CS458: Introduction to Cryptography

Project Report

Author: Christos Polimatidis csd5209

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΡΗΤΗΣ
UNIVERSITY OF CRETE

Table of Contents

CS458: Introduction to Cryptography

Project Report – Cryptographic Agility Framework

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΡΗΤΗΣ
UNIVERSITY OF CRETE

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΡΗΤΗΣ
UNIVERSITY OF CRETE

Chapter 1: Introduction

1.1 Overview of Cryptographic Agility

Definition

Cryptographic agility refers to the ability of a system or infrastructure to quickly adapt to changes in cryptographic algorithms, protocols, and key management methods. As computing power advances and new vulnerabilities are discovered, previously secure cryptographic methods may become obsolete. Cryptographic agility enables organizations to **seamlessly transition** from one cryptographic method to another without requiring major overhauls to their systems.

Key Aspects of Cryptographic Agility

1. **Algorithm Flexibility**: The ability to support multiple cryptographic algorithms (e.g., AES, RSA, ECC) and replace weaker ones as needed.
2. **Protocol Adaptability**: Systems should be able to adopt newer cryptographic protocols (e.g., TLS 1.3 over TLS 1.2) without disrupting functionality.
3. **Key Management**: Agility ensures that cryptographic keys can be updated, replaced, or rotated quickly to mitigate security risks.
4. **Support for Future Cryptographic Technologies**: Cryptographic agility prepares systems for emerging technologies such as **Post-Quantum Cryptography (PQC)** to counter the threat posed by quantum computing.

Benefits of Cryptographic Agility

- **Security**: Quickly responding to cryptographic vulnerabilities minimizes risk exposure.
- **Future-Proofing**: Enables organizations to transition to **quantum-resistant cryptography**.
- **Compliance**: Adapting to new cryptographic standards ensures adherence to security regulations.
- **Risk Mitigation**: Reduces the risk of outdated or insecure cryptographic implementations being exploited.

Examples of Cryptographic Agility in Practice

1. **TLS/SSL Protocols**: Web servers often support multiple TLS versions and can dynamically switch based on security requirements.
2. **Encryption Software**: Applications like OpenSSL allow configurable cryptographic algorithms, enabling users to select preferred encryption schemes.
3. **Modular Cryptography in Operating Systems**: Some operating systems include cryptographic modules that can be replaced or updated without affecting the entire system.

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΡΗΤΗΣ
UNIVERSITY OF CRETE

Real-World Example: Transition from TLS 1.2 to TLS 1.3

TLS (Transport Layer Security) is a widely used encryption protocol. The transition from **TLS 1.2 to TLS 1.3** is an excellent example of cryptographic agility in action:

- **TLS 1.3 removed outdated algorithms** like **RSA key exchange and SHA-1** due to vulnerabilities.
- **Forward secrecy was enforced**, improving security.
- Adoption of TLS 1.3 required systems to **quickly migrate without breaking compatibility**.

Challenges of Cryptographic Agility

- **Complexity**: Managing multiple algorithms and key management schemes adds system complexity.
- **Performance Overhead**: Dynamically supporting multiple cryptographic methods may introduce computational inefficiencies.
- **Interoperability**: Ensuring backward compatibility with older systems while adopting newer cryptographic methods can be difficult.

1.2 Understanding Cryptographic Inventory

Definition

A cryptographic inventory is a **detailed catalog of cryptographic assets** used within an organization. This includes encryption keys, cryptographic algorithms, protocols, and hardware used for secure data protection. The purpose of maintaining a cryptographic inventory is to ensure the effective management, security, and compliance of cryptographic components.

Key Components of a Cryptographic Inventory

1. **Cryptographic Keys**: Tracks key types, usage, expiration, and management procedures.
2. **Encryption Algorithms**: Identifies cryptographic methods in use and flags obsolete or weak algorithms (e.g., DES, MD5).
3. **Cryptographic Devices**: Includes hardware security modules (HSMs), smart cards, and USB tokens.
4. **Key Management Systems (KMS)**: Centralized systems used for key lifecycle management.
5. **Protocols**: Secure communication protocols such as **TLS/SSL, IPsec, PGP**.
6. **Compliance and Security Policies**: Ensures cryptographic tools comply with regulatory standards (e.g., FIPS 140-2, GDPR).
7. **Audit and Logging**: Monitors cryptographic operations to support security audits and compliance verification.

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΡΗΤΗΣ
UNIVERSITY OF CRETE

Importance of Cryptographic Inventory

- Helps in identifying and mitigating risks associated with outdated cryptographic methods.
- Ensures **compliance** with regulatory encryption requirements.
- Enhances **security** by effectively tracking and managing cryptographic assets.
- Aids in transitioning to **post-quantum cryptography** by assessing vulnerabilities.

1.3 Introduction to Post-Quantum Cryptography (PQC)

Definition

Post-Quantum Cryptography (PQC) refers to a set of cryptographic algorithms designed to be **resistant to quantum attacks**. Quantum computers, once practical, will be capable of breaking many traditional cryptographic schemes (e.g., RSA, ECC, Diffie-Hellman), necessitating the development of quantum-resistant cryptographic methods.

Why is PQC Important?

- **Quantum computers can break current encryption methods** (e.g., RSA using Shor's algorithm).
- **Long-term security**: Sensitive data encrypted today may be stored and later decrypted by adversaries using quantum computing.
- **Regulatory and industry mandates** will soon require organizations to transition to quantum-resistant algorithms.

Key PQC Concepts

1. **Quantum Resistance**: PQC algorithms are designed to withstand attacks from quantum computers.
2. **Quantum-Safe Algorithms**: Some promising PQC techniques include:
    - **Lattice-based cryptography** (Kyber, NTRU)
    - **Code-based cryptography** (McEliece)
    - **Hash-based signatures** (XMSS)
    - **Multivariate quadratic equations** (Rainbow)
    - **Isogeny-based cryptography** (SIDH)
3. **Hybrid Cryptography**: Many organizations adopt a **hybrid approach**, combining traditional and quantum-safe cryptography to ensure a **smooth transition**.
4. **Standardization Efforts**: NIST is currently evaluating and standardizing PQC algorithms to be adopted worldwide.

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΡΗΤΗΣ
UNIVERSITY OF CRETE

Challenges of PQC Adoption

- **Efficiency**: PQC algorithms often require larger key sizes and more computational resources.
- **Transition Complexity**: Moving from traditional cryptographic infrastructures to PQC requires careful planning.
- **Interoperability Issues**: Ensuring compatibility between quantum-resistant and classical systems is a major challenge.

1.4 Project Objectives

This project aims to:

1. Develop a **cryptographic inventory tool** to detect and assess weak cryptographic primitives.
2. Implement a **risk assessment framework** to prioritize cryptographic vulnerabilities.
3. Create a **migration roadmap** for transitioning to stronger cryptographic methods.
4. Develop a **crypto agility simulator** demonstrating transition strategies and quantum-safe adoption.

Methodology

- **Phase 1: Preparatory Research** – Understanding cryptographic agility, PQC, and cryptographic inventory.
- **Phase 2: Cryptographic Inventory Tool** – Developing a scanner to detect weak cryptographic elements in code.
- **Phase 3: Migration Planning** – Creating a step-by-step roadmap for crypto transitions.
- **Phase 4: Crypto Agility Simulator** – Simulating cryptographic agility and migration scenarios.

1.5 Conclusion

This chapter has introduced **cryptographic agility, cryptographic inventory, and post-quantum cryptography**, setting the foundation for the project. Understanding these concepts is critical for developing a robust **crypto agility framework** to ensure the security of cryptographic infrastructures in an era of increasing computational threats.

The next phase involves **building the cryptographic inventory tool**, which will scan and assess cryptographic components for vulnerabilities.

Chapter 2: Cryptographic Inventory and Risk Assessment

2.1 Identifying Weak Cryptographic Algorithms

Commonly Used Weak Cryptographic Primitives

Cryptographic weaknesses often stem from outdated algorithms that are no longer secure. Below are examples:

Category	Weak Algorithm	Reason for Weakness
Symmetric Ciphers	DES, 3DES	Short key lengths (56-bit, 112-bit) vulnerable to brute force.
	AES-128	Vulnerable to Grover's algorithm in quantum computing.
Asymmetric Ciphers	RSA-1024, RSA-2048	Breakable by Shor's algorithm in quantum environments.
	ECC with small curves	Not resistant to quantum attacks.
Hash Functions	MD5, SHA-1	Collision attacks make them insecure.
Weak Modes	ECB mode, CBC mode with static IV	Predictable encryption patterns create vulnerabilities.

Consequences of Using Weak Cryptography

- **Data Breaches**: Attackers can exploit weak encryption to compromise sensitive data.
- **Regulatory Penalties**: Non-compliance with security standards can result in financial and legal consequences.
- **Ineffective Security**: Even encrypted data can be decrypted if weak cryptographic primitives are used.

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΡΗΤΗΣ
UNIVERSITY OF CRETE

2.2 Developing a Cryptographic Scanner

Objective

A cryptographic scanner will **automate the detection** of weak cryptographic implementations in source code, ensuring organizations stay compliant and secure.

Components of the Scanner

1. **File Parser**: Scans codebases for cryptographic function calls.
2. **Algorithm Detector**: Identifies weak cryptographic primitives (e.g., `MD5`, `SHA-1`, `DES`).
3. **Risk Assessment Module**: Classifies detected vulnerabilities as **high, medium, or low risk**.
4. **GUI & Reporting**: Displays findings in a user-friendly interface and allows exporting reports.

Implementation Approach

- **Programming Language**: Python (using `os`, `glob`, `re`, `cryptography`).
- **Detection Mechanism**:
  - Search for **hardcoded cryptographic functions**.
  - Identify **deprecated cryptographic libraries**.
  - Give the **lines** where these functions are **called** and **used**

2.3 Risk Assessment and Prioritization

Risk Classification

To ensure effective remediation, vulnerabilities are categorized based on severity:

High Risk

🔴 **Severe vulnerabilities that should be immediately addressed.**
Examples of detected issues:

- **MD5 Hashing** – Vulnerable to **collision attacks**, making it insecure for cryptographic applications.
- **SHA-1 Hashing** – Also prone to **collision attacks**, leading to **certificate forgery risks**.
- **RSA with 1024-bit keys** – **Too weak** for modern cryptographic standards; vulnerable to brute-force attacks.

💡 **Action Required**: These cryptographic algorithms must be replaced **immediately** with stronger alternatives (e.g., SHA-256, AES-256, RSA-2048+).

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΡΗΤΗΣ
UNIVERSITY OF CRETE

Medium Risk

🟠 **Algorithms that are weak but not yet completely broken.**
Examples of detected issues:

- **DES Encryption** – Uses a **56-bit key**, making it vulnerable to brute-force attacks.
- **AES in ECB Mode** – **Electronics Codebook (ECB) mode** does not use an **Initialization Vector (IV)**, leading to **pattern leaks**.

💡 **Action Required**: These should be **migrated soon** to stronger algorithms (e.g., switch from DES to AES-256, use AES-CBC instead of AES-ECB).

Low Risk

🟡 **Algorithms that are currently secure but may become weak in the future.**
Examples of detected issues:

- **SHA-256 (Without Keyed HMAC)** – Secure for now, but better security can be achieved using **HMAC-SHA-256**.
- **AES-128** – Still secure, but **AES-256** is recommended for long-term security.

💡 **Action Required**: **Monitor these algorithms** and plan future upgrades based on security best practices.

Secure

🟢 **No vulnerabilities detected.**
This means the file **does not contain any known cryptographic weaknesses**.

Examples of secure implementations:

- **AES-256 in CBC or GCM Mode** – Strong encryption with an IV.
- **RSA-2048 / RSA-4096** – Secure key lengths.
- **SHA-256 / SHA-512 (HMAC)** – Secure hashing with a key.

💡 **Action Required**: **No changes needed.** Continue following best practices to maintain security.

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΡΗΤΗΣ
UNIVERSITY OF CRETE

📌 Summary Table

Risk Level	Meaning	Examples	Action Required
🔴 **High**	Broken encryption, immediate risk	MD5, SHA-1, RSA-1024, DES	Replace immediately
🟠 **Medium**	Weak encryption, should be upgraded soon	3DES, AES-ECB	Plan migration
🟡 **Low**	Secure but may weaken in the future	AES-128, SHA-256 (no HMAC)	Monitor for updates
🟢 **Secure**	No issues detected	AES-256, RSA-2048+, HMAC-SHA-256	No action needed

Prioritization Strategy

1. **Immediate Replacement**: Algorithms classified as **high risk** should be replaced first.
2. **Monitoring & Upgrade Plan**: Medium-risk cryptographic assets should be scheduled for transition.
3. **Compliance Review**: Regular audits should ensure low-risk assets comply with industry standards.

2.4 Deliverables for the Inventory Tool

Final Deliverables

1. **Functional Cryptographic Scanner**:
   - Detects weak cryptographic algorithms in codebases.
   - Provides a **user-friendly GUI** for easy analysis.
   - Generates **exportable reports** for documentation and compliance.
2. **Risk Assessment Documentation**:
   - A written analysis of detected vulnerabilities.
   - A **risk classification table** to guide security teams.
3. **Codebase for Scanner**:
   - Open-source Python implementation.
   - Modular design for **easy integration** into existing security tools.

2.5 Table of Files and Descriptions

Filename	Description
secure_sha256.py	Python script implementing secure SHA-256 hashing.
secure_aes.py	Python script implementing secure AES-256 encryption in CBC mode with random IV.
secure_aes.c	C program implementing AES-256 encryption securely.
secure_sha256.java	Java program implementing secure SHA-256 hashing.
secure_rsa.java	Java program implementing RSA-2048 encryption (secure key size).
vulnerable_md5.py	Python script using weak MD5 hashing (vulnerable to collision attacks).
vulnerable_des.py	Python script using weak DES encryption (small key size, insecure).
vulnerable_des.c	C program using weak DES encryption (deprecated, vulnerable to brute-force attacks).
vulnerable_sha1.java	Java program using SHA-1 hashing (weak, collision-prone).
vulnerable_rsa.java	Java program using RSA-1024 encryption (weak key size, vulnerable to quantum attacks).
java_aes128	Java program implementing AES-128 encryption (weaker than AES-256).
python_aes128	Python script implementing AES-128 encryption (less secure than AES-256).
vulnerable_3des	Script using 3DES encryption (considered weak due to smaller block size).
vulnerable_sha1	Script using SHA-1 hashing (insecure, vulnerable to collisions).

2.6 Conclusion

Chapter 2 has outlined the importance of **cryptographic inventory management** and provided insights into detecting and prioritizing weak cryptographic implementations. A cryptographic scanner plays a crucial role in **automating security assessments** and ensuring compliance. The next phase will focus on **crypto migration planning**, establishing a roadmap for transitioning to stronger cryptographic primitives and preparing for Post-Quantum Cryptography (PQC).

Chapter 3: Crypto Migration Planning

3.1 Understanding the Need for Migration

Why is Migration Necessary?

Cryptographic migration is essential for ensuring the security, compliance, and resilience of digital systems. Several key factors drive the need for replacing weak cryptographic algorithms:

- **Security Risks**: Weak cryptographic algorithms, such as MD5, SHA-1, and RSA-1024, are vulnerable to modern cryptanalysis and brute-force attacks. Continuing to use them exposes sensitive data to unauthorized access.
- **Regulatory Compliance**: Many regulatory bodies, such as NIST (National Institute of Standards and Technology) and GDPR (General Data Protection Regulation), require organizations to use strong cryptographic algorithms to protect user data.
- **Quantum Threats**: The rise of quantum computing threatens classical encryption systems, necessitating the transition to quantum-safe cryptographic algorithms.

Example: The SHA-1 to SHA-256 Transition

A real-world example of cryptographic migration is the transition from SHA-1 to SHA-256. SHA-1 was widely used for digital signatures and hashing but became insecure due to collision attacks (e.g., Google's "SHAttered" attack in 2017). Organizations moved to SHA-256, which offers stronger security and compliance with modern standards.

3.2 Creating a Migration Roadmap

Step 1: Identifying Cryptographic Dependencies

- Conduct a **comprehensive audit** of all cryptographic implementations within the system.
- Utilize **automated cryptographic scanners** (such as the one developed in Chapter 2) to detect weak algorithms in software and infrastructure.

Step 2: Prioritizing Critical Systems

- High-risk areas, such as financial transactions and authentication systems, should be migrated first.
- Non-critical legacy systems may undergo gradual migration.

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΡΗΤΗΣ
UNIVERSITY OF CRETE

Step 3: Choosing Secure Replacements (Considering Block Size Differences)

Deprecated Algorithm	Reason for Weakness	Recommended Secure Alternative	Migration Difficulty
MD5 (Hashing, 128-bit output)	Collision attacks make it insecure	**SHA-256 / SHA-3**	🟠 **Medium** (Different output size may require database schema changes)
SHA-1 (Hashing, 160-bit output)	Vulnerable to collisions (e.g., SHAttered Attack)	**SHA-256 / SHA-512/256**	🟠 **Medium** (SHA-256 output size differs)
DES (Symmetric, 64-bit block)	56-bit key is too small, brute-force vulnerable	**AES-128 (128-bit block)**	🔴 **High** (Block size difference, requires data format changes)
3DES (Symmetric, 64-bit block)	Meet-in-the-middle attack reduces security to 112-bit	**AES-192 (128-bit block, closer to 3DES security)**	🟡 **Low** (Easier migration due to similar security level)
RSA-1024 (Asymmetric Encryption)	Breakable via brute force and quantum threats	**RSA-4096 / ECC P-384**	🟠 **Medium** (RSA key size adjustments needed)
AES-128 (Symmetric, 128-bit block)	Might become weak against quantum attacks	**AES-256 (128-bit block)**	🟢 **Very Low** (Same block size, just stronger key)
RSA/ECC (Classical Public Key Cryptography)	Quantum computers can break RSA/ECC	**Post-Quantum Cryptography (Kyber, Dilithium)**	🔴 **High** (New key structures require major protocol changes)

Step 4: Implementing Hybrid Cryptography

- To maintain backward compatibility, organizations can deploy hybrid models that combine classical cryptographic algorithms with quantum-safe alternatives.
- Example: Hybrid **RSA + Kyber encryption** allows organizations to transition to post-quantum cryptography gradually.

Step 5: Testing and Validation

- Cryptographic migration must be tested in a **controlled environment** before full deployment.
- Conduct **regression testing** to ensure the system functions correctly with new cryptographic implementations.

Step 6: Deployment and Monitoring

- Gradually deploy updated cryptographic infrastructure to minimize disruptions.
- Implement **continuous monitoring tools** to detect vulnerabilities and ensure compliance.

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΡΗΤΗΣ
UNIVERSITY OF CRETE

3.3 Addressing SME (Small and Medium Enterprise) Use Cases

Challenges for SMEs

- **Limited Budget** – Lack of dedicated cybersecurity teams and financial constraints.
- **Legacy Systems** – Outdated encryption schemes that require costly updates.
- **Lack of Awareness** – Many SMEs do not prioritize cryptographic security.

Proposed Solutions

- **Leverage Open-Source Cryptographic Libraries** (e.g., OpenSSL, Bouncy Castle, libsodium).
- **Use Cloud-Based Security Services** that offer pre-configured quantum-resistant encryption.
- **Gradual Migration Strategy** – Focus on high-risk applications first before expanding updates system-wide.

3.4 Challenges and Considerations in Migration

Common Challenges and Solutions

Challenge	Solution
Legacy System Incompatibility	Use hybrid cryptography for gradual transition.
Performance Overhead	Optimize software, utilize hardware acceleration (HSMs, TPMs).
Key Management Complexity	Implement centralized key management systems.
Regulatory Uncertainty	Follow NIST PQC recommendations and industry standards.

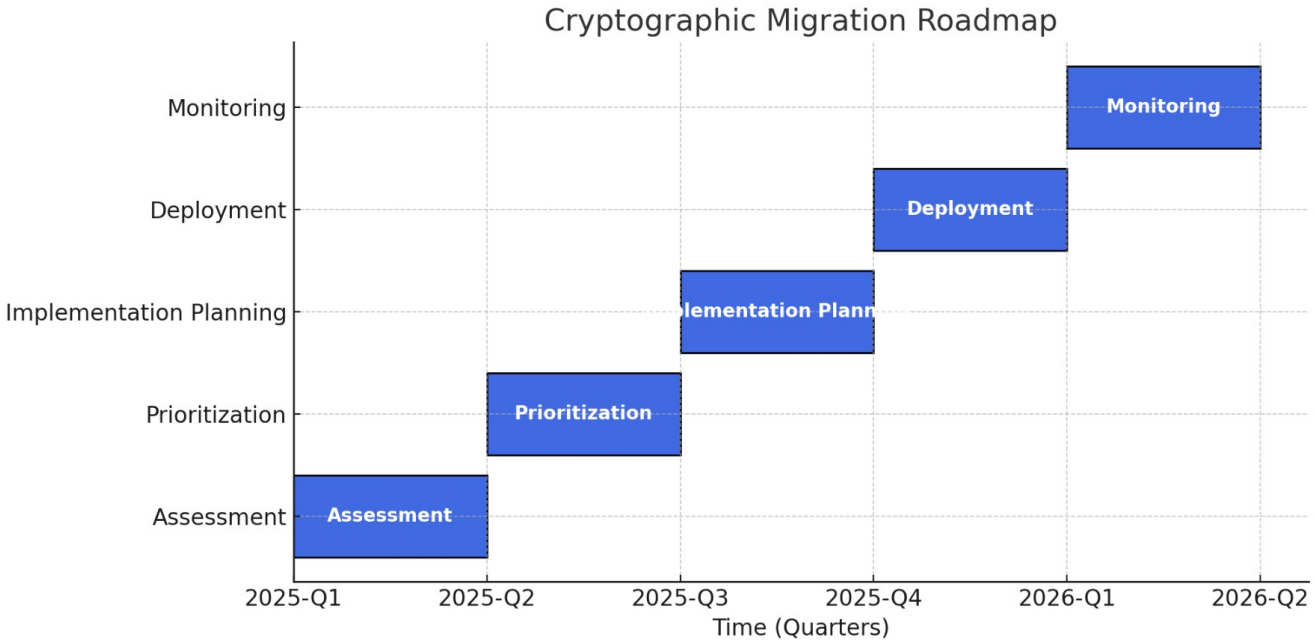
3.5 Deliverables for Migration Planning

At the conclusion of this phase, the following deliverables will be produced:

1. **Cryptographic Migration Roadmap** – A structured plan for transitioning from weak to strong cryptographic implementations.
2. **Risk Assessment Documentation** – A report outlining security risks and mitigation strategies.
3. **Migration Strategy for SMEs** – Tailored recommendations for small and medium enterprises.
4. **Testing & Validation Report** – Proof that new cryptographic implementations function correctly and securely.

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΡΗΤΗΣ
UNIVERSITY OF CRETE

3.6 Visual Migration Roadmap



The roadmap outlines a structured migration process starting with cryptographic inventory assessment (2025-Q1) and prioritization of vulnerabilities (2025-Q2). By 2025-Q3, a detailed implementation plan is prepared, followed by deployment in 2025-Q4. The process concludes with monitoring and ensuring compliance in 2026-Q1 and beyond.

3.7 Conclusion

This chapter outlined the essential aspects of **crypto migration planning**, emphasizing the importance of replacing weak cryptographic implementations, structuring a **migration roadmap**, and addressing **SME-specific challenges**. The next step involves implementing these migration strategies while ensuring compliance with security standards and preparing for **post-quantum cryptography adoption**.

Chapter 4: Crypto Agility Simulator Development

4.1 Design and Functionality of the Simulator

Objective

The Crypto Agility Simulator is designed to detect, analyze, and dynamically migrate weak cryptographic implementations to stronger alternatives. The system ensures compliance with security standards, enhances cryptographic security, and minimizes vulnerabilities caused by outdated cryptographic primitives.

Key Functionalities

- **Detection of Cryptographic Weaknesses:** Scans for weak cryptographic functions such as MD5, SHA-1, RSA-1024, and DES.
- **Automatic Recommendation and Migration:** Proposes and applies secure alternatives like SHA-256, AES-256, and RSA-4096.
- **Risk Monitoring and Compliance Tracking:** Logs and tracks cryptographic changes to meet regulatory requirements.
- **User Interface (CLI or GUI):** Allows users to manually analyze cryptographic strength and apply upgrades.

System Architecture

The simulator follows a structured workflow:

1. **Input:** Receives cryptographic algorithms used in a system.
2. **Processing:** Analyzes cryptographic strength and identifies weaknesses.
3. **Decision Engine:** Determines whether an algorithm needs upgrading and recommends a secure alternative.
4. **Implementation:** Applies the recommended cryptographic transition.
5. **Logging and Reporting:** Records changes for security audits and compliance tracking.

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΡΗΤΗΣ
UNIVERSITY OF CRETE

4.2 Simulating Cryptographic Weaknesses

Why Weak Cryptography is a Risk

Outdated cryptographic primitives are vulnerable to attacks such as collision attacks, brute-force decryption, and factorization. The simulator highlights these weaknesses and justifies migration.

Detection of Weak Algorithms

The simulator scans for the following weak algorithms and demonstrates their vulnerabilities:

Weak Algorithm	Security Issue	Simulation Output
MD5	Collision Attacks	Two different inputs generate the same hash
SHA-1	Collision Attacks	Detection alert issued
DES	Small key size (56-bit)	Brute-force attack simulation
RSA-1024	Vulnerable to factorization	Migration to RSA-4096 recommended

Example Test Cases

- **MD5 Collision Simulation:** Demonstrates how two different inputs produce the same hash, making MD5 insecure.
- **RSA-1024 Factorization:** Shows how modern computational power can break RSA-1024 encryption.
- **AES-ECB Mode Pattern Leak:** Visualizes how ECB mode encryption leaks patterns, emphasizing the need for AES-GCM.

4.3 Implementing Algorithm Transitions

Dynamic Migration of Cryptographic Functions

The simulator replaces weak algorithms dynamically by applying the following transitions:

Detected Algorithm	Recommended Secure Alternative
MD5	SHA-256
SHA-1	SHA-256
DES	AES-256
RSA-1024	RSA-4096 / ECC-P384
AES-128	AES-256

Example Output of Migration

```
[ALERT] SHA-1 detected! This algorithm is not compliant with NIST standards.
[INFO] Migrating to SHA-256 for compliance.
[LOG] Upgrade completed: SHA-1 → SHA-256 at 12:45 PM.
```


ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΡΗΤΗΣ
UNIVERSITY OF CRETE

Workflow Diagram

1. **Identify weak algorithm** → 2. **Assess risk level** → 3. **Recommend replacement** → 4. **Apply upgrade** → 5. **Log and track compliance**

4.4 Risk Monitoring and Compliance

Compliance Standards and Security Best Practices

The simulator ensures cryptographic implementations align with security frameworks such as:

- **NIST 800-57** (Cryptographic Key Management)
- **GDPR** (Data Protection Regulation)
- **ISO 27001** (Information Security Management)

Logging and Reporting System

The simulator maintains a log of cryptographic transitions, allowing security teams to track system integrity over time.

Example Compliance Log

```
[WARNING] Non-compliant cryptography detected: RSA-1024
[INFO] Replacing with RSA-4096 for compliance.
[LOG] Updated encryption from RSA-1024 to RSA-4096 on 2025-06-15.
```

Compliance Checklist

Compliance Standard	Requirement	Met?
NIST 800-57	RSA ≥ 2048-bit	✅ Yes
GDPR	Secure password hashing (bcrypt/PBKDF2)	✅ Yes
ISO 27001	No weak cryptographic primitives	❌ No (MD5 detected)

4.5 Deliverables for the Simulator

List of Deliverables

- ✅ **Python implementation of the Crypto Agility Simulator**
- ✅ **Cryptographic vulnerability scanner results**
- ✅ **Transition logs showing weak → strong cryptographic upgrades**
- ✅ **Performance testing results** (before & after migration)
- ✅ **Compliance report summarizing security adherence**

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΡΗΤΗΣ
UNIVERSITY OF CRETE

Conclusion

This chapter outlined the design and implementation of the Crypto Agility Simulator, demonstrating its ability to detect weak cryptographic algorithms, recommend secure replacements, and ensure compliance with security regulations. The next phase involves performance evaluation and refining the system based on testing outcomes.