryption lifecycle

How it works (diagram explanation)

1. The client encrypts the file locally using a key it manages (Client Key).
2. The encrypted file is uploaded to S3 over HTTP(S).
3. S3 simply stores the encrypted file without knowing the key or the plaintext.
4. To retrieve the file, the client downloads it and performs the decryption using the same client-managed key.

> *Note: AWS never sees or stores the key or the unencrypted data in this method.*

Amazon S3 – Encryption in Transit (SSL/TLS)

- **Encryption in flight** is also known as **SSL/TLS**
- Amazon S3 exposes two endpoints:
  - **HTTP Endpoint** – non-encrypted

    - **HTTPS Endpoint** – provides encryption in transit
- **HTTPS is recommended**
- **HTTPS is mandatory for SSE-C**
- Most clients use the **HTTPS endpoint by default**

Amazon S3 – Force Encryption in Transit (`aws:SecureTransport`)

- You can enforce HTTPS-only access to your S3 bucket using a **bucket policy**
- The `aws:SecureTransport` condition key ensures requests using **HTTP are denied**
- This improves security by forcing encryption in transit (SSL/TLS)

Example Bucket Policy

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Principal": "*",
      "Action": "s3:GetObject",
      "Resource": "arn:aws:s3:::my-bucket/*",
      "Condition": {
        "Bool": {
          "aws:SecureTransport": "false"
        }
      }
    }
  ]
}
```

How it works (diagram explanation)

- When a user makes a request over **HTTP**, access is **denied** by the bucket policy.
- When a user uses **HTTPS**, the request is **allowed**.
- This helps ensure that all data in transit is encrypted using SSL/TLS.

Amazon S3 – Default Encryption vs. Bucket Policies

- **SSE-S3 encryption** is automatically applied to new objects stored in the S3 bucket (if configured)
- Optionally, you can **force encryption** using a **bucket policy**, denying any `PUT` request that lacks the proper encryption headers (e.g. for SSE-KMS or SSE-C)

Example Bucket Policies to Force Encryption

1. Deny if encryption header is not `aws:kms`

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
```

```
      "Effect": "Deny",
      "Action": "s3:PutObject",
      "Principal": "*",
      "Resource": "arn:aws:s3:::my-bucket/*",
      "Condition": {
        "StringNotEquals": {
          "s3:x-amz-server-side-encryption": "aws:kms"
        }
      }
    }
  ]
}
```

2. Deny if no encryption header is present at all (e.g., client-side encryption not used)

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Action": "s3:PutObject",
      "Principal": "*",
      "Resource": "arn:aws:s3:::my-bucket/*",
      "Condition": {
        "Null": {
          "s3:x-amz-server-side-encryption-customer-algorithm": "true"
        }
      }
    }
  ]
}
```

> ***Note:*** *Bucket Policies are evaluated* ***before*** *"Default Encryption" settings.*

What is CORS?

- **CORS** stands for **Cross-Origin Resource Sharing**
- **Origin** is defined as: `scheme (protocol)` + `host (domain)` + `port`
  - Example: `https://www.example.com`
    (Port is implied: 443 for HTTPS, 80 for HTTP)
- **Web browser-based mechanism** to allow requests to other origins while visiting the main origin

Origin comparison

- **Same origin** example:
  `http://example.com/app1` and `http://example.com/app2`
- **Different origins** example:
  `http://www.example.com` and `http://other.example.com`

- Requests to other origins will **not be fulfilled** unless the other origin **explicitly allows** them using **CORS headers**
    - Example header: `Access-Control-Allow-Origin`

What is CORS? (Preflight Request Example)

When making cross-origin requests, especially with custom headers or methods (e.g., `PUT`, `DELETE`), the browser performs a **preflight request** using the `OPTIONS` method before the actual request.

Preflight Request Flow

1. **Browser** sends a `OPTIONS` request to the target (cross-origin) server:

OPTIONS / Host: www.other.com Origin: https://www.example.com

2. **Cross-Origin Server** (`https://www.other.com`) responds with CORS headers:

Access-Control-Allow-Origin: https://www.example.com Access-Control-Allow-Methods: GET, PUT, DELETE

3. If the response includes the appropriate headers, the **browser proceeds** with the actual request:

GET / Host: www.other.com Origin: https://www.example.com

4. Since the browser has already received valid **CORS headers**, the request is allowed.

> *This mechanism ensures that cross-origin requests are **explicitly authorized** by the target server, improving security.*

Amazon S3 – CORS

- If a client makes a **cross-origin request** to an S3 bucket, you must configure the **correct CORS headers**
- This is a **popular exam question**
- You can allow:
    - A **specific origin**, or
    - Use `"*"` to allow **all origins**

Example Use Case

- A static website is hosted in the S3 bucket `my-bucket-html`
- That website references images hosted in a different S3 bucket: `my-bucket-assets`

1. Browser requests `index.html` from:

http://my-bucket-html.s3-website.us-west-2.amazonaws.com

2. `index.html` includes an image from:

http://my-bucket-assets.s3-website.us-west-2.amazonaws.com

3. The browser sends a **GET request** with the `Origin` header set to the source bucket.

4. The image bucket (`my-bucket-assets`) must respond with the proper CORS header:

Access-Control-Allow-Origin: http://my-bucket-html.s3-website.us-west-2.amazonaws.com

Without this header, the image will not be loaded due to CORS restrictions.

Amazon S3 – MFA Delete

- **MFA (Multi-Factor Authentication)** requires users to generate a code on a device (e.g. mobile phone or hardware key) before performing sensitive operations on S3

MFA is required to:

- Permanently delete an object version
- Suspend versioning on a bucket

MFA is **not** required to:

- Enable versioning
- List deleted versions

Important Notes

- To use **MFA Delete**, **versioning must be enabled** on the bucket
- **Only the bucket owner (root account)** can enable or disable MFA Delete

S3 Access Logs

- For **audit purposes**, you may want to **log all access** to S3 buckets
- Any request made to S3 — from any account, whether **authorized or denied** — will be logged to another **S3 bucket**
- This data can be **analyzed using data analysis tools**
- The **target logging bucket** must be in the **same AWS region** as the source bucket

Log Format

You can find details about the S3 access log format here:
https://docs.aws.amazon.com/AmazonS3/latest/dev/LogFormat.html

S3 Access Logs: Warning

- **Do not set your logging bucket to be the monitored bucket**
- This will cause a **logging loop**, and as a result:
  - **Your bucket will grow exponentially**

> *Always use a **separate bucket** for logging to avoid recursive log generation.*

Amazon S3 – Pre-Signed URLs

- You can generate **pre-signed URLs** using:
  - **S3 Console**
  - **AWS CLI**
  - **SDK**

URL Expiration

- **S3 Console**: valid from 1 minute up to 720 minutes (12 hours)
- **AWS CLI**: use `--expires-in` parameter (in seconds)
  - Default: 3600 seconds (1 hour)
  - Maximum: 604800 seconds (~7 days)

Permissions

- A pre-signed URL **inherits the permissions** of the IAM user or role that generated it
- It allows access to the object for **GET** or **PUT**, depending on the generated operation

Examples

- Allow only logged-in users to download a premium video from your S3 bucket
- Dynamically generate pre-signed URLs for an ever-changing list of users
- Temporarily allow a user to **upload** a file to a specific location in your S3 bucket

How it works (diagram explanation)

1. The **bucket owner** generates a pre-signed URL from a **private S3 bucket**
2. The **pre-signed URL** is shared with the user
3. The user uses the URL to **download or upload** a file, depending on the operation type

S3 Glacier Vault Lock

- Adopts a **WORM** model (Write Once, Read Many)
- You must **create a Vault Lock Policy**
- Once locked, the policy:
  - **Cannot be changed or deleted**
  - Is enforced for all future operations

Use Cases

- Ensures **compliance** with regulatory requirements
- Enables **data retention** by preventing deletion or modification of archived data

S3 Object Lock (versioning must be enabled)

- Adopts a **WORM** model (Write Once, Read Many)
- Blocks deletion of an object version for a **specified amount of time**

Retention Modes

Compliance Mode:

- Object versions **cannot** be overwritten or deleted by **any user**, including the **root user**
- Retention mode and period **cannot be changed or shortened**

Governance Mode:

- Most users **cannot** overwrite or delete an object version or change its lock settings
- Some users with **special permissions** can modify retention or delete the object

Retention Period

- Protects the object for a **fixed period**
- The period **can be extended**, but **not shortened**

Legal Hold

- Protects the object **indefinitely**, regardless of the retention period
- Can be applied or removed using the `s3:PutObjectLegalHold` permission

S3 – Access Points

- **Access Points** simplify security management for S3 Buckets, especially at scale.
- Each Access Point has:
  - Its own **DNS name** (Internet Origin or VPC Origin)
  - An **access point policy** (similar to bucket policy)

Use Case Example

You can define multiple access points for the same bucket, each with specific permissions:

- **Finance Access Point**:
  - Grants **read/write** access to `/finance` prefix
  - Used by **Finance** team
- **Sales Access Point**:
  - Grants **read/write** access to `/sales` prefix
  - Used by **Sales** team
- **Analytics Access Point**:
  - Grants **read-only** access to the **entire bucket**
  - Used by **Analytics** team

This setup allows centralized data storage with **segmented access policies** per team, improving **security and maintainability**.

S3 – Access Points (VPC Origin)

- You can define an **S3 Access Point** to be accessible **only from within a VPC**
- To use it, you must create a **VPC Endpoint**:
  - Can be a **Gateway Endpoint** or **Interface Endpoint**
- The **VPC Endpoint Policy** must allow:
  - Access to the **target S3 bucket**
  - Access to the **Access Point**

Access Path

1. EC2 Instance in VPC
2. Connects through VPC Endpoint (with proper policy)
3. Hits Access Point (with its own policy)
4. Accesses objects in the S3 Bucket

Example VPC Endpoint Policy

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Principal": "*",
      "Action": [
        "s3:GetObject"
      ],
      "Effect": "Allow",
```

```
      "Resource": [
        "arn:aws:s3:::awsexamplebucket1/*",
        "arn:aws:s3:us-west-2:123456789012:accesspoint/example-vpc-ap/object/*"
      ]
    }
  ]
}
```

S3 Object Lambda

- Use **AWS Lambda Functions** to modify an object **before it's returned** to the caller application.
- Only one S3 bucket is required, with:
  - An **S3 Access Point**
  - An **S3 Object Lambda Access Point**

Use Cases

- **Redacting** personally identifiable information (PII) for analytics or non-production use.
- **Converting** data formats (e.g. XML to JSON).
- **Resizing or watermarking** images dynamically using caller-specific data (e.g. the requesting user).

Flow Description

1. The object is stored in a single S3 bucket.
2. A **supporting S3 Access Point** gives the Object Lambda access.
3. Different applications (e.g. analytics, marketing) access the object via **S3 Object Lambda Access Points**.
4. Lambda functions (e.g. redact, enrich) transform the object before returning it to the caller.
5. Transformed content is returned (e.g. redacted or enriched object).

This allows dynamic, context-aware object manipulation without duplicating data.