06. S3 Encryption

1. Chiffrement au repos

Le dilemme à l'examen repose toujours sur deux critères : **Qui gère la clé ?** et **Où se fait le chiffrement ?**

A. Server-Side Encryption (SSE) : AWS chiffre pour vous

Les données arrivent brutes chez AWS, et c'est S3 qui s'occupe de les chiffrer avant de les écrire sur le disque.

- **SSE-S3 (Chiffrement géré par S3)**
  - **Le concept :** AWS gère tout. La clé est créée, stockée et gérée automatiquement par AWS.
  - **Algorithme :** AES-256.
  - **Pourquoi ?** C'est la solution par défaut (gratuite et zéro gestion).
  - **Ce qu'il faut savoir pour l'examen :** Depuis 2023, **SSE-S3 est activé par défaut** sur tous les nouveaux buckets S3. Si vous n'en spécifiez aucun, c'est celui-là qui s'applique.
  - **Header HTTP à l'upload :** `"x-amz-server-side-encryption": "AES256"`

- **SSE-KMS (Chiffrement via AWS KMS)**
  - **Le concept :** Vous utilisez le service AWS Key Management Service (KMS) pour gérer les clés. Vous pouvez utiliser la clé AWS par défaut (`aws/s3`) ou créer votre propre clé personnalisée (Customer Managed Key).
  - **Pourquoi ?** Pour avoir un **contrôle d'accès strict** (qui a le droit d'utiliser la clé) et un **suivi d'audit (CloudTrail)** complet de chaque chiffrement/déchiffrement.
  - **Ce qu'il faut savoir pour l'examen :** Génère des coûts (appels API KMS).
    - Attention aux quotas de requêtes KMS en cas de fort trafic.
    - Pour déchiffrer un objet, l'utilisateur a besoin de la permission S3 **ET** de la permission `kms:Decrypt`.
  - **Header HTTP à l'upload :** `"x-amz-server-side-encryption": "aws:kms"`

- **SSE-C (Chiffrement avec clés fournies par le client)**
  - **Le concept :** Vous possédez et gérez la clé secrète, mais vous la passez à S3 dans la requête HTTPS pour qu'AWS fasse le travail de chiffrement. **AWS ne stocke jamais votre clé**. Une fois l'opération finie, AWS jette la clé de sa mémoire vive.
  - **Pourquoi ?** Pour les entreprises qui ont des obligations de conformité strictes interdisant de confier la gestion des clés à un tiers (même AWS).
  - **Ce qu'il faut savoir pour l'examen :** Si vous perdez la clé, vos données sont perdues à jamais. AWS a récemment renforcé la sécurité en désactivant le support SSE-C par défaut sur les nouveaux buckets (il faut l'autoriser explicitement via l'API

si besoin).
    - **Header HTTP :** Vous devez envoyer la clé dans chaque requête HTTP (`x-amz-server-side-encryption-customer-key`).

B. Client-Side Encryption : Vous chiffrez avant d'envoyer

- **Le concept :** Vos données sont chiffrées **au sein même de votre application** (par exemple avec le SDK AWS / S3 Encryption Client) avant de franchir le réseau. AWS reçoit un fichier qui est déjà un bloc illisible.
- **Pourquoi ?** Sécurité maximale "End-to-End". AWS n'a aucun moyen de voir ce qu'il y a dans votre fichier, même s'ils le voulaient.

2. Chiffrement en transit (Encryption in Transit)

Le chiffrement en transit garantit que personne ne peut intercepter vos données entre votre client et les serveurs d'AWS.

- **Le comment :** S3 supporte nativement les protocoles **HTTP** et **HTTPS (TLS)**.
- **Pour l'examen (Piège classique) :** Comment forcer le chiffrement en transit ? On utilise une **Bucket Policy** avec une condition explicite qui rejette (`Deny`) toutes les connexions non sécurisées.

Voici à quoi ressemble la règle absolue à connaître pour le SAA-C03 :

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

Comme `aws:SecureTransport` est `false` (donc du HTTP classique), l'accès est bloqué (`Deny`).

Revision #3
Created 2026-06-02 09:30:07 UTC by Victor
Updated 2026-06-02 09:53:26 UTC by Victor