Privacy-Preserving Multi-Cloud GRC Framework Using Homomorphic Encryption for Secure Threat Intelligence Aggregation

Author: Yashwardhan Singh

Date: June 6, 2025

1. Abstract

Because many organizations now choose multi-cloud, ensuring secure and compliant cross-cloud threat sharing is very important. Traditionally, GRC solutions are inadequate for guaranteeing that private data is safe and in compliance with various international privacy standards. This report recommends and reviews a method for handling privacy in the multi-cloud GRC approach with Homomorphic Encryption.

The framework was written in Python with help from the TenSEAL library and examined by using a cloud provider simulation in the EU, US and India. All providers use the same public context to scramble their threat ratings and then the GRC server only receives the aggregated results without unscrambling them. The last step in the process is for the decryptor with the secret key to uncover insights and ensure that all the data stays fully private.

To enact real privacy practices, it was decided to prevent data from regions not under the authority of specified jurisdictions. To evaluate the feasibility of HE-based GRC operations, records were made of CPU time, the amount of memory needed and total running time. The evidence shows that aggregating threat scores using HE is possible and permissible, giving a good option for privacy-minded collaboration between clouds.

2. Introduction

Nowadays, many companies use different cloud systems to address their daily, legal and strategic requirements. Although multi-cloud adoption makes management more flexible, it also leads to challenges in handling Governance, Risk and Compliance (GRC), mainly when sharing sensitive threat intelligence takes place among different countries.

The process of centralizing data in traditional GRC frameworks is likely to break data privacy regulations like GDPR and CCPA. Passing unprotected threat data to main databases can cause breaches of regulations or risk disclosure from inside the company.

To deal with this, we designed a privacy-protecting multi-cloud GRC framework that applies Homomorphic Encryption (HE). Thanks to HE, analyses can be done right on encrypted information, preventing data from being exposed. Due to this, data is kept private even as it moves from one computer to another.

We apply Python and TenSEAL library to build a sample architecture where several cloud services secure their threat intelligence information and send it to a central system where all the information is collected. If only authorized persons with the secret key can read the data, it remains safe and follows regulations.

To solve the problem, a proposal was made with jurisdiction-based access features, the simulation of compliance checks and processing time measurements.

3. Problem Statement

Multiple-cloud usage by businesses results in tough challenges related to Governance, Risk and Compliance (GRC) rules. When data is stored and managed across different cloud providers for expanding capabilities and safeguards, it causes various issues related to data privacy, regulations and the compliance process.

It is impossible for traditional GRC systems to perform useful analysis without raw data which doesn’t meet the requirements of laws such as the GDPR and others. Sending sensitive

information or compliance logs between different places or providers could break regulations, harm data security and have legal problems.

Also, different security strategies, encryption mechanisms and regulations used by cloud providers make it tough to achieve secure analysis with data from all of them. The important challenges are:

- Ensuring that the data remains private during every step of the process.
- Obeying the rules for sharing and accessing data set by regions.
- Blocking unauthorized attempts to get into sensitive information about security threats.
- Using ways that keep the number of calculations low for security procedures.

For this reason, a GRC framework that safeguards privacy is needed to connect all processes and ensure compliance without giving away details of the underlying data. It is necessary for the solution to apply access controls that are aware of different jurisdictions, perform encryption and provide light reporting methods, all suitable for practical use.

This paper suggests and shows an HE-powered GRC system that allows secure data processing and safe exchanges of threat intelligence content among cloud entities.

4. Proposed Framework

- To overcome the issues of privacy, security and meeting regulations, we suggest using Homomorphic Encryption (HE) so that multiple cloud providers can safely and confidentially work with security intelligence.

- The framework is made up of several key elements as follows:

  Cloud providers take their threat intelligence data and encrypt it locally, using a shared public Homomorphic Encryption (HE) context that is distributed. Consequently, all sensitive data stays private until the transmitting process is complete.

  A homomorphic computation engine allows data and risk analysis to be done within the encrypted data.
  The Compliance Monitor module tests the grouped encrypted data against set policies and laws, ensuring organizations stay compliant at all times and without revealing their internal data.

  The Key Management Service (KMS) safely produces, delivers and maintains encryption keys so that only the right people are allowed to decrypt the sensitive outcomes.
  It logs all encryption and compliance review data, making these logs unchangeable to reinforce investigations and meet audit requirements.

  Researchers start by encrypting the threat scores from their cloud and passing the encrypted data to the main server. Homomorphic aggregation is carried out by the server and those with the secret key are the only ones who can decrypt the final results. By using this design, data remains confidential until the very end while still making it possible for different clouds to collaborate.

5. Technical Details

It gives a description of the architecture, modules, encryption and performance related to GRC in the cloud by using Homomorphic Encryption (HE). Security, regulatory obligations and efficient threat intelligence circulation in several clouds are given main importance during implementation.

5.1 Technology Stack

- Python 3 was used in the development of this project.

- The Encryption Library uses the TenSEAL (BFV scheme) in the Homomorphic Computation Applications.
- For this, I'm using profiling tools such as time and psutil to see CPU, wall time and memory usage.
- The simulation is set up using local Ubuntu which works with three cloud environments (EU, US, IN) and one central GRC server.

5.2 Encryption Context Generation

Encryption starts by producing a homomorphic encryption context using the BFV system.

```
def generate_context():
    context = ts.context(
        ts.SCHEME_TYPE.BFV,
        poly_modulus_degree=8192,
        plain_modulus=1032193
    )
    context.generate_galois_keys()
    context.generate_relin_keys()
    return context
```

- The Decryptor secretes away the private context for its use in deciphering.
- The public part of the key is shared with third-party cloud providers in an open manner.

```
def get_public_context(private_context):
    public_context = private_context.copy()
    public_context.make_context_public()
    return public_context
```

5.3 Cloud Provider Module

Every simulated cloud provider safeguards its threat intelligence ratings (low, medium, high) by encrypting them with the common public context.

```
class CloudProvider:
```

```
    def encrypt_scores(self, low, medium, high):
        return ts.bfv_vector(self.public_context, [low, medium, high])
```

Example log:

[CloudA] Encrypting threat levels (JURISDICTION: EU)

5.4 Central GRC Server Module

Encrypted vectors are gathered by the CentralGRCServer and homomorphic calculation occurs without the decryption process.

```
class CentralGRCServer:
    def compute_totals(self):
        total_vector = self.encrypted_batches[0][2]
        for _, _, vec in self.encrypted_batches[1:]:
            total_vector += vec
        return total_vector
```

It supports compliance logic:

- Example: Rejecting data from a restricted jurisdiction (e.g., IN)
- Logging:

[COMPLIANCE ALERT] Data from CloudC rejected due to jurisdiction 'IN' restrictions!

5.5 Decryptor Module

Decryption is done by a party that holds the private portion of the context.

```
class Decryptor:
    def decrypt_result(self, encrypted_result):
        return encrypted_result.decrypt(secret_key=self.context.secret_key())
```

Sample output:

[Decryptor] Final decrypted threat score: [14, 11, 3]

5.6 Performance Profiling

For every function, the timed_operation() tool is used to capture and record information about CPU, wall time and memory usage.

[Timing] generate_context: CPU=1.6281s, Wall=1.6311s, Mem=139.56MB
[Timing] encrypt_scores: CPU=0.0112s, Wall=0.0111s, Mem=0.12MB
[Timing] compute_totals: CPU=0.0052s, Wall=0.0052s, Mem=0.50MB
[Timing] decrypt_result: CPU=0.0060s, Wall=0.0060s, Mem=0.00MB

The profiling made sure that small-vector use cases handle GRC efficiently with all data privacy protected.

6. Simulation and Results

A simulation based on Python and the TenSEAL homomorphic encryption library was used to find out if the Privacy-Preserving Multi-Cloud GRC Framework is possible. In the simulation, data that needs to be protected is passed and studied safely between cloud platforms.

6.1 Simulation Setup

- **Cloud Providers**: Three virtual providers called CloudA (EU), CloudB (US) and CloudC (IN) were created to operate under their own regulations.
- **Threat Intelligence Vectors**: The providers used the same public encryption key to encrypt their threat level vectors as Low, Medium or High.
- **GRC Server**: Without first decrypting the vectors, the GRC Server did the needed computation on them centrally.
- **Compliance Layer**: This layer filters out blocked regions (for example, CloudC from ‘IN’) from sending requests to the network.
- **Decryption**: The Decryptor module which has the private key, was used to decipher the MOD-Cipher aggregated ciphertext.

6.2 Execution Log

```
python3 run_simulation.py
=== Enhanced Regulatory-Aware HE-Based GRC Simulation ===
[Timing] generate_context: CPU=1.6281s, Wall=1.6311s, Mem=139.56MB
[Timing] get_public_context: CPU=1.6058s, Wall=1.6058s, Mem=81.55MB
[CloudA] Encrypting threat levels (JURISDICTION: EU)
[Timing] encrypt_scores: CPU=0.0112s, Wall=0.0111s, Mem=0.12MB
[CloudB] Encrypting threat levels (JURISDICTION: US)
[Timing] encrypt_scores: CPU=0.0140s, Wall=0.0139s, Mem=0.25MB
[CloudC] Encrypting threat levels (JURISDICTION: IN)
[Timing] encrypt_scores: CPU=0.0184s, Wall=0.0184s, Mem=0.50MB
[GRC] Received encrypted data from CloudA (EU)
[GRC] Received encrypted data from CloudB (US)
[GRC] Received encrypted data from CloudC (IN)
[COMPLIANCE ALERT] Data from CloudC rejected due to jurisdiction 'IN' restrictions!
[COMPLIANCE REPORT] Some data was rejected due to jurisdiction restrictions.
[GRC] Aggregating encrypted threat levels across authorized clouds...
[Timing] compute_totals: CPU=0.0052s, Wall=0.0052s, Mem=0.50MB
[Decryptor] Final decrypted threat score: [14, 11, 3]
[Timing] decrypt_result: CPU=0.0060s, Wall=0.0060s, Mem=0.00MB
[REPORT] Total Threats → Low: 14, Medium: 11, High: 3
```

6.3 Outcome and Insights

- The system was able to spot and reject non-compliant data and add approved information securely using HE.
- At each step, the raw data was kept confidential and no traces of final grade could be seen.
- Speed and memory use: Encryption, aggregation and decryption occurred instantly and used very little memory (no more than 1MB), showing the tool fits lightweight compliance scenarios.
- The framework is designed to restrict data according to each jurisdiction’s rules while the application is running.

7. Challenges and Mitigations

Still, although the simulation was successful, several practical issues were noticed in the multi-cloud GRC framework.

Managing The Keys

- It is difficult to handle encryption keys using multiple cloud service providers because of security risks.
- Only the decryptor holds the secret key, while cloud providers are just given a public element for encryption. This strategy helps avoid trusting someone outside the network and ensures decrypters are still in charge.

Computational Overhead

- Challenge: Performing homomorphic encryption takes more computational resources than standard calculation methods.
- The framework processes briefly decrypted information about threats and relies on profiling to improve program performance.

Following the rules set by the government

- Issue: Different regions have unique legal rules for cloud computing which complicates the use of unified analytics.
- The framework ensures the rejection of data contributions from non-compliant areas during the whole aggregation phase (for instance, CloudC was excluded from IN).

Reliable and True Source

- Problem: Ensuring that the data you get is genuine and that it has not been edited.
- Future updates may use digital signatures and check the source of data to build more trust.

The capabilities of HE for scaling up are not great.

- Problem: HE is not able to process large amounts of data or do difficult analytics efficiently.
- The existing implementation is limited to GRC projects that are not very big. In future, it might be useful to look at batching technologies and hybrid forms of cryptography.

8. Conclusion and Future Work

The project shows a real-world example of how a privacy-reliant GRC system can be used in multi-cloud settings with the help of homomorphic encryption. Thanks to the system, threat intelligence can be safely brought together without exposing original data so that privacy standards are always met.

In the simulation:

- All the cloud providers we were using were able to seamlessly gather encrypted data.
- Confirmation that the process of enforcing rules on taxes follows the guidelines of the jurisdictions.
- The results were as expected, showing that the system could be used for lighter types of computing.

Future Work

- Add SMPC technology to the architecture so that users and their data can be protected while deciding together.
- Test the framework using significant GRC datasets in order to see how it scales.
- By using Zero-Knowledge Proofs, some of the data's veracity can be proved without disclosing its whole content.
- When auditing, use automated encryption of data to ensure compliance issues are always caught and dealt with promptly.
- Testing in Real Situations: Deploy your prototype into hybrid cloud environments that use tough security and can send alerts for compliance.

9. Refrences

- Microsoft SEAL: Simple Encrypted Arithmetic Library. https://github.com/microsoft/SEAL
- TenSEAL: A Library for Homomorphic Encryption Operations on Tensors. OpenMined. https://github.com/OpenMined/TenSEAL
- European Union. General Data Protection Regulation (GDPR). https://gdpr.eu
- California Consumer Privacy Act (CCPA). https://oag.ca.gov/privacy/ccpa
- Craig Gentry. "Fully Homomorphic Encryption Using Ideal Lattices." STOC, 2009.
- Acar, A., Aksu, H., Uluagac, A. S., & Conti, M. (2018). A Survey on Homomorphic Encryption Schemes: Theory and Implementation. *ACM Computing Surveys (CSUR)*, 51(4), 1–35.
- Rivest, R. L., Adleman, L., & Dertouzos, M. L. (1978). On data banks and privacy homomorphisms. *Foundations of Secure Computation*, 169–180.
- Secure Multi-Party Computation (SMPC). Cryptography and Privacy Engineering Group. https://crypto.stanford.edu