

L4 – Security of Computer Systems

Evaluating data encryption on locally hosted open- source Amazon S3 compatible object storage platform

Evangelos Dimoulis

Department of Computer Science & Engineering

University of Ioannina

February 2021



Presentation

- Object Storage Systems
- Amazon Simple Storage Service
- MinIO Object Storage
- Threat Model
- Encryption Schemes
- Experimental Setup
- Performance Evaluation



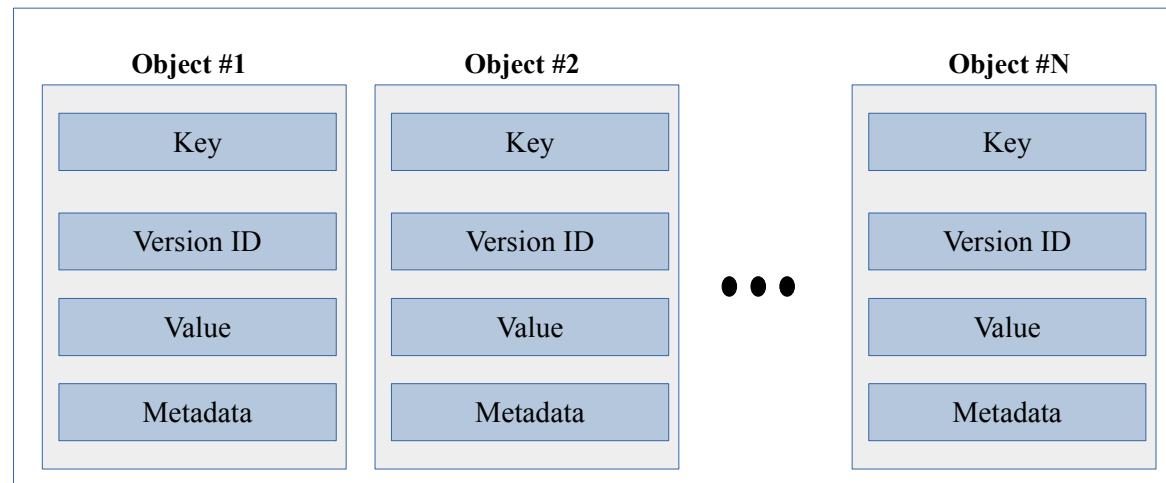
Object Storage Systems

- Processing data as discrete units called objects.
- Objects are variable-length size and can be used to store all types of data (e.g., database records, medical records, system backup).
- Benefits:
 - Scalability: increase number of storage nodes.
 - Reduced complexity: removing unnecessary hierarchies.
 - Pay as you go: pricing is usually volume-based.
 - Cloud compatible



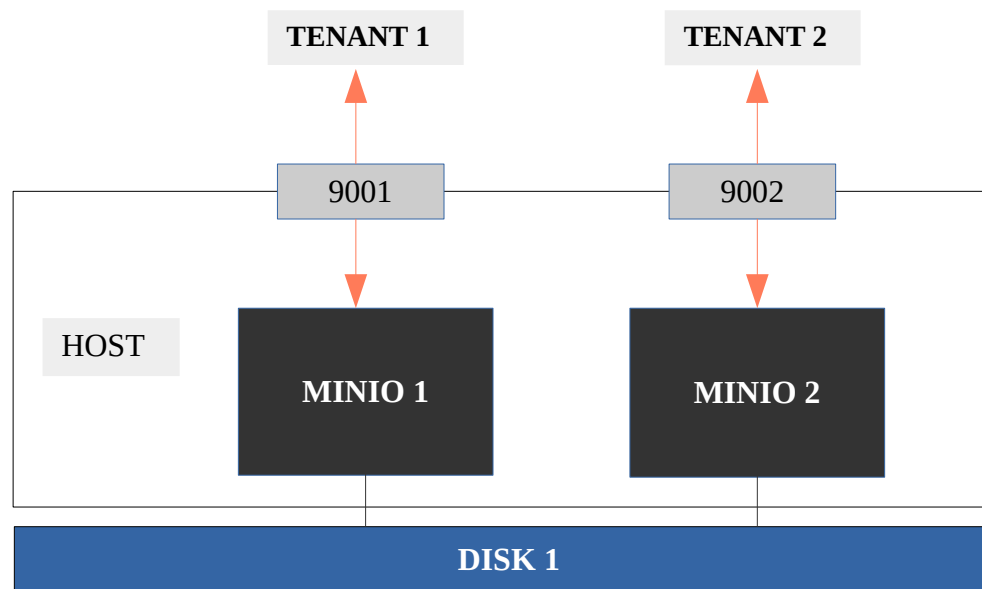
Amazon Simple Storage Service (Amazon S3)

- Amazon S3 is an object storage service provides high scalability, data availability, security and performance.
- Objects are stored and managed into buckets.
- Provides REST API for managing objects and buckets.
- Bucket architecture:



MinIO

- MinIO: High performance object storage released under Apache License v2.0
- API compatible with Amazon S3 cloud storage service.
- Deployment: a) Standalone, b) Distributed, and c) Cloud Scale.
- Multi-Tenant Standalone Deployment:



Accessing MinIO with TLS

- Transport Security Layer (TLS) enforces essential security guarantees:
 - Encryption: hides data being transferred.
 - Authentication: ensure identity of communicating parties.
 - Integrity: verifies the content of the transferred data.
- Communication with MinIO servers and clients is protected via TLS.
- MinIO server provides public certificate signed by a private key to establish communication.
- The certificate is provided through a Certificate Authority (CA) or is self-signed.



MinIO Encryption

- MinIO supports two types of Server-Side Encryption (SSE):
 - SSE-C: en/decrypting objects with client provided keys.
 - SSE-S3: object encryption keys are managed by a Key Management System (KMS).
- For any encrypted object in the server there exist three keys:
 - OEK: unique key stored encrypted to en/decrypt an object
 - KEK: key to en/decrypt the OEK, is not stored anywhere
 - EK: external key, either client provided or managed by KMS
- Encryption algorithm: AES-256-GCM with keys 256 bits long.



Threat Model

- We focus on three kind of principals:
 - User: someone who want to securely store data online
 - Third-Party Service: an entity that claims or requests access to the user's data.
 - Cloud Storage Provider: a MinIO object storage server instance.
- Vulnerability: the third-party service gain access to an object's en/decryption keys through side-channel attacks e.g., cold boot attacks, and/or storage server is compromised.
- Goal: revoke access to the third-party service by introducing client side encryption schemes.



Client Side Encryption

```
1.  /* export encrypt */
2.  func encrypt(fileToEncrypt string, fileName string, pathToKey string) *C.char{
3.
4.      encrypted_file := fileName + ".bin"
5.      plaintext, err := ioutil.ReadFile(fileToEncrypt)
6.
7.      /* Read key file content and convert it to string */
8.      content, err := ioutil.ReadFile(pathToKey)
9.      keyString := string(content)
10.
11.     /* Decode key in order to get the key as type byte[] */
12.     key, _ := hex.DecodeString(keyString)
13.
14.     /* Create a new Cipher Block from the key */
15.     block, err := aes.NewCipher(key)
16.
17.     /* Create a new GCM */
18.     gcm, err := cipher.NewGCM(block)
19.
20.     /* Never use more than 2^32 random nonces with a given key */
21.     nonce := make([]byte, gcm.NonceSize())
22.     if _, err := io.ReadFull(rand.Reader, nonce); err != nil {
23.         log.Fatal(err)
24.     }
25.     /* Encrypt the data using GCM Seal function */
26.     ciphertext := gcm.Seal(nonce, nonce, plaintext, nil)
27.
28.     /* Save back to file */
29.     err = ioutil.WriteFile(encrypted_file, ciphertext, 0777)
30.
31.     /* Returns a pointer to the encrypted file */
32.     return C.CString(encrypted_file)
33. }
```

```
1.  /* Loading C shared encryption library */
2.  lib = cdll.LoadLibrary("libencrypt.so")
3.
4.  /* Structure that handles type convert between modules */
5.  class go_string(Structure):
6.      _fields_ = [
7.          ("p", c_char_p),
8.          ("n", c_int)]
9.
10. /* Encryption function wrapper */
11. def encrypt(file_path, file_name, AES_KEY):
12.     lib.encrypt.restype = c_char_p
13.     fp = go_string(c_char_p(file_path.encode('utf-8')), len(file_path))
14.     key = go_string(c_char_p(AES_KEY.encode('utf-8')), len(AES_KEY))
15.     file_name = go_string(c_char_p(file_name.encode('utf-8')), len(file_name))
16.     encrypted_file_name = lib.encrypt(fp, file_name, key)
17.
18.     /* Returns the name of the encrypted file */
19.     return encrypted_file_name
```



MinIO Server Side Encryption (SSE)

- Python client code for SSE-C encryption with client provided key.

```
1.  /* Encodes key and returns SSE configuration */
2.  def sse_encryption(key):
3.
4.      f = open(key, "r")
5.      key_str = f.read()
6.      key_str = key_str.replace("\n", " ")
7.      key = key_str.encode('ascii')
8.      SSE = SseCustomerKey(key)
9.
10.     return (SSE)
11.
12.  /* Custom HTTP client inquiring valid SSL certificate */
13.  httpClient = urllib3.PoolManager(
14.      timeout=urllib3.Timeout.DEFAULT_TIMEOUT,
15.      cert_reqs='CERT_REQUIRED',
16.      ca_certs=PUBLIC_CERTIFICATE,
17.      retries=urllib3.Retry(
18.          total=5,
19.          backoff_factor=0.2,
20.          status_forcelist=[500, 502, 503, 504]
21.      )
22.  )
23.  /* MinIO Client initialized with access information */
24.  client = Minio(MINIO_URL,
25.      access_key='minio',
26.      secret_key='minio123',
27.      secure=True,
28.      http_client=httpClient)
29.
30.  /* SSE Customer provided key encryption */
31.  SSE = sse_encryption(key_path)
32.
33.  /* Clients putting object to server bucket with SSE encryption */
34.  client.fput_object( BUCKET_NAME, file_name, file_path, sse=SSE )
```



Key Rotation

- MinIO supports a key rotation mechanism through a special HTTP COPY Request.
- Key Rotation:
 - Provide initial external secret key and a newly derived secret key.
 - Decrypt OEK with the old key and re-encrypt with the newly provided key.

```
1.  /* Source object customer provided key */
2.  SSE_SRC = sse_encryption(OLD_AES_KEY)
3.
4.  /* Destination Object SSE Customer provided key encryption */
5.  SSE_DST = sse_encryption(NEW_AES_KEY)
6.
7.  /* Copy the object to the same bucket using a different key.
8.   * Object does not exit the server side using this method.
9.   */
10. result = client.copy_object(
11.     BUCKET_NAME,
12.     file_name,
13.     CopySource(BUCKET_NAME, file_name, ssec=SSE_SRC),
14.     sse=SSE_DST,
15. )
```

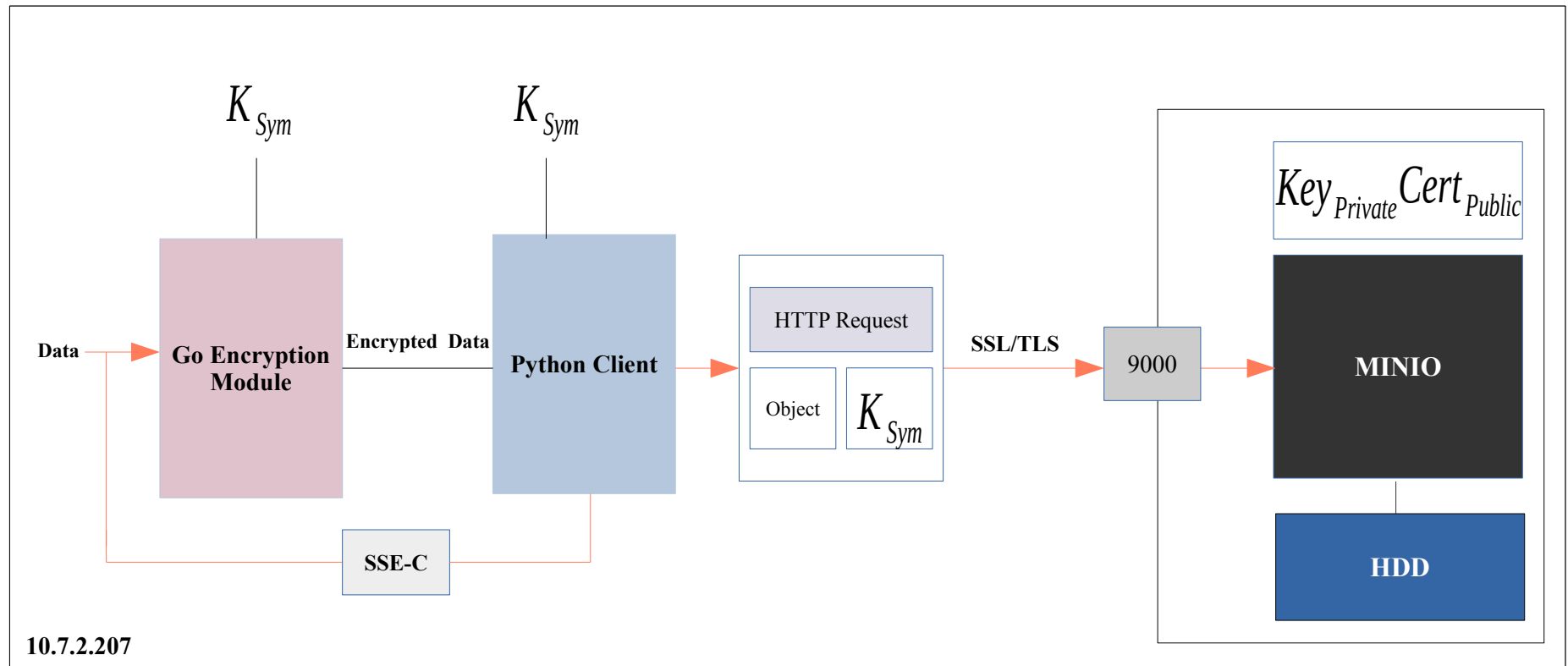


Experimental Setup

- Standalone deployment with a single storage server.
- Server equipped with 4 cores Intel (R) Core (TM) i5-4590 at 3.30 GHz and 7.7 GB of memory.
- Debian 10 64-bit Linux distribution 4.19.0-13-amd64
- OpenSSL version 1.1.1d.
- Go version go1.15.6 linux/amd64 and latest MinIO built from source.



System Architecture



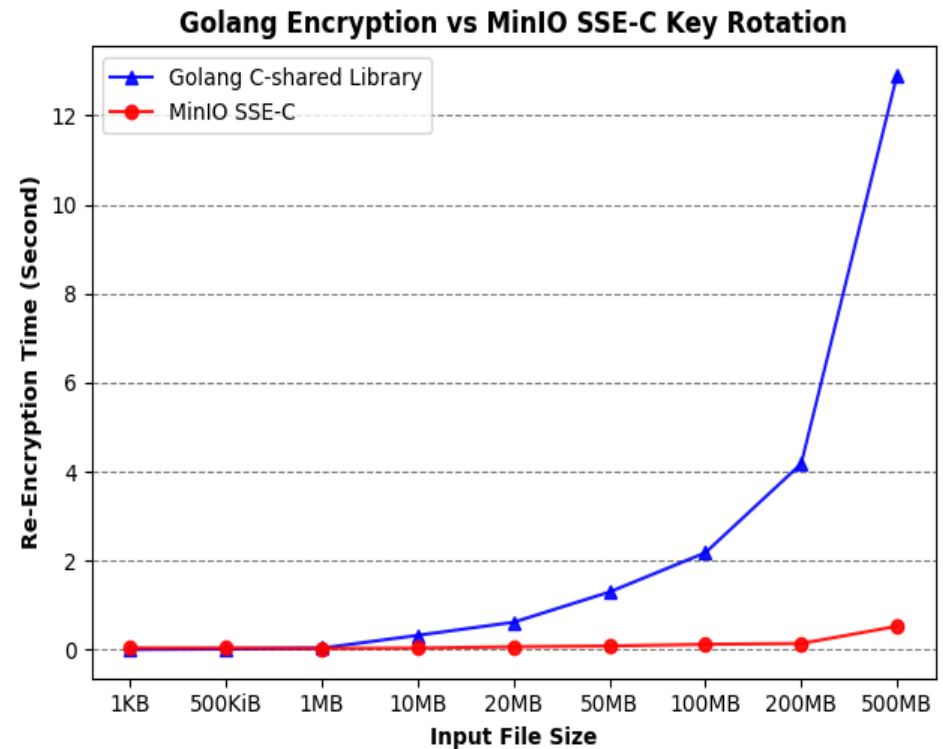
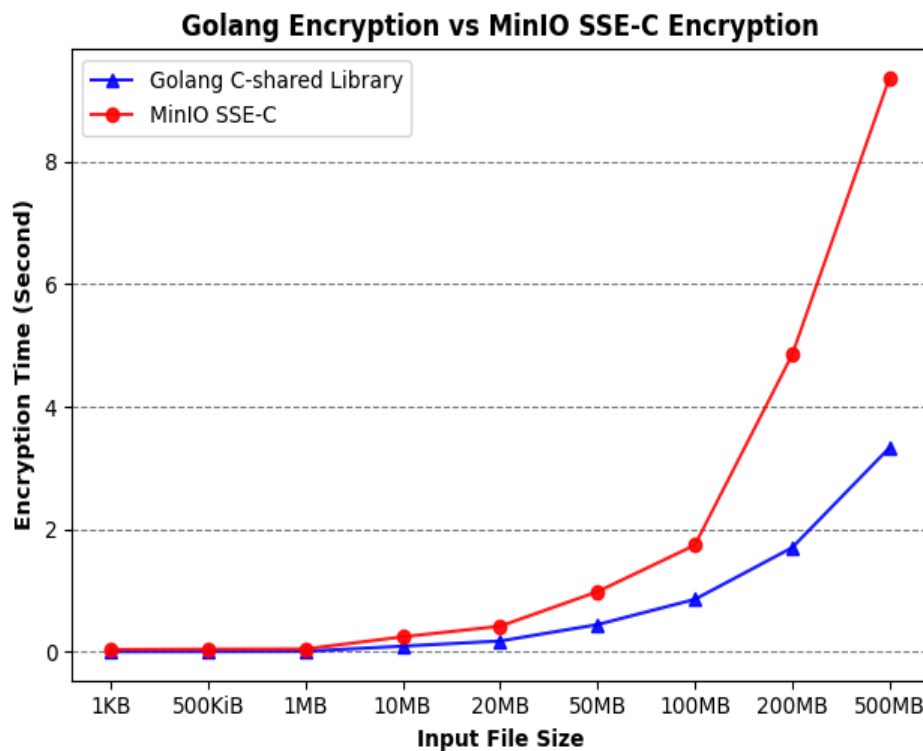
Performance Evaluation (1)

- Benchmark
 - Binary files ranging from 1KB up to 500 MB worth of data.
 - Runned tests 100 times for each input file.
 - Plotting median for encryption metrics and system resource utilization.
- Performance Metrics
 - Python's psutil for system resource utilization.
 - Encryption Time (second): time spent for encrypting input files.
 - Re-Encryption Time (second): time spent for client side re-encryption and SSE-C key rotation.
 - CPU Usage (%) and RAM Usage (%): system resource usage percentage during evaluation.



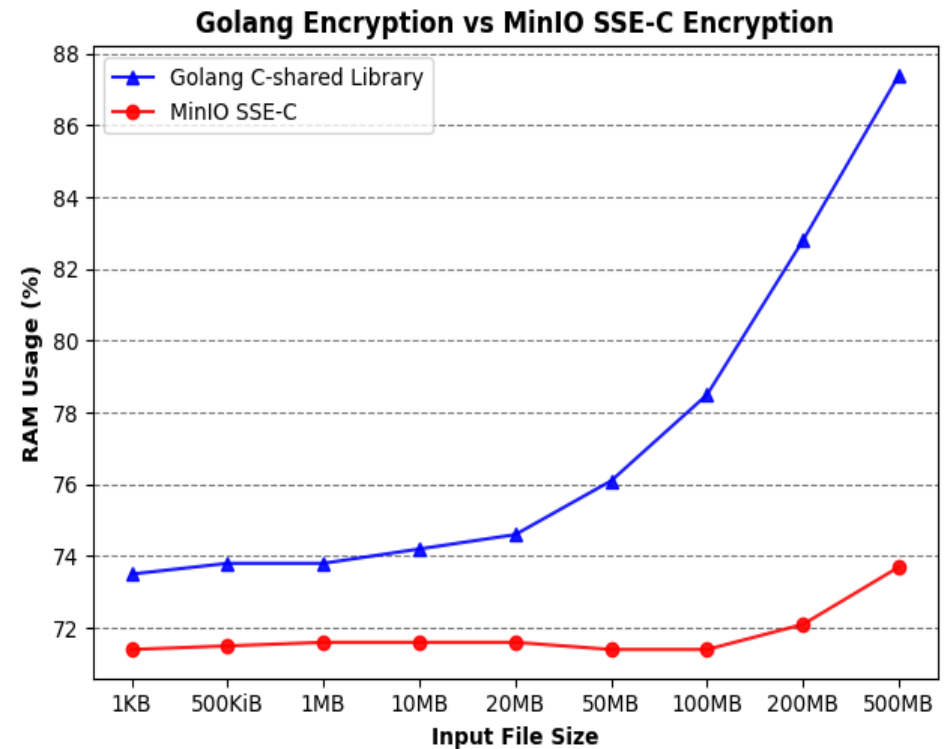
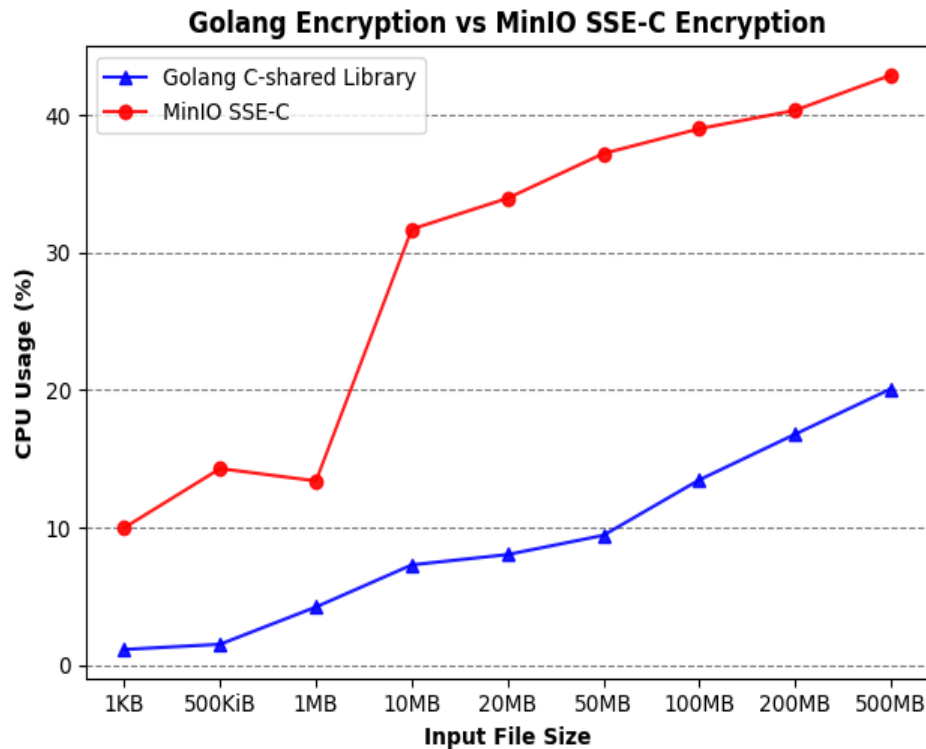
Performance Evaluation (2)

- Encryption Time and Re-Encryption Time of Golang client-side scheme and MinIO SSE-C.



Performance Evaluation (3)

- CPU and RAM utilization of Golang encryption and MinIO SSE-C encryption.



Questions

