# Report Cloud Computing IA2

**Sania Parekh 16010122132**

**Tanaya Pawar 16010122143**

**Sarthak Pokale 16010122146**

## Introduction

The **GDPR**, enforced in May 2018, sets stringent rules on personal data handling, requiring organizations to:

- Ensure transparency
- Maintain user consent
- Enable data erasure ("right to be forgotten")
- Report breaches within 72 hours
- Ensure data protection by design and by default

These regulations affect all entities handling EU residents' data, including cloud service providers (CSPs). However, **cloud environments are complex**, involving:

- Multiple parties (users, CSPs, third-party services)
- Geographic distribution (data centers in various jurisdictions)
- Dynamic provisioning (scaling, storage, migration)

This creates a **compliance gap** between **legal requirements** and **technical cloud infrastructure**.

---

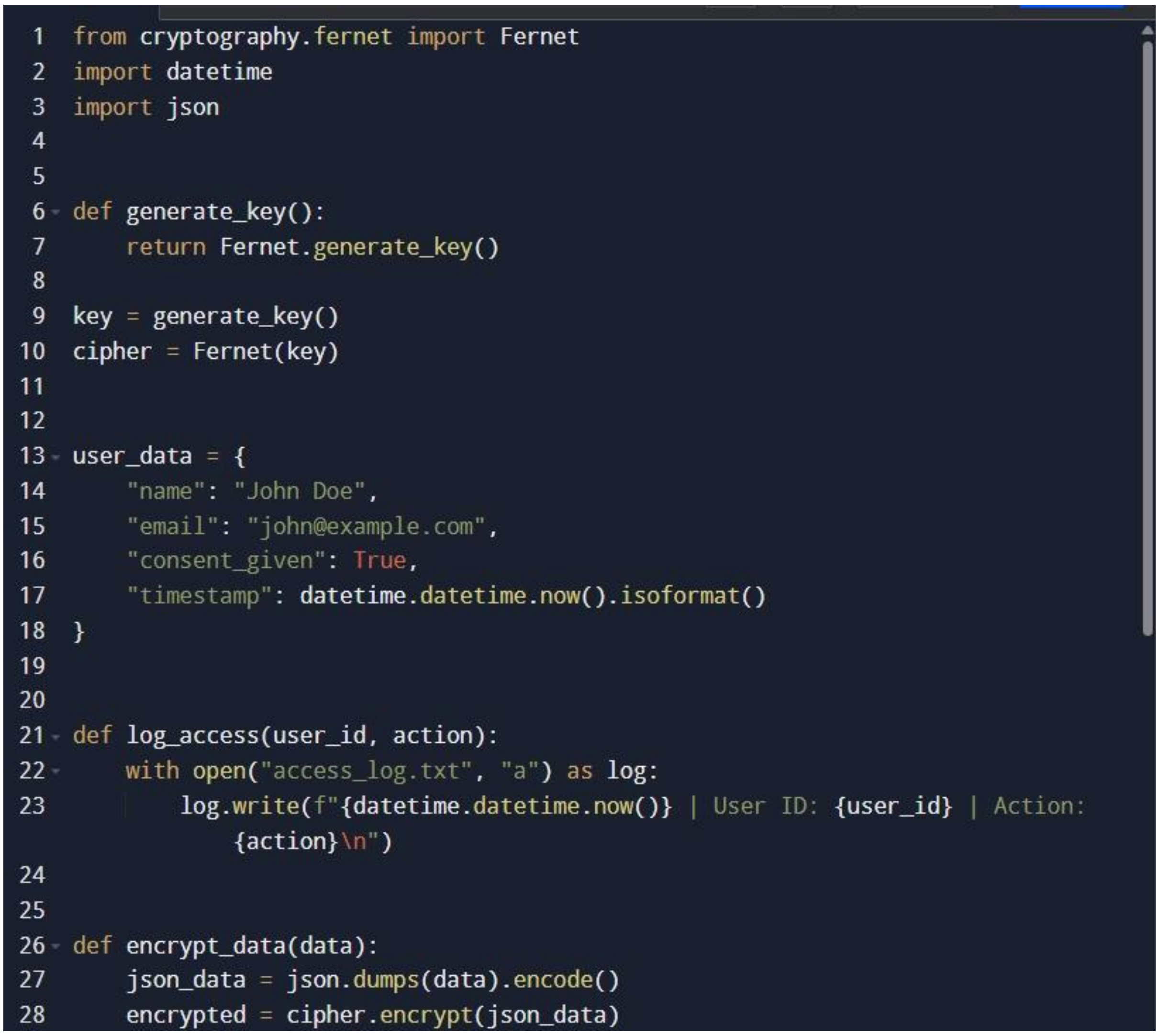
## Objectives of the Study

1. Analyze GDPR's impact on cloud data processing
2. Highlight key compliance challenges in cloud environments
3. Propose a **formal model for policy enforcement** of GDPR rules in clouds
4. Enable **auditable, privacy-preserving cloud data operations**

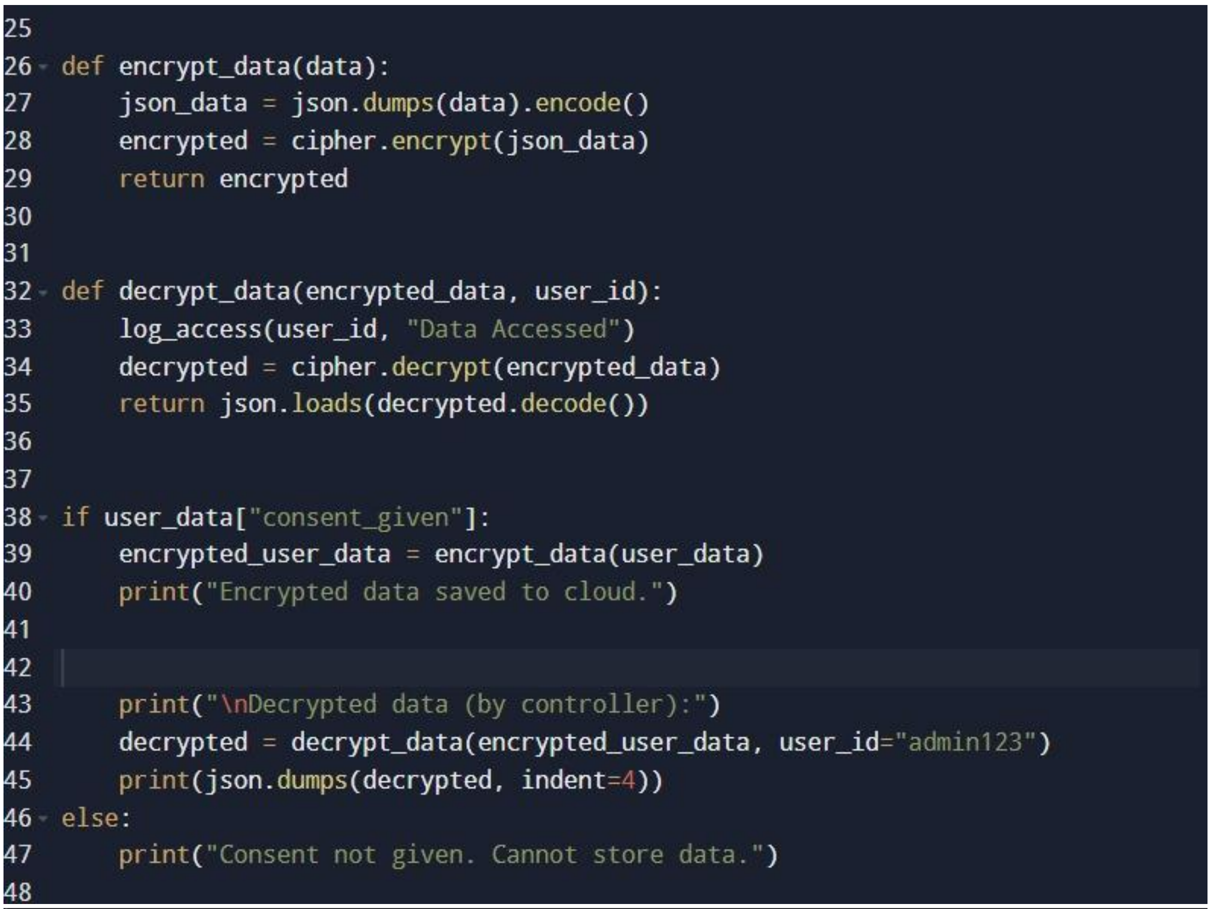
## Implementation

**A Python-based simulation was developed to demonstrate GDPR-compliant data handling in cloud environments, inspired by principles from the IEEE paper "Cloud Computing and the New EU General Data Protection Regulation". The program verifies user consent before processing (Article 6), encrypts personal data to enforce data protection by design (Article 25), and logs each data access for accountability (Article 30). It also supports access and erasure rights (Articles 15 & 17) by allowing secure retrieval of user data, while encryption ensures the security of processing (Article**

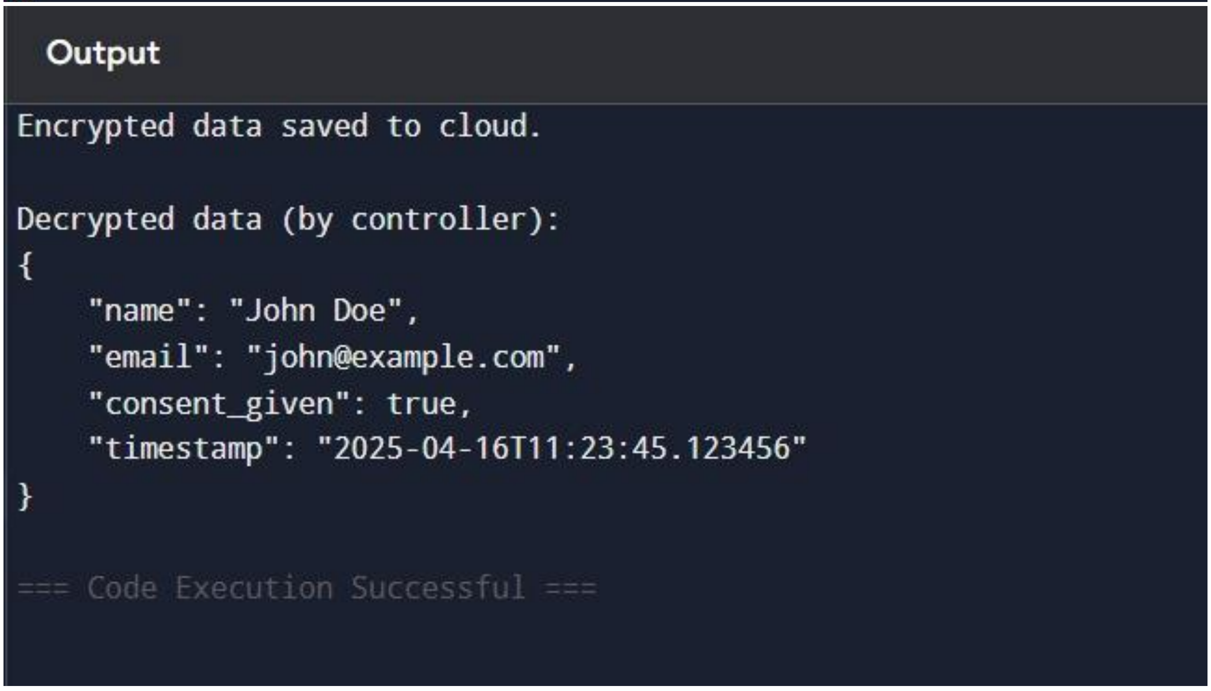
**32), even in case of data breaches.**



```python
from cryptography.fernet import Fernet
import datetime
import json


def generate_key():
    return Fernet.generate_key()

key = generate_key()
cipher = Fernet(key)


user_data = {
    "name": "John Doe",
    "email": "john@example.com",
    "consent_given": True,
    "timestamp": datetime.datetime.now().isoformat()
}


def log_access(user_id, action):
    with open("access_log.txt", "a") as log:
        log.write(f"{datetime.datetime.now()} | User ID: {user_id} | Action: {action}\n")


def encrypt_data(data):
    json_data = json.dumps(data).encode()
    encrypted = cipher.encrypt(json_data)
```



```python

def encrypt_data(data):
    json_data = json.dumps(data).encode()
    encrypted = cipher.encrypt(json_data)
    return encrypted


def decrypt_data(encrypted_data, user_id):
    log_access(user_id, "Data Accessed")
    decrypted = cipher.decrypt(encrypted_data)
    return json.loads(decrypted.decode())


if user_data["consent_given"]:
    encrypted_user_data = encrypt_data(user_data)
    print("Encrypted data saved to cloud.")


    print("\nDecrypted data (by controller):")
    decrypted = decrypt_data(encrypted_user_data, user_id="admin123")
    print(json.dumps(decrypted, indent=4))
else:
    print("Consent not given. Cannot store data.")

```



Output

```
Encrypted data saved to cloud.

Decrypted data (by controller):
{
    "name": "John Doe",
    "email": "john@example.com",
    "consent_given": true,
    "timestamp": "2025-04-16T11:23:45.123456"
}

=== Code Execution Successful ===
```

**Technical Challenges Identified**

1. **Lack of centralized control** over data movement and replication
2. **Data residency issues** – cross-border transfer limitations
3. **Insufficient logging** – actions on data are not transparently auditable
4. **Complex access control** – need for fine-grained, context-aware authorization
5. **Policy enforcement gaps** – compliance requires technical support, not just legal documentation

---

**Proposed Solution: Policy-Based Data Processing**

The authors propose a **declarative policy framework** where:

- Data processing rules are specified as **formal policies**
- Policies can include conditions like consent, purpose, time limits
- Access requests are evaluated **in real-time** against GDPR rules
- All actions are **logged and auditable**

**Key Components:**

- **Policy Language**: Based on usage control (UCON) models
- **Enforcement Point**: Controls access based on rules and logs decisions
- **Obligation Engine**: Ensures follow-up actions like notifying users or deleting data

---

**Example Use Case**

Imagine a **cloud storage service**:

- A user uploads personal data with consent for processing only for healthcare research.
- A third-party analytics service requests access.
- The **policy engine checks**:
  - If the user has given explicit consent
  - If the processing purpose matches
  - If retention time is still valid
- If all checks pass, access is granted and logged. Otherwise, it is denied.

This system ensures **accountability, transparency, and privacy enforcement** automatically.

---

## Evaluation

The authors implemented a **prototype policy engine** and tested it in simulated cloud scenarios. Results show:

- **Low performance overhead** (~10ms per decision)
- **High expressiveness** – can capture most GDPR rules
- **Auditability** – full access logs and traceability
- Scales well in distributed environments

---

## Conclusion

This paper makes a valuable contribution to bridging the **legal-technical gap** in GDPR compliance. By proposing a **formal policy-based access control system**, the authors offer a scalable and auditable way for cloud providers to respect user privacy rights.

This research emphasizes that **GDPR compliance is not only legal** — it must be **enforced by design** in the technical architecture of cloud systems.