Crypto Agility Framework for Weak Cryptography and PQC transition

TEAM30 - Efthimis Papageorgiou CSDP1344 – Nikos Giovanopoulos CSD4613

Part 1: Preparatory Phase

EUROPE COMMISSION RECOMMENDATION

Key Concepts and Terminology

Post-Quantum Cryptography (PQC) represents a critical evolution in securing digital infrastructures against the emerging threat of quantum computing. With the advent of quantum computers, current cryptographic standards, particularly asymmetric cryptography, face the risk of being rendered obsolete due to their vulnerabilities to quantum attacks. The European Commission's "Coordinated Implementation Roadmap for the Transition to Post-Quantum Cryptography" establishes a foundational framework for transitioning to PQC.

Context and Importance

The European Commission emphasizes the strategic importance of encryption in safeguarding the digital ecosystem. This includes ensuring the confidentiality and integrity of sensitive communications, securing public administration systems, and maintaining critical infrastructure resilience. Quantum computing introduces a dual-edged potential—while unlocking groundbreaking opportunities, it simultaneously threatens traditional cryptographic mechanisms.

Objectives Outlined in the EU Recommendation

The document delineates clear objectives to guide member states through a coordinated and harmonized transition:

1. **Defining a Coordinated Roadmap**: Establishing a joint implementation strategy across the European Union.
2. **Standardizing Algorithms**: Selecting and adopting PQC algorithms as Union-wide standards.
3. **Supporting National Transition Plans**: Encouraging member states to align national plans with the EU's coordinated roadmap.
4. **Stakeholder Engagement**: Involving a broad spectrum of actors, including cybersecurity experts, industry participants, and government agencies.

Vulnerabilities in Cryptographic Systems

Quantum computing poses specific risks to:

- **Asymmetric cryptography**: Algorithms like RSA, ECC, and DH are highly susceptible to quantum attacks.

- **Public Key Infrastructure (PKI)**: The foundational mechanisms ensuring trust in digital certificates face significant challenges.
- **Long-Term Confidentiality**: Sensitive data stored today could be decrypted retrospectively with future quantum advancements.

Implementation Strategy

The document proposes a structured approach:

- **Hybrid Cryptographic Models**: Gradual integration of PQC with existing cryptographic systems to ensure compatibility during the transition phase.
- **Interoperability Focus**: Collaboration at the EU level to establish standards ensuring seamless communication across borders.
- **Monitoring and Review**: Continuous assessment of progress and effectiveness, with provisions for additional legislative actions if required.

Compliance and Governance

The roadmap aligns with broader EU cybersecurity strategies and respects fundamental rights under the EU Charter of Fundamental Rights and the European Convention on Human Rights. It mandates the inclusion of representatives from relevant national security and cybersecurity agencies, fostering an inclusive governance model.

Modular Integration

This section forms the foundational layer of understanding PQC transition. Subsequent parts will expand on open-source tools, cryptographic inventories, and agility mechanisms while tying back to the principles and strategies outlined here.

QUANTUM-READINESS: MIGRATION TO POST-QUANTUM CRYPTOGRAPHY

Building Quantum-Readiness for Migration to PQC

The “Quantum-Readiness: Migration to Post-Quantum Cryptography” framework developed by the CISA, NSA, and NIST provides practical guidance for organizations—especially those managing critical infrastructure—on preparing for the shift to Post-Quantum Cryptography (PQC). It emphasizes proactive measures to mitigate the risk posed by cryptanalytically relevant quantum computers (CRQC).

Key Concepts and Urgency of Preparation

The Threat of CRQC

Cryptanalytically relevant quantum computers have the potential to break widely used public-key cryptographic systems such as RSA, ECC, and ECDSA. Organizations need to anticipate this future threat, especially for systems safeguarding sensitive data with long-term confidentiality requirements.

Why Prepare Now?

Early preparation minimizes risks associated with the “harvest now, decrypt later” strategy by adversaries who collect encrypted data now for future decryption. Organizations must transition to PQC before such quantum threats become operational realities.

Establishing a Quantum-Readiness Roadmap

Organizations are advised to build a roadmap by:

1. **Forming a Project Management Team**: Tasked with identifying and prioritizing quantum-vulnerable systems and overseeing migration efforts.
2. **Conducting Cryptographic Discovery**: Mapping current dependencies on quantum-vulnerable algorithms like RSA and ECC.
3. **Collaborating with Vendors**: Understanding vendors' quantum-readiness roadmaps and influencing their timelines for PQC integration.

Preparing a Cryptographic Inventory

A comprehensive cryptographic inventory is central to quantum-readiness:

- **Purpose**:
  - Identifies systems at risk due to quantum-vulnerable cryptography.
  - Guides prioritization in transitioning to PQC.
  - Supports a transition to zero-trust architectures.
- **Steps**:
  1. **Discovery Tools**: Utilize tools to identify cryptographic algorithms in:
     - Network protocols.
     - End-user applications and firmware.
     - Continuous delivery pipelines.
  2. **Vendor Engagement**: Request detailed lists of embedded cryptographic technologies from vendors.
  3. **Integration with Risk Assessments**: Feed inventory data into broader organizational risk assessments to prioritize critical transitions.
- **Special Focus**:
  - Correlate the cryptographic inventory with asset inventories and access management systems.
  - Identify dependencies involving high-risk or long-term confidentiality data.

Engagement with Technology Vendors

Vendor Responsibilities

- Vendors must align their development roadmaps with emerging PQC standards.
- Testing and integration of quantum-resistant algorithms into products are crucial to the Secure by Design principle.

Key Actions for Organizations

- Actively engage with vendors on:
  - Transition timelines for both legacy and modern products.
  - Costs and implications of migration to PQC.

- Collaborate with cloud service providers to plan for PQC integration, ensuring compatibility and effective implementation.

Addressing Supply Chain Dependencies

Organizations should assess quantum-vulnerable cryptographic dependencies across their supply chain, prioritizing:

- **Critical Systems**: Industrial control systems (ICS) and long-term secrecy needs.
- **Custom Technologies**: Custom-built solutions require bespoke upgrades to migrate to PQC.
- **Cloud Services**: Partner with providers to enable PQC configurations post-standards release.

Post-Quantum Cryptography (PQC) Migration Handbook

Chapter 1: Introduction

The **Post-Quantum Cryptography (PQC) Migration Handbook** serves as a practical guide for organizations to address the impending risks posed by quantum computers to current cryptographic systems. While numerous papers highlight the urgency and risks associated with quantum computing, this handbook provides actionable steps to develop a migration strategy, targeting organizations that need immediate preparedness, referred to as *urgent adopters*.

Key Concepts and Goals

1. **Cryptography and Its Importance**:
   - Cryptography underpins cybersecurity, safeguarding sensitive data, ensuring data integrity, and preventing unauthorized access.
   - Weak cryptography exposes organizations to risks such as data breaches, theft, and unauthorized access to sensitive systems.
2. **Threat of Quantum Computers**:
   - Classical cryptography, secure against traditional computing attacks, will become vulnerable with the advancement of quantum computers.
   - Within 10–20 years, quantum computers could render many current cryptographic algorithms obsolete.
3. **Post-Quantum Cryptography (PQC)**:
   - PQC schemes are resistant to quantum computing attacks, necessitating their adoption as the future standard.
   - Migration to PQC involves substantial resources (time, budget, manpower) and technical adaptations, especially for legacy systems.

Urgency of Migration

Three key risks justify early action:

1. **Store-Now-Decrypt-Later Attacks**:

    - Sensitive information intercepted today can be decrypted in the future using quantum computing.
    - Long-term sensitive data is already at risk.
2. **Long-Lived Systems**:
    - Systems developed today may not support future PQC updates, posing risks for critical infrastructure.
3. **Complexity of Migration**:
    - Historical migrations, like from SHA-1 to SHA-256, demonstrate that transitioning cryptographic systems is time-consuming and resource-intensive, requiring years of preparation.

Handbook Objectives

This document assists organizations in:

- Assessing risks posed by quantum computing to their cryptographic landscape.
- Outlining the steps needed for a successful migration to PQC.
- Addressing technical and strategic considerations for migration.

Associated Risks of Quantum Threats

- **Timing Uncertainty**: While quantum computers are not yet capable of breaking classical cryptography, breakthroughs may hasten their arrival.
- **Asset Lifespan**:
    - Assets requiring long-term confidentiality (e.g., sensitive data) are already vulnerable to quantum decryption attacks.
    - Other functionalities like authentication face threats only when quantum systems become operational.
- **Premature Migration Risks**: Migrating too early may result in repeated efforts if vulnerabilities are discovered in new PQC algorithms.
- **Delayed Migration Risks**: Late migration could lead to severe consequences, including data breaches and reputational damage.

Document Structure

The handbook follows a **three-step approach** for PQC migration:

1. **Diagnosis**:
    - Identify urgency and assess cryptographic dependencies within the organization using PQC personas and decision trees.
2. **Planning**:
    - Formulate technical and organizational migration strategies, prioritizing assets based on risk and urgency.
3. **Execution**:
    - Implement migration strategies with detailed technical guidance.

The subsequent chapters provide:

- **Chapter 2**: PQC diagnosis, urgency assessment, and inventory creation.
- **Chapter 3**: Migration planning at technical and organizational levels.
- **Chapter 4**: Technical strategies for migrating cryptographic algorithms.

- **Chapter 5**: Technical references for cryptographic schemes.

Quantum Key Distribution (QKD) vs. PQC

While QKD offers quantum-resistant key exchange, its practicality is limited due to:

- High infrastructure costs and reliance on physical communication channels.
- Security vulnerabilities in QKD devices.
- Inability to replace classical cryptography entirely.

PQC is considered the superior and more practical solution for quantum threats, endorsed by major security agencies.

Key Recommendations

- Early diagnosis of cryptographic assets and vulnerabilities to determine migration urgency.
- Preparation for PQC adoption by prioritizing assets, establishing migration plans, and allocating resources.
- Collaboration with vendors and stakeholders to ensure cryptographic agility and compatibility with PQC standards.

Chapter 2: Diagnosis

This chapter outlines the foundational steps for determining an organization's need to migrate to Post-Quantum Cryptography (PQC). It provides tools and frameworks for assessing the urgency of migration and conducting a comprehensive diagnosis of an organization's cryptographic landscape.

2.1 PQC Personas

To streamline PQC adoption, organizations are categorized into *PQC Personas*, reflecting their migration urgency based on assets, risks, and dependencies:

1. **Urgent Adopters**:
   - Handle sensitive or long-lived data or critical infrastructure.
   - Immediate action is required to safeguard against quantum threats like *store-now-decrypt-later* attacks.
   - Subcategories include:
     - **Personal Data Handlers**: Focus on long-term protection of individual data (e.g., healthcare, finance).
     - **Organizationally Sensitive Data Handlers**: Secure organizational data like trade secrets or state information (e.g., governments, military).
     - **Critical Infrastructure Providers**: Maintain essential services (e.g., energy, water, telecoms).
     - **Long-Lived Infrastructure Providers**: Develop systems with a lifespan beyond 20 years requiring future-proofing (e.g., satellites, payment systems).
2. **Regular Adopters**:
   - Do not face immediate quantum risks but should monitor developments and maintain crypto-agility.
   - Often includes retailers, schools, and smaller organizations with lower data sensitivity.

3. **Cryptography Experts**:
    - Develop and supply cryptographic tools and solutions.
    - Must prepare quantum-safe products and communicate readiness to clients.

Determining Your Persona

Organizations should evaluate:

- Their cryptographic infrastructure, knowledge, and dependency on others.
- Supply-chain risks and inherited personas from suppliers or clients.

Recommendation: Organizations on the boundary between urgent and regular adopters should err on the side of caution and consider initiating the diagnostic process early.

2.2 PQC Diagnosis

For organizations identified as **Urgent Adopters**, the PQC diagnosis is the first step in migration. It focuses on gathering information to assess risks, prioritize assets, and plan mitigation strategies. The diagnosis involves the following key steps:

1. **Risk Assessment**:
    - Reassess risks with the quantum threat in mind.
    - Identify new vulnerabilities in cryptographic algorithms and anticipate quantum-related attacks.
2. **Inventory of Cryptographic Assets**:
    - Compile a detailed list of all cryptographic assets, including:
        - Algorithms, key lengths, and usage.
        - Dependencies on external suppliers.
    - Utilize tools like Configuration Management Databases (CMDBs) and automated discovery tools.
3. **Inventory of Data Assets**:
    - Categorize and assess data based on:
        - Type (e.g., at rest, in transit).
        - Location, value, and sensitivity.
        - Classification and risk assessment.
4. **Inventory of Cryptographic Dependencies**:
    - Identify suppliers of cryptographic assets and ensure their readiness for PQC.
    - Establish communication with vendors to understand their migration plans.
    - Consider supply-chain risks, including dependencies on third-party cryptographic infrastructure.

Key Considerations

1. **Interoperability**:

   - Organizations within interconnected ecosystems should coordinate migration efforts to maintain security and operational compatibility.
2. **Multiple Personas**:
   - Some organizations may identify with multiple personas (e.g., financial institutions as both Personal and Organizational Data Handlers). Action steps should prioritize assets based on specific risks.
3. **Regular Adopters**:
   - While immediate migration is unnecessary, regular adopters should:
     - Maintain crypto-agility.
     - Stay informed on PQC standards and timelines.
     - Begin preparatory steps such as risk assessment and asset inventory.
4. **Cryptography Experts**:
   - Experts should actively transition products to PQC and support client readiness.
   - Communicate timelines and quantum-safety measures to ensure downstream compliance.

Running the PQC Diagnosis

The diagnosis phase is a critical precursor to migration. It provides insights into:

- Which assets to prioritize for quantum-safe upgrades.
- Dependencies on vendors and their readiness.
- Data and infrastructure vulnerabilities requiring urgent mitigation.

Outcome: The results of the diagnosis guide the development of a strategic and technical migration plan in subsequent phases.

Chapter 3: Migration Planning

This chapter provides detailed guidance for planning the transition to Post-Quantum Cryptography (PQC). It focuses on when and how organizations, particularly urgent adopters, should initiate the migration process, ensuring a structured and efficient approach.

3.1 When to Start Migrating?

3.1.1 Migration Scenarios

The timeline for migration depends on three critical factors:

- **X**: The time the asset must remain secure.
- **Y**: The time required for migration.
- **Z**: The estimated time until quantum computers can break current cryptographic systems.

Migration should satisfy **Mosca's inequality: $X + Y < Z$**. If **$X > Z$**, immediate migration is essential.

Four Migration Scenarios

Based on milestones in PQC development, organizations can choose from:

1. **Scenario 1**: Immediate migration using uncertified libraries.
2. **Scenario 2a/2b/2c**: Migration during the availability of production-level or community-tested PQC libraries.
3. **Scenario 3**: Migration after certified libraries and standards are established.

Key Considerations:

- Earlier migration (Scenario 1 or 2) involves risks like using uncertified libraries, but may be necessary for high-risk assets.
- Later migration (Scenario 3) ensures stability but risks exposure to quantum threats during the waiting period.
- Organizations must balance their risk tolerance, asset criticality, and timeline estimates.

Estimating Timelines

- **Wi (Waiting Time)**: Based on milestones for production-ready or certified PQC solutions. Organizations can influence these timelines through vendor engagement.
- **Yi (Migration Time)**: Varies by asset and complexity of implementation.
- **Z (Quantum Breakthrough Timeline)**: Expert consensus suggests 2030–2040 as potential milestones for quantum computers breaking RSA-2048.

Step-by-Step Process

1. **Estimate Timelines**: Use available data and expert opinions to approximate Wi, Yi, and Z.
2. **Determine Migration Scenario**: Use a decision tree to align assets with the most appropriate scenario.
3. **Develop a General Strategy**:
   - Begin with modernizing cryptographic systems.
   - Transition to PQC incrementally, starting with the most critical assets.

3.2 Advice on Migration Planning

3.2.1 Business Process Planning

Migration involves significant business considerations:

- **Migration Manager**: Appoint a leader with organization-wide access to oversee the transition.
- **Budget Allocation**: Plan for costs related to personnel, infrastructure, and potential system replacements.
- **Downtime Management**: Anticipate and minimize service interruptions by coordinating with stakeholders and similar organizations.

Costs

Migration is resource-intensive, requiring:

- Dedicated teams to inventory and prioritize cryptographic assets.
- Potential hardware upgrades to support PQC algorithms, which often require more computational resources.
- Contingency planning for vendor delays or insufficient PQC support.

Interoperability

- Coordinate with partners and stakeholders to maintain compatibility during the migration.

- Collaborative planning can reduce workloads and ensure smooth transitions.

3.2.2 Technical Planning

Dependency of Assets

- Identify interdependencies between cryptographic assets.
- Maintain interoperability by temporarily supporting dual systems (classical and quantum-safe protocols).

Cryptography Replacement

- Decide whether to replace, redesign, or retire each cryptographic asset.
- Select quantum-safe solutions that are crypto-agile, ensuring flexibility for future updates.

Asset Isolation

For particularly sensitive assets:

- Use isolation to protect against attacks during migration.
- Employ physical or logical separation, though this may disrupt functionality.

Hardware Replacement

- Assess the capability of existing hardware to support PQC.
- Plan for deployment timelines if hardware upgrades are necessary.

Testing

- Thoroughly test new algorithms and hardware to ensure compatibility and security.
- Incorporate testing into the migration timeline to prevent post-migration vulnerabilities.

Key Takeaways

- **Timely Migration**: Organizations must evaluate the urgency of migration based on the criticality of their assets and their estimated timelines for quantum threats.
- **Structured Planning**: Prioritize high-risk assets, allocate sufficient resources, and coordinate with stakeholders to ensure smooth transitions.
- **Flexibility**: Opt for crypto-agile solutions to accommodate evolving PQC standards and mitigate future risks.

Chapter 4: Execution Summary

Chapter 4 provides a comprehensive guide for executing the migration to post-quantum cryptography (PQC). This process involves general strategies, detailed steps for migrating cryptographic primitives and protocols, and an emphasis on ensuring cryptographic agility. Key highlights include:

4.1 General Strategies

- **Long Process Awareness**: Migration is a multi-year endeavor. Start with planning and updating cryptographic inventories.
- **Dynamic IT Environments**: Continuously update asset inventories to reflect current cryptographic use.

- **Careful Implementation**: Missteps in cryptographic replacement can introduce vulnerabilities. Hybrid schemes combining classical and PQC algorithms are recommended for early stages.

4.2 Cryptographic Agility

Cryptographic agility enables quick adaptation to new cryptographic standards without major architectural changes. Steps include:

1. Abstract cryptographic operations using high-level libraries or external key management solutions.
2. Conduct cryptographic agility scans as part of CI/CD pipelines.
3. Evaluate hardware readiness for PQC's computational demands.

4.3 Migration of Primitives vs. Protocols

- **Primitives**: Low-level algorithms (e.g., RSA, AES) form the foundation of cryptographic protocols.
- **Protocols**: Systems like TLS, SSH, and IPSec that integrate primitives.

Migration of Primitives

1. **Symmetric Cryptography and Hash Functions**:
   - Increase key lengths and hash outputs to mitigate quantum threats (e.g., AES-256, SHA-3-256).
   - Update long-lived systems that cannot be modified later.
2. **Asymmetric Cryptography**:
   - **Hybrid Solutions**: Use classical and PQC algorithms in tandem to ensure security during transition.
   - Avoid "hybrid OR" setups prone to downgrade attacks.
   - Consider pre-shared keys for environments with strict trust and control, but acknowledge scalability limits.

Migration of Protocols

Protocols require adaptation to ensure quantum safety:

1. **TLS**:
   - Options: Pre-shared keys or hybrid key exchange.
   - Configure cipher suites for quantum-safe encryption (e.g., AES-256-GCM).
2. **SSH**:
   - Employ hybrid key exchange for secure remote access.
3. **S/MIME**:
   - Limited research; hybrid approaches or avoiding sensitive email exchanges are recommended.
4. **PGP**:
   - Await further research and developments for quantum-safe alternatives.
5. **IPSec**:
   - Use pre-shared keys or hybrid solutions for secure VPN communication.
6. **X.509 Certificates**:

– Transition to hybrid certificates for authenticating public keys.

Challenges and Recommendations

- **Downgrade Attacks**: Hybrid OR configurations may allow attackers to bypass post-quantum algorithms.
- **Pre-shared Keys**: High-security but impractical for large-scale systems due to logistical challenges.

Table of Recommended, Acceptable, and Prohibited Cryptographic Primitives

The handbook provides a detailed table for cryptographic primitives categorized by functionality (e.g., block ciphers, stream ciphers, digital signatures). Highlights include:

- **Recommended**: AES-256, CRYSTALS-KYBER (PQC public-key encryption).
- **Acceptable**: Camellia-256, FrodoKEM (conservative PQC alternatives).
- **Do Not Use**: RSA, DSA, and MD5, as they lack quantum resistance.

Chapter 5: Background on Primitives

Purpose of the Chapter

This chapter provides guidance for library developers and organizations on selecting cryptographic primitives for quantum-safe protocols. It also aids in:

- **Asset discovery**: Identifying cryptographic assets in use.
- **Risk assessment**: Evaluating the vulnerabilities of these assets. The chapter assumes familiarity with the cryptographic landscape and provides a categorized list of commonly used primitives, their characteristics, and their quantum security status.

5.1 Classical Primitives

Classical primitives are categorized into:

1. **Symmetric Ciphers**
2. **Asymmetric Ciphers**
3. **Hash Functions**
4. **Message Authentication Codes (MACs)**
5. **Stateful Hash-Based Signatures (HBS)**

5.1.1 Symmetric Ciphers

- **AES**: Widely used block cipher supporting 128, 192, and 256-bit keys. It is quantum-secure with 256-bit keys.
- **(T)DES**: Deprecated due to its short key lengths. Insecure against both classical and quantum attacks.
- **ChaCha20**: Stream cipher known for speed and simplicity. Quantum-secure with 256-bit keys.
- **Blowfish**: Legacy cipher with small block sizes, making it vulnerable to birthday attacks.
- **RC4**: Insecure in classical settings and not recommended.
- **Camellia**: Secure alternative to AES with similar features. Quantum-secure with 256-bit keys.

5.1.2 Asymmetric Ciphers

- **RSA**: Based on integer factorization. Insecure against quantum attacks.
- **ElGamal**: Relies on the Diffie-Hellman problem. Quantum-unsafe.
- **ECDSA/ECDH**: Elliptic curve variants for digital signatures and key exchange. Both are quantum-unsafe.
- **EdDSA**: Modern digital signature scheme, but quantum-unsafe.

5.1.3 Hash Functions

- **SHA Family**: Includes SHA-2 and SHA-3, quantum-secure if 256-bit or higher output sizes are used.
- **MD5**: Deprecated due to vulnerabilities. Insecure even in classical settings.
- **BLAKE2**: Faster alternative to SHA-3. Quantum-secure with 256-bit or higher outputs.

5.1.4 Message Authentication Codes (MACs)

- **HMAC**: Constructed using cryptographic hashes. Quantum-secure if underlying hash is quantum-safe.
- **BLAKE2-MAC**: Faster than HMAC due to integrated keying. Quantum-secure with BLAKE2b.
- **CBC-MAC and CMAC**: Derived from block ciphers, both quantum-secure with 128-bit or higher quantum-safe hashes.
- **Poly1305**: High-speed MAC paired with ChaCha20. Quantum-secure.

5.1.5 Stateful Hash-Based Signatures (HBS)

- **XMSS and XMSSMT**: Based on Merkle hash trees, suitable for one-time and limited-use signatures. Quantum-secure but require state management.
- **LMS and HSS**: Variants of stateful hash-based signatures. Quantum-secure with proper implementation.

5.3 Post-Quantum Primitives

Post-quantum primitives address vulnerabilities in classical cryptography caused by quantum computing. They are classified into:

1. **Digital Signature Schemes (DSS)**
2. **Key Exchange Mechanisms (KEMs)**

5.3.1 Digital Signature Schemes

- **CRYSTALS-Dilithium**: Lattice-based with high confidence and security levels. Suitable for general use.
- **FALCON**: Lattice-based, efficient but requires floating-point operations.
- **SPHINCS+**: Stateless hash-based signature with reduced sizes. High confidence and security.

5.3.2 Key Exchange Mechanisms

- **BIKE**: Code-based KEM with high security but large key sizes. Still under evaluation.
- **Classic McEliece**: Code-based KEM with very large public keys but small ciphertexts. Suitable for specific use cases.
- **CRYSTALS-Kyber**: Lattice-based KEM with moderate key and ciphertext sizes. High confidence and efficiency.
- **FrodoKEM**: Lattice-based KEM known for conservative security but large key sizes. Not selected for NIST standardization.

- **HQC**: Code-based KEM focusing on strong theoretical foundations and moderate key sizes.

Recommendations for PQC Migration

1. **Cryptographic Inventory**:
   - Identify all cryptographic primitives in use.
   - Evaluate their quantum security status.
2. **Selection of Primitives**:
   - Transition to quantum-safe primitives, prioritizing those with high confidence and standardization (e.g., CRYSTALS-Dilithium and Kyber).
   - Replace deprecated algorithms like MD5, RC4, and (T)DES.
3. **Performance and Scalability**:
   - Balance security with operational efficiency.
   - Consider resource-intensive primitives carefully (e.g., McEliece's large keys).
4. **Implementation Considerations**:
   - Ensure proper state management for stateful signatures.
   - Opt for stateless signatures (e.g., SPHINCS+) where possible.

Part 2: Cryptographic Inventory and Risk Assessment

Introduction

This report documents the implementation and results of **Part 2** of the project, focusing on developing a **Cryptographic Inventory Tool** capable of detecting, prioritizing, and managing vulnerabilities in cryptographic primitives. We have extended our implementation to fully cover the required features and bonus tasks as described in the project guidelines.

Building upon concepts outlined in **Part 1**, the project leverages principles of cryptographic agility and post-quantum cryptography (PQC) readiness. Our approach employs both **pattern-based analysis** and **semantic inspection** for identifying vulnerabilities, with Python as the target programming language to ensure thorough detection.

Features

Comprehensive Vulnerability Detection

The tool detects a range of vulnerabilities in Python code by combining two methods:

- **Pattern Matching**: Quickly identifies cryptographic issues using predefined expressions.
- **Semantic Inspection**: Provides deeper analysis, capturing vulnerabilities involving parameters, configurations, or logic spread across multiple lines.

This dual-layered approach ensures robust coverage of potential weaknesses, balancing speed and accuracy in detection.

Risk Prioritization

The tool categorizes vulnerabilities by severity:

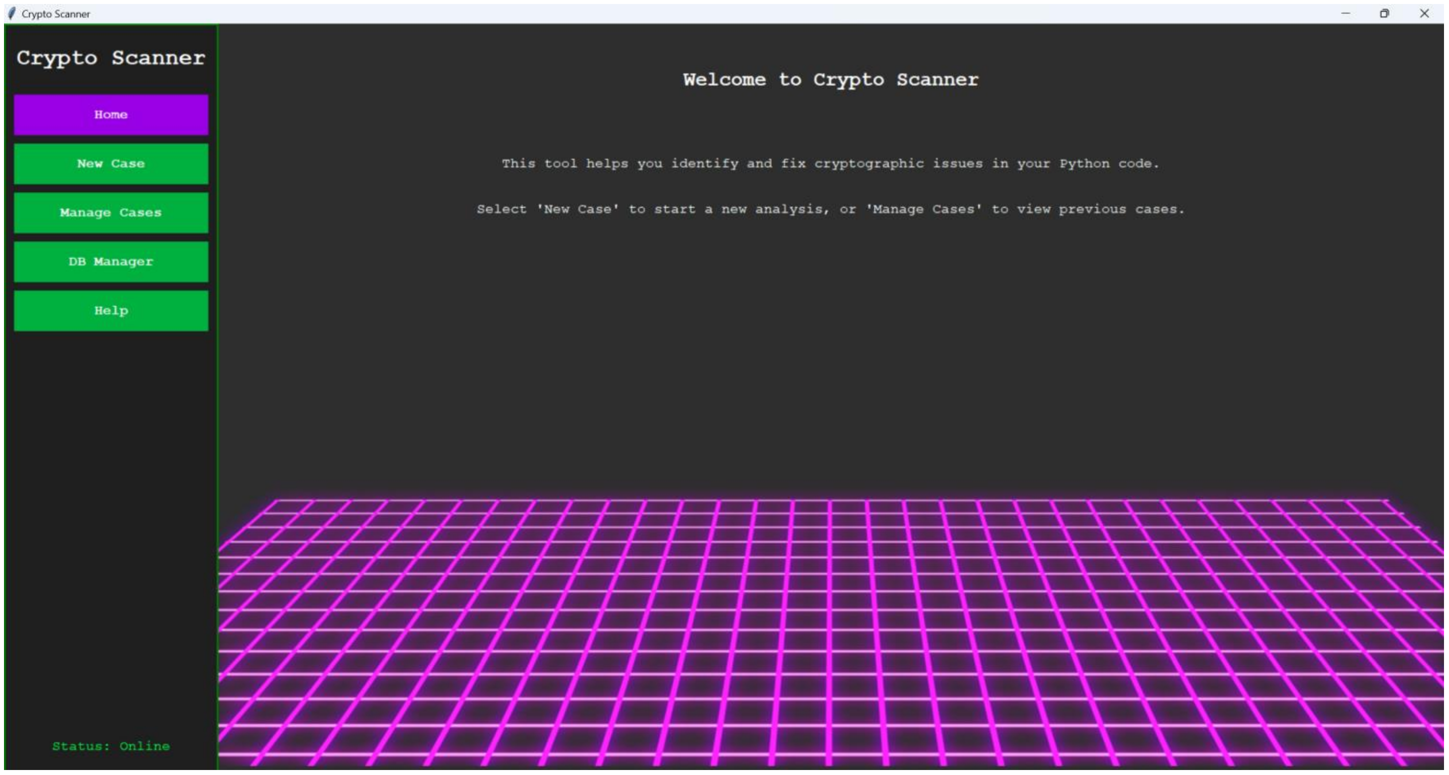
- **Critical**: Requires immediate attention.
- **High**: Significant but less urgent.
- **Medium**: Requires action in the medium term.
- **Low**: No immediate action required.

Risk levels are dynamically assigned based on vulnerability details, and findings are flagged as **quantum-vulnerable** when applicable. To aid decision-making, results can be ordered by severity and visualized with colored tags for each risk level.

User Interface (GUI)

The GUI provides an intuitive platform for:

- Selecting directories for scanning.
- Viewing and filtering results by severity, issue type, and file name.
- Searching findings by file name or issue details.
- Exporting findings to CSV for reporting or further analysis.
- Managing cases, including creating, loading, and deleting investigations.



Welcome to Crypto Scanner

This tool helps you identify and fix cryptographic issues in your Python code.

Select 'New Case' to start a new analysis, or 'Manage Cases' to view previous cases.

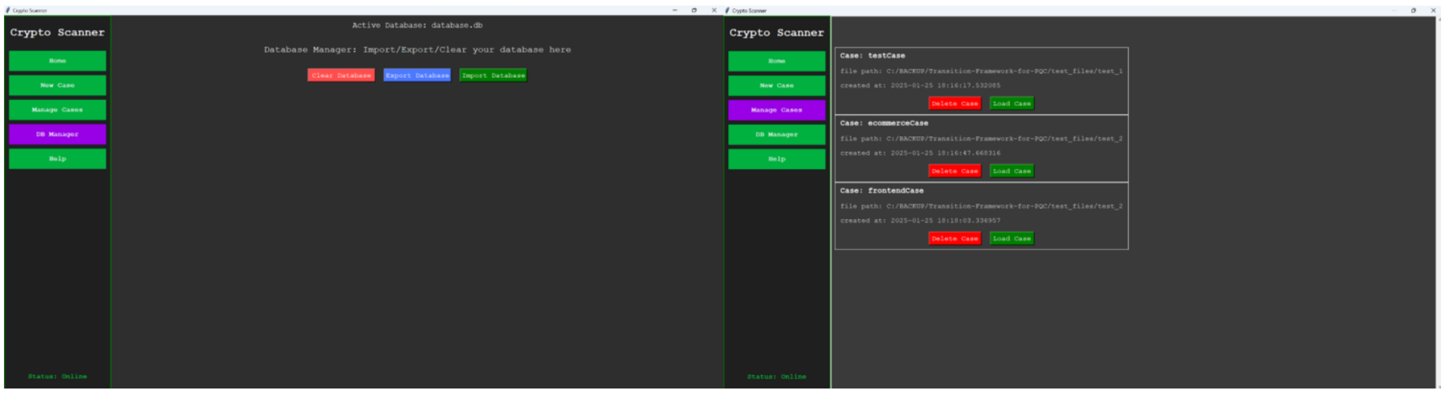
Bonus Features

1. Expanded Vulnerability Detection

Additional patterns and semantic rules identify issues such as deprecated ECC curves, weak bcrypt parameters, and hardcoded private keys.

2. Case Management

Users can create and manage specific cases, with findings stored under unique names for future review or updates.



Active Database: database.db

Database Manager: Import/Export/Clear your database here

Status: Online

Case: testCase
file path: C:/BACKUP/Transition-Framework-for-PQC/test_files/test_1
created at: 2025-01-25 18:16:17.532085

Case: ecommerceCase
file path: C:/BACKUP/Transition-Framework-for-PQC/test_files/test_2
created at: 2025-01-25 18:16:47.668316

Case: frontendCase
file path: C:/BACKUP/Transition-Framework-for-PQC/test_files/test_2
created at: 2025-01-25 18:18:03.336957

Status: Online

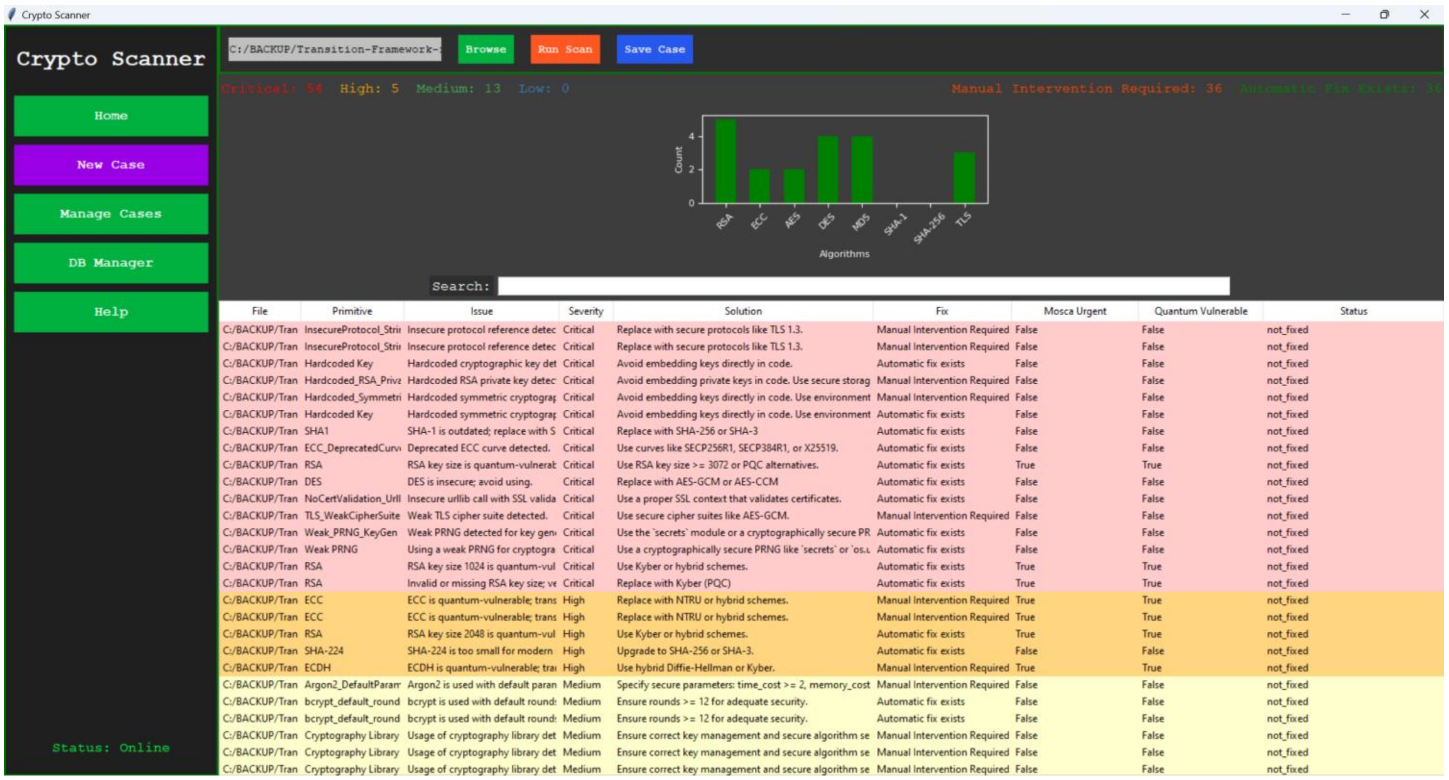
3. Database Management

Functionalities include:

- Exporting findings to CSV.
- Importing data from previously exported files.
- Clearing the database for fresh investigations.

4. Enhanced Visualization

The GUI includes statistical breakdowns of vulnerabilities by severity, accompanied by bar charts for improved analysis.



C:/BACKUP/Transition-Framework-

Critical: 54 High: 5 Medium: 13 Low: 0 Manual Intervention Required: 36 Automatic Fix Exists: 36

File	Primitive	Issue	Severity	Solution	Fix	Mosca Urgent	Quantum Vulnerable	Status
C:/BACKUP/Tran	InsecureProtocol_Stri	Insecure protocol reference detec	Critical	Replace with secure protocols like TLS 1.3.	Manual Intervention Required	False	False	not_fixed
C:/BACKUP/Tran	InsecureProtocol_Stri	Insecure protocol reference detec	Critical	Replace with secure protocols like TLS 1.3.	Manual Intervention Required	False	False	not_fixed
C:/BACKUP/Tran	Hardcoded Key	Hardcoded cryptographic key det	Critical	Avoid embedding keys directly in code.	Automatic fix exists	False	False	not_fixed
C:/BACKUP/Tran	Hardcoded_RSA_Priva	Hardcoded RSA private key detec	Critical	Avoid embedding private keys in code. Use secure storag	Manual Intervention Required	False	False	not_fixed
C:/BACKUP/Tran	Hardcoded_Symmetri	Hardcoded symmetric cryptogra	Critical	Avoid embedding keys directly in code. Use environment	Manual Intervention Required	False	False	not_fixed
C:/BACKUP/Tran	Hardcoded Key	Hardcoded symmetric cryptogra	Critical	Avoid embedding keys directly in code. Use environment	Automatic fix exists	False	False	not_fixed
C:/BACKUP/Tran	SHA1	SHA-1 is outdated; replace with S	Critical	Replace with SHA-256 or SHA-3	Automatic fix exists	False	False	not_fixed
C:/BACKUP/Tran	ECC_DeprecatedCurv	Deprecated ECC curve detected.	Critical	Use curves like SECP256R1, SECP384R1, or X25519.	Automatic fix exists	False	False	not_fixed
C:/BACKUP/Tran	RSA	RSA key size is quantum-vulnerab	Critical	Use RSA key size >= 3072 or PQC alternatives.	Automatic fix exists	True	True	not_fixed
C:/BACKUP/Tran	DES	DES is insecure; avoid using.	Critical	Replace with AES-GCM or AES-CCM	Automatic fix exists	False	False	not_fixed
C:/BACKUP/Tran	NoCertValidation_Urll	Insecure urllib call with SSL valida	Critical	Use a proper SSL context that validates certificates.	Automatic fix exists	False	False	not_fixed
C:/BACKUP/Tran	TLS_WeakCipherSuite	Weak TLS cipher suite detected.	Critical	Use secure cipher suites like AES-GCM.	Manual Intervention Required	False	False	not_fixed
C:/BACKUP/Tran	Weak_PRNG_KeyGen	Weak PRNG detected for key gen	Critical	Use the `secrets` module or a cryptographically secure PR	Automatic fix exists	False	False	not_fixed
C:/BACKUP/Tran	Weak PRNG	Using a weak PRNG for cryptogra	Critical	Use a cryptographically secure PRNG like `secrets` or `os.u	Automatic fix exists	False	False	not_fixed
C:/BACKUP/Tran	RSA	RSA key size 1024 is quantum-vul	Critical	Use Kyber or hybrid schemes.	Automatic fix exists	True	True	not_fixed
C:/BACKUP/Tran	RSA	Invalid or missing RSA key size; ve	Critical	Replace with Kyber (PQC)	Automatic fix exists	True	True	not_fixed
C:/BACKUP/Tran	ECC	ECC is quantum-vulnerable; trans	High	Replace with NTRU or hybrid schemes.	Manual Intervention Required	True	True	not_fixed
C:/BACKUP/Tran	ECC	ECC is quantum-vulnerable; trans	High	Replace with NTRU or hybrid schemes.	Manual Intervention Required	True	True	not_fixed
C:/BACKUP/Tran	RSA	RSA key size 2048 is quantum-vul	High	Use Kyber or hybrid schemes.	Automatic fix exists	True	True	not_fixed
C:/BACKUP/Tran	SHA-224	SHA-224 is too small for modern	High	Upgrade to SHA-256 or SHA-3.	Automatic fix exists	False	False	not_fixed
C:/BACKUP/Tran	ECDH	ECDH is quantum-vulnerable; tra	High	Use hybrid Diffie-Hellman or Kyber.	Manual Intervention Required	True	True	not_fixed
C:/BACKUP/Tran	Argon2_DefaultParam	Argon2 is used with default paran	Medium	Specify secure parameters: time_cost >= 2, memory_cost	Manual Intervention Required	False	False	not_fixed
C:/BACKUP/Tran	bcrypt_default_round	bcrypt is used with default round	Medium	Ensure rounds >= 12 for adequate security.	Automatic fix exists	False	False	not_fixed
C:/BACKUP/Tran	bcrypt_default_round	bcrypt is used with default round	Medium	Ensure rounds >= 12 for adequate security.	Automatic fix exists	False	False	not_fixed
C:/BACKUP/Tran	Cryptography Library	Usage of cryptography library det	Medium	Ensure correct key management and secure algorithm se	Manual Intervention Required	False	False	not_fixed
C:/BACKUP/Tran	Cryptography Library	Usage of cryptography library det	Medium	Ensure correct key management and secure algorithm se	Manual Intervention Required	False	False	not_fixed
C:/BACKUP/Tran	Cryptography Library	Usage of cryptography library det	Medium	Ensure correct key management and secure algorithm se	Manual Intervention Required	False	False	not_fixed

Status: Online

Design and Implementation

Vulnerability Detection

he tool employs two complementary methods:

- **Pattern Matching**: Identifies cryptographic issues through predefined expressions, such as locating the use of AES-ECB or static IVs. For example:
  - **AES-ECB Mode**: `\bAES\.new\(.*?,\s*AES\.MODE_ECB\)`
  - **Static IVs**: `IV=(0x[a-fA-F0-9]+)`

  These patterns ensure rapid detection of vulnerabilities in languages where AST parsing is unavailable.

- **Semantic Inspection**: Analyzes code structures using Abstract Syntax Trees (AST) to uncover issues like insecure parameter configurations, missing authentication tag verifications, and hardcoded keys. For example:

  ```
  cipher = AES.new(key, AES.MODE_GCM)
  data = cipher.decrypt(encrypted_data)  # Missing tag verification detected
  ```

  This approach adds depth and precision, complementing Regex-based scanning.

Cryptographic Inventory Tool

The tool was developed in **Python**, leveraging its extensive library support and suitability for rapid prototyping. It is designed with modularity and clean code principles to ensure maintainability, extensibility, and ease of collaboration.

Key components and libraries include:

- **`tkinter`**: Provides an intuitive graphical user interface for tasks such as selecting directories, viewing results, and managing cases.
- **`re`**: Facilitates pattern-based matching to quickly identify cryptographic vulnerabilities in source code.
- **`ast`**: Enables semantic analysis of Python code, uncovering vulnerabilities that may not be detected through simple pattern recognition.
- **`sqlite3`**: Ensures persistent and lightweight database management for findings, supporting features like case management, data import/export, and summary statistics.

The implementation emphasizes a clear separation of concerns:

- **`CryptoAnalyzer`**: Combines regex-based and AST-based analysis to detect vulnerabilities with precision and depth.
- **`DatabaseManager`**: Handles interactions with the SQLite database, providing storage and retrieval mechanisms for findings, as well as exporting capabilities.
- **`CryptoScannerGUI`**: Acts as the interface layer, allowing users to interact with the tool easily, manage cases, and visualize findings.

Risk Prioritization with Mosca's Inequality

To assess the urgency of transitioning from vulnerable cryptographic primitives, the tool integrates **Mosca's Inequality**:

$$X + Y \geq Z$$

Where:

- **X**: Time required to replace cryptography.
- **Y**: Useful lifetime of sensitive data.
- **Z**: Time until a quantum computer can break current cryptographic standards.

This framework is applied to findings involving quantum-vulnerable primitives such as RSA and ECC. For such primitives, the tool flags vulnerabilities as **Mosca Urgent** when $ X + Y \geq Z $, prompting immediate attention. This prioritization ensures that the most critical vulnerabilities, especially those posing quantum threats, are addressed first.

The combination of a robust framework and a risk assessment mechanism like Mosca's Inequality equips the tool to effectively manage cryptographic agility challenges and prepare systems for a post-quantum era.

Vulnerabilities Detected by Cryptographic Scanner

The **Cryptographic Scanner** is designed to identify a wide array of cryptographic vulnerabilities, ranging from outdated algorithms to insecure implementations. This section provides a detailed explanation of the vulnerabilities detected by the scanner.

1. Symmetric Ciphers

Data Encryption Standard (DES)

- **Severity**: Critical
- **Issue**: DES is an outdated symmetric cipher with a 56-bit key, vulnerable to brute-force attacks.
- **Recommendation**: Replace DES with AES-GCM or AES-CCM for secure encryption.

Triple DES (3DES)

- **Severity**: Critical
- **Issue**: 3DES is insecure and vulnerable to brute-force attacks. Variants with fewer than three independent keys are particularly weak.
  - **3DES with 1 Key**: Equivalent to DES.
  - **3DES with 2 Keys**: Provides only 80-bit security, which is inadequate.
  - **3DES with 3 Keys**: Deprecated and quantum-vulnerable.
- **Recommendation**: Replace with AES-GCM or AES-CCM.

AES with Insecure Parameters

- **AES-128/192**: Medium severity due to quantum vulnerability.
- **AES-ECB Mode**: Critical severity as ECB leaks plaintext patterns.
- **Static IVs in AES-CBC Mode**: Critical severity due to IV reuse leading to predictable ciphertexts.
- **Recommendation**: Use AES-256 in GCM or CCM mode with randomized IVs.

Static IV Detection in Assignments or Concatenations

- **Severity**: Critical
- **Issue**: Reuse of static IVs compromises ciphertext security.
- **Recommendation**: Always use a randomized IV for each encryption operation.

Blowfish

- **Severity**: Critical (key size < 128 bits), Low (key size ≥ 128 bits).
- **Issue**: Blowfish is outdated; short keys are especially insecure.

- **Recommendation**: Use AES-256 instead.

RC4

- **Severity**: Critical
- **Issue**: RC4 is insecure due to biases in its output, leading to plaintext recovery.
- **Recommendation**: Avoid using RC4 altogether.

---

2. Asymmetric Ciphers

RSA

- **Severity**: High to Critical
- **Issue**: Key sizes < 2048 bits are quantum-vulnerable and insecure. RSA without padding is also vulnerable to attacks.
- **Recommendation**: Use RSA with a key size ≥ 3072 bits and OAEP or PSS padding. Transition to post-quantum cryptography (e.g., Kyber).

Elliptic Curve Cryptography (ECC)

- **Severity**: High
- **Issue**: ECC is quantum-vulnerable. Deprecated curves (e.g., SECP112R1) are particularly insecure.
- **Recommendation**: Use modern curves (e.g., SECP256R1) or transition to post-quantum alternatives.

Deprecated ECC Curves

- **Severity**: Critical
- **Issue**: Deprecated ECC curves (e.g., SECP112R1) are insecure and should be avoided.
- **Recommendation**: Use modern curves like SECP256R1 or X25519.

Diffie-Hellman (DH)

- **Severity**: Critical
- **Issue**: Weak parameters such as small modulus sizes or generators can be exploited.
- **Recommendation**: Use a secure modulus (≥ 2048 bits) and generator values ≥ 2.

Weak Diffie-Hellman Generator

- **Severity**: Critical
- **Issue**: Generator values of 1 or (p-1) are insecure and weaken the key exchange.
- **Recommendation**: Use a secure generator value (e.g., 2).

---

3. Hash Functions

MD5

- **Severity**: Critical
- **Issue**: MD5 is vulnerable to collisions and should not be used for security purposes.
- **Recommendation**: Replace with SHA-256 or SHA-3.

SHA-1

- **Severity**: Critical
- **Issue**: SHA-1 is vulnerable to collision attacks and is no longer secure.
- **Recommendation**: Replace with SHA-256 or SHA-3.

Whirlpool

- **Severity**: Medium
- **Issue**: Whirlpool is secure but uncommon, potentially leading to interoperability issues.
- **Recommendation**: Use SHA-3 for wider support.

---

4. Weak Cryptographic Modes

ECB Mode

- **Severity**: Critical
- **Issue**: ECB mode leaks patterns in plaintext, compromising confidentiality.
- **Recommendation**: Use GCM or CCM instead.

CBC Mode with Static IV

- **Severity**: Critical
- **Issue**: Reusing IVs in CBC mode enables ciphertext manipulation and pattern discovery.
- **Recommendation**: Use GCM or CCM with randomized IVs.

Missing GCM Tag Verification

- **Severity**: Critical
- **Issue**: Failing to verify the authentication tag in GCM mode exposes the ciphertext to tampering.
- **Recommendation**: Always verify authentication tags.

AES GCM Without Authentication Tag Verification

- **Severity**: Critical
- **Issue**: GCM mode used without verifying the authentication tag enables tampering.
- **Recommendation**: Always verify the authentication tag during decryption.

---

5. Deprecated Protocols

SSL/TLS

- **Severity**: Critical
- **Issue**: Deprecated versions (e.g., SSLv3, TLSv1.0) are insecure.
- **Recommendation**: Upgrade to TLS 1.3 with secure cipher suites.

TLS Weak Cipher Suite

- **Severity**: Critical
- **Issue**: Weak cipher suites like DES, 3DES, or RC4 compromise security in TLS communication.
- **Recommendation**: Use AES-GCM or ChaCha20 cipher suites.

SSH

- **Severity**: Critical
- **Issue**: Deprecated algorithms like `ssh-rsa` and `ssh-dss` are insecure.
- **Recommendation**: Use Ed25519 or modern RSA keys with ≥ 2048 bits.

IPsec

- **Severity**: Critical
- **Issue**: IKEv1 is deprecated and vulnerable to multiple attacks.

- **Recommendation**: Use IKEv2 with secure configurations.

Deprecated Protocol References

- **Severity**: Critical
- **Issue**: Strings referencing insecure protocols like `TLSv1.0`, `SSLv3`, or `IKEv1` indicate potential weaknesses.
- **Recommendation**: Update to modern protocols such as TLS 1.3.

---

6. Other Vulnerabilities

Missing Certificate Validation

- **Severity**: Critical
- **Issue**: SSL/TLS calls without certificate validation compromise security.
- **Recommendation**: Enable certificate validation to prevent man-in-the-middle attacks.

Hardcoded Keys

- **Severity**: Critical
- **Issue**: Embedding keys directly in code compromises security.
- **Recommendation**: Use environment variables or secure key management solutions.

Hardcoded Usernames

- **Severity**: Critical
- **Issue**: Hardcoded usernames in code increase the risk of exposure and compromise.
- **Recommendation**: Use environment variables or secure configuration files for user credentials.

Hardcoded RSA Private Key

- **Severity**: Critical
- **Issue**: PEM-encoded RSA private keys should not be embedded in code.
- **Recommendation**: Use secure key management solutions.

Weak PRNGs

- **Severity**: High
- **Issue**: Using weak PRNGs (e.g., `random.randint`) for cryptographic purposes is insecure.
- **Recommendation**: Use the `secrets` module or `os.urandom`.

Weak Argon2 Parameters

- **Severity**: Critical
- **Issue**: Parameters with low time cost, memory cost, or parallelism weaken password hashing.
- **Recommendation**: Use time_cost ≥ 2, memory_cost ≥ 65536, and parallelism ≥ 2.

Argon2 Default Parameters

- **Severity**: Medium
- **Issue**: Usage of Argon2 without explicitly specifying parameters defaults to insecure settings.
- **Recommendation**: Specify secure parameters: time_cost ≥ 2, memory_cost ≥ 65536, parallelism ≥ 2.

Weak bcrypt Rounds

- **Severity**: Critical
- **Issue**: bcrypt with rounds < 12 is computationally weak.

- **Recommendation**: Use rounds ≥ 12.

Deprecated API Usage

- **Severity**: Critical
- **Issue**: Deprecated APIs (e.g., `ssl.PROTOCOL_TLSv1`) weaken security.
- **Recommendation**: Replace with modern equivalents like `ssl.PROTOCOL_TLSv1_2`.

bcrypt Default Rounds

- **Severity**: Medium
- **Issue**: bcrypt without specifying rounds defaults to insufficient iterations.
- **Recommendation**: Specify rounds ≥ 12 for bcrypt.

Reuse of Key Material in KDFs

- **Severity**: Critical
- **Issue**: Reusing key material in KDFs compromises key derivation security.
- **Recommendation**: Use unique salts and diversify derivation inputs.

Missing Salt in Password Hashing

- **Severity**: Critical
- **Issue**: Password hashing without salt allows for dictionary attacks.
- **Recommendation**: Always use a unique, random salt.

Insufficient PBKDF2 Iterations

- **Severity**: Critical
- **Issue**: PBKDF2 with fewer than 100,000 iterations is computationally weak.
- **Recommendation**: Use PBKDF2 with ≥ 100,000 iterations.

Key Material Reuse in HKDF

- **Severity**: Critical
- **Issue**: Reusing the same key material in HKDF derivation compromises security.
- **Recommendation**: Use unique salts and diversify derivation inputs for each key derivation.

Conclusion

Our Cryptographic Inventory Tool successfully identifies, categorizes, and prioritizes vulnerabilities in Python code. By combining pattern-based and semantic methods, incorporating risk prioritization, and providing an intuitive GUI, the tool offers a comprehensive solution for managing cryptographic risks. The inclusion of bonus features enhances its utility, making it a robust framework for assessing cryptographic agility.

Part 3: Crypto Migration Planning

Introduction

This report outlines a detailed migration plan for addressing cryptographic vulnerabilities identified through a comprehensive scan of various Python test files. The findings are categorized by severity and quantum vulnerability, and the roadmap provides clear steps to transition from outdated or insecure cryptographic primitives to stronger, post-quantum cryptography (PQC) solutions.

1. Findings Summary

The cryptographic scan conducted in **Part 2** analyzed a total of **72 vulnerabilities** across Python source code files. These vulnerabilities were identified and categorized based on their severity, quantum vulnerability status, and specific cryptographic primitives or configurations used.

Total Files and Vulnerabilities

- **Number of Files Scanned**: 42 (Python Files Only)
- **Total Vulnerabilities Identified**: 72
  - **Critical Issues**: 54
  - **High Issues**: 5
  - **Medium Issues**: 13

Breakdown by Severity

- **Critical Issues (75% of findings)**: These vulnerabilities pose immediate risks to the confidentiality, integrity, or availability of the system and must be addressed as a top priority. Examples include:
  - **Outdated Cryptographic Primitives**:
    - Use of MD5 and SHA-1 for hashing.
    - DES and RC4 for encryption.
  - **Weak Configurations**:
    - AES in ECB mode, which leaks plaintext patterns.
    - Static IVs in CBC mode, leading to predictable ciphertext.
  - **Hardcoded Secrets**:
    - Embedded API keys and credentials directly in source code.
  - **Deprecated Protocols**:
    - Use of SSLv3 and TLSv1.0.
- **High Issues (7% of findings)**: These vulnerabilities relate to quantum vulnerability or deprecated practices that, while not immediately critical, require significant remediation. Examples include:
  - Use of **deprecated ECC curves** (e.g., SECP192R1).
  - Quantum vulnerability in RSA-2048 and ECDH key exchanges.
- **Medium Issues (18% of findings)**: These issues involve weak parameterization or general warnings for cryptographic usage. While not immediately exploitable, they weaken overall system security. Examples include:
  - Default parameters in **Argon2** and **bcrypt**.
  - Generic usage of cryptographic libraries requiring review.

Quantum Vulnerability Analysis

Out of the 72 vulnerabilities:

- **Quantum-Vulnerable Issues**: 30%
  - Examples:
    - ECC curves like SECP192R1 are vulnerable to quantum attacks.
    - RSA keys ≤ 2048 bits susceptible to Shor's algorithm.

Key Insights from Part 2 Results

- **Most Frequent Critical Issues**:
  - Outdated hash algorithms (MD5, SHA-1) appear in 12 findings, comprising **16.6% of total findings**.
  - AES in ECB mode is another recurring issue, observed in multiple files.
- **Notable Quantum Vulnerable Findings**:
  - RSA-1024 and RSA-2048 keys (10 occurrences) require migration to post-quantum algorithms such as Kyber.
  - Deprecated ECC curves like SECP192R1 flagged as needing replacement.
- **Common Weak Configurations**:
  - Static IVs in CBC mode and weak PRNG usage.
  - Hardcoded keys or credentials embedded in code.

Risk Reduction Potential

Remediation efforts, as outlined in the migration roadmap, aim to:

- **Eliminate 100% of critical vulnerabilities** through immediate fixes in Phase 1.
- **Address all quantum-vulnerable issues** by transitioning to post-quantum cryptography in Phase 3.
- **Achieve compliance** with cryptographic standards such as **NIST SP 800-57**, **NIST PQC standards**, and **ENISA PQC guidelines**.

To provide a clear overview of the vulnerabilities detected during the cryptographic inventory scan, the following table summarizes the issues identified in each Python file.

File	Primitive/Issue	Parameters	Issue Description	Severity	Suggestion	Quantum Vulnerable	Mosca Urgent
aes_ecb_mode.py	AES_ECB_Mode	AES.new(key, AES.MODE_ECB)	ECB mode leaks plaintext patterns.	Critical	Switch to AES-GCM or AES-CCM.	0	0
bcrypt_weak_rounds.py	bcrypt_weak_rounds, bcrypt	rounds=4	Weak bcrypt rounds.	Critical	Use bcrypt.gensalt(rounds=12) or higher.	0	0
blowfish_insecure.py	Blowfish_Weak Key	Blowfish.MODE_ECB	Weak Blowfish key detected.	Critical	Use a Blowfish key of at least 128 bits or switch to AES.	0	0
blowfish_static_iv.py	Blowfish_Weak Key	Blowfish.MODE_CBC, iv	Weak Blowfish key	Critical	Use a Blowfish key of at least	0	0

			detected.		128 bits or switch to AES.		
deprecated_protocols_usage.py	TLS, InsecureProtocol_Strings	SSLv3	Deprecated TLS version detected.	Critical	Upgrade to TLS 1.3 with PQC support.	0	0
dh_vulnerable_params.py	DH_WeakParams, DH_WeakGenerator, RSA	key_size=1024	Weak Diffie-Hellman parameters and quantum-vulnerable RSA keys.	Critical	Use secure parameters (>= 2048 bits) and switch to PQC.	1	1
diffie_hellman.py	DH_WeakGenerator	2	Weak Diffie-Hellman generator detected.	Critical	Use generator=2 or higher. Avoid 1 or (p-1).	0	0
ecc_deprecated_curve.py	ECC_DeprecatedCurve, ECC	SECP192R1	Deprecated ECC curve and quantum vulnerability detected.	High	Replace with NTRU or hybrid schemes.	1	1
hardcoded_credentials.py	Hardcoded_Credentials	admin	Hardcoded credentials detected.	Critical	Avoid embedding credentials in code. Use environment variables.	0	0
hmac_with_md5_sha1.py	MD5, SHA1	MD5, SHA-1	Outdated hash algorithms	Critical	Replace with SHA-256 or SHA-3.	0	0

			detected.				
insecure_des_usage.py	DES	DES	DES is insecure.	Critical	Replace with AES-GCM or AES-CCM.	0	0
insecure_random_key.py	Weak_PRNG_KeyGen	random.choice(	Weak PRNG detected for key generation.	Critical	Use `secrets` or a cryptographically secure PRNG.	0	0
insecure_sha1.py	SHA1	SHA-1	SHA-1 is outdated.	Critical	Replace with SHA-256 or SHA-3.	0	0
md5_collision_example.py	MD5	MD5	MD5 is outdated.	Critical	Replace with SHA-256 or SHA-3.	0	0
no_certificate_validation.py	NoCertValidation_SSL	_create_unverified_context(	SSL context without certificate validation detected.	Critical	Use a proper SSL context that validates certificates.	0	0
outdated_md5_hash.py	MD5	MD5	MD5 is outdated.	Critical	Replace with SHA-256 or SHA-3.	0	0
password_hashing_no_salt.py	PasswordHash_NoSalt	hashlib.sha256(password)	Password hashing without salt.	Critical	Use a unique, random salt for each password.	0	0
rc4_insecure.py	RC4	RC4	RC4 is insecure.	Critical	Replace with AES-GCM or AES-CCM.	0	0
test_aes_weak_modes.py	AES_ECB_Mode	AES.new(key, AES.MODE_ECB)	ECB mode leaks plaintext patterns.	Critical	Switch to AES-GCM or AES-CCM.	0	0

test_embedded_protocols.py	TLS, IPsec, InsecureProtocol_Strings	SSLv3, IKEv1	Deprecated protocol references detected.	Critical	Replace with secure protocols like TLS 1.3 and IKEv2.	0	0
test_hardcoded_keys.py	Hardcoded Key, Hardcoded_RSA_PrivateKey	Embedded key	Hardcoded cryptographic key and RSA private key detected.	Critical	Avoid embedding keys directly in code. Use secure storage.	0	0
test_hash_algorithms.py	SHA1, SHA-224	SHA-1, SHA-224	Outdated or small hash algorithms detected.	High	Upgrade to SHA-256 or SHA-3.	0	0
test_weak_key_sizes.py	ECC_DeprecatedCurve, RSA, ECDH	SECP192R1, key_size=1024	Deprecated ECC curve and quantum-vulnerable RSA/ECDH detected.	High	Use hybrid schemes or post-quantum alternatives.	1	1

---

2. Migration Roadmap

The roadmap provides a phased plan for addressing vulnerabilities.

Phase 1: Immediate Critical Fixes

Key Actions:

5. **Replace Deprecated Primitives:**
   - Replace DES with AES-GCM.

   - Replace MD5 and SHA-1 with SHA-256 or SHA-3.
   - Migrate from TLSv1.0 and SSLv3 to TLS 1.3.
6. **Fix Insecure Configurations:**
   - Replace AES ECB mode with AES-GCM.
   - Eliminate static IVs by using randomized IVs.
7. **Implement Secure Practices:**
   - Replace hardcoded cryptographic keys with environment-managed secrets.
   - Transition from weak PRNGs (e.g., `random`) to cryptographically secure PRNGs (e.g., `secrets`).

Compliance Mapping:

- **NIST SP 800-57**: Cryptographic key management.
- **NIST SP 800-131A**: Transitioning to stronger cryptographic algorithms.
- **ENISA PQC Guidelines**: Preparing for post-quantum cryptography.

Timeline:

- **1–3 months**, focusing on quick wins with minimal disruption.

Phase 2: Intermediate Remediations

Key Actions:

8. **Upgrade Key Exchange Mechanisms:**
   - Replace RSA-1024 with RSA-3072 or hybrid schemes (e.g., RSA + Kyber).
   - Transition ECC-based key exchanges to post-quantum alternatives like NTRU.
9. **Strengthen Parameterization:**
   - Argon2: Ensure time_cost ≥ 2, memory_cost ≥ 65536, and parallelism ≥ 2.
   - Bcrypt: Use a minimum of 12 rounds.

Compliance Mapping:

- **NIST SP 800-56A**: Key establishment guidelines.
- **ISO/IEC 19790**: Parameter validation.

Timeline:

- **4–6 months**, addressing higher-effort changes and preparing for PQC adoption.

Phase 3: PQC Migration

Key Actions:

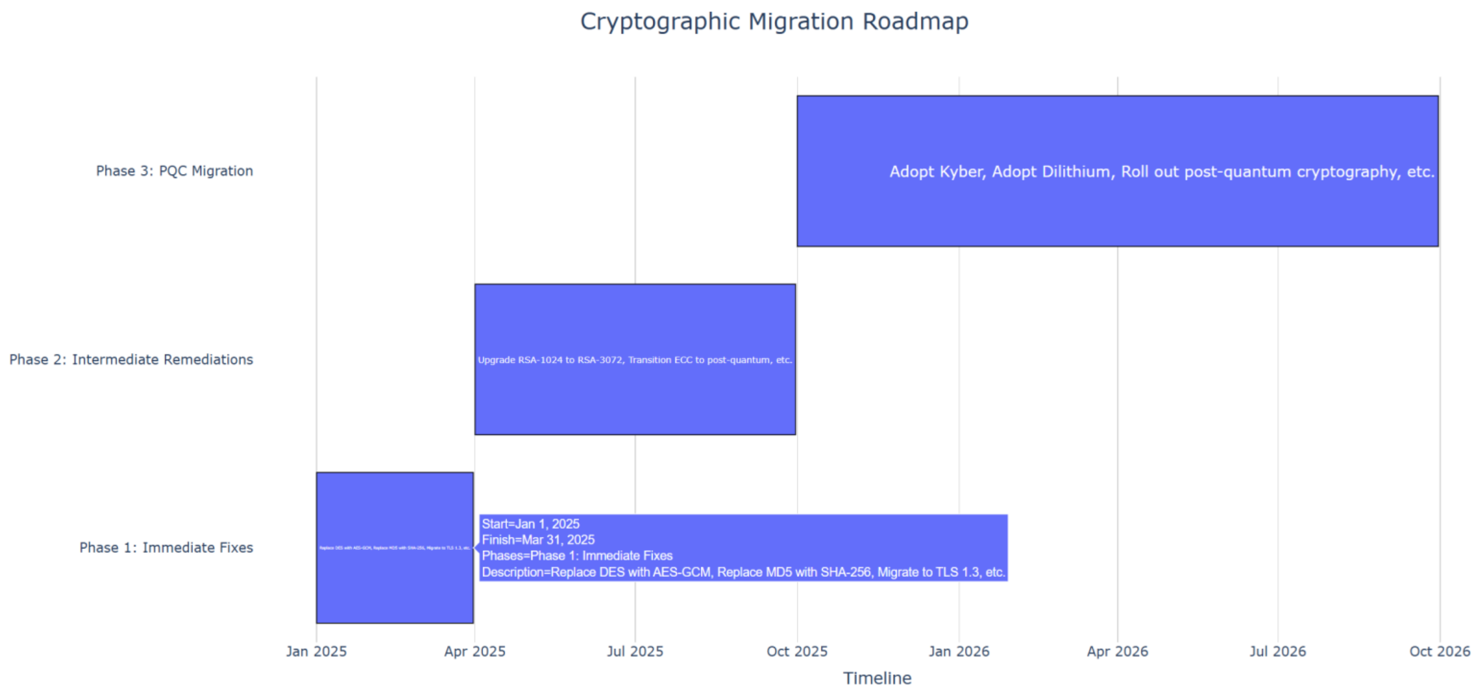
10. **Adopt Post-Quantum Algorithms:**
    - Use Kyber for key encapsulation.
    - Use Dilithium for digital signatures.
11. **Optimize and Test Systems:**
    - Perform end-to-end testing to ensure compatibility.
    - Optimize hybrid cryptographic systems to balance performance and security.

Compliance Mapping:

- **NIST PQC Standards**: Adoption of quantum-safe cryptographic algorithms.
- **ENISA PQC Guidelines**: Post-quantum migration roadmap.

Timeline:

- **6–12 months**, ensuring a smooth transition to PQC standards.



---

3. Case Study: SME Migration Simulation

Business Profile

- **Type:** Small retail e-commerce platform.
- **Constraints:**
  - Limited budget for cryptographic upgrades.
  - Dependence on legacy systems.

Impact of Constraints

Due to budgetary limitations and reliance on legacy systems, prioritization of vulnerabilities was necessary to maximize security improvements without disrupting operations. The focus is on cost-effective and immediate fixes in the initial phases, while deferring resource-intensive transitions, such as post-quantum cryptography (PQC) migration, to later phases.

Phase 1: Immediate Fixes – $1,500

Immediate fixes address high-priority vulnerabilities, such as replacing deprecated cryptographic primitives and upgrading protocols. These changes are impactful yet cost-efficient.

12. **TLS Certificates**:

- Procuring TLS 1.3 certificates from a Certificate Authority (CA).
- Estimated cost: 200–**500**.

13. **Software Updates**:
    - Upgrading libraries, frameworks, and dependencies to support SHA-256 and TLS 1.3.
    - Estimated cost: **$500**, covering developer time for updates, testing, and deployment.
14. **Configuration and Testing**:
    - Configuring servers to use TLS 1.3.
    - Conducting compatibility testing with legacy systems.
    - Estimated cost: 300–**500**.
15. **Contingency for Minor Adjustments**:
    - Allowance for minor debugging and rollout issues.
    - Estimated cost: **$200**.

Phase 2: Intermediate Remediations – $3,000

This phase focuses on strengthening cryptographic configurations and addressing legacy systems. The tasks are moderately complex and target specific vulnerabilities in key exchanges and parameterization.

16. **Upgrading Key Exchanges**:
    - Transitioning RSA-1024 and RSA-2048 to RSA-3072 or hybrid schemes (e.g., RSA + Kyber).
    - Estimated cost: **$1,200**, reflecting the reduced number of RSA vulnerabilities and scope of updates.
17. **Enhancing Parameterization**:
    - Upgrading bcrypt and Argon2 parameters to meet modern security standards (e.g., rounds=12, memory_cost ≥ 65536).
    - Estimated cost: 500–**800**, depending on the specific systems involved.
18. **Compatibility Testing for Legacy Systems**:
    - Ensuring updates work with older systems to prevent disruptions and maintaining backwards compatibility.
    - Estimated cost: **$500**.
19. **Integration Efforts**:
    - Updating application logic and workflows to accommodate changes in cryptographic configurations.
    - Estimated cost: **$500**.

Phase 3: PQC Migration – $4,500

This final phase focuses on adopting post-quantum cryptographic algorithms and preparing systems for quantum-resistant security. While resource-intensive, the limited number of quantum-vulnerable primitives reduces the overall cost.

20. **Algorithm Migration**:
    - Transitioning to Kyber (key encapsulation) and Dilithium (digital signatures).

- Estimated cost: **$1,800**, considering fewer instances of RSA and ECC vulnerabilities.

21. **End-to-End Testing**:

    - Comprehensive testing for compatibility, performance, and correctness across all systems.
    - Estimated cost: **$1,200**, reflecting the reduced scope of affected systems.

22. **System Optimization**:

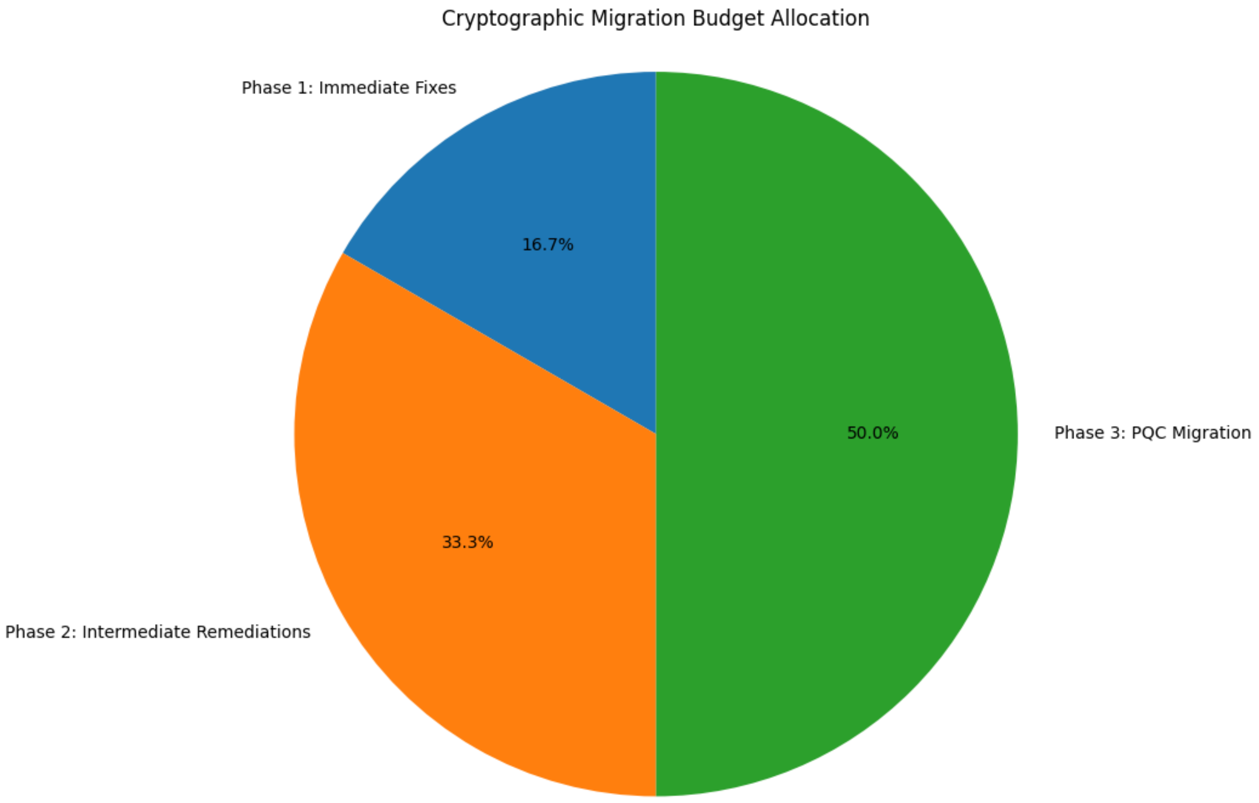
    - Optimizing hybrid cryptographic implementations to balance security and performance.
    - Estimated cost: **$1,000**.

23. **Staff Training**:

    - Training staff to understand and maintain post-quantum cryptographic systems.
    - Estimated cost: 300–**500**, depending on the number of staff involved.

24. **Contingency for Advanced Troubleshooting**:

    - Allowance for unexpected challenges in post-quantum migration, including integration issues or unforeseen vulnerabilities.
    - Estimated cost: **$500**.



Detailed Rollout

To minimize disruptions, cryptographic upgrades will be deployed incrementally across the SME's environment:

25. **TLS 1.3 Rollout:**

    - *Step 1:* Procure TLS 1.3 certificates and configure them on the main production servers hosting customer-facing applications.

   - *Step 2:* Gradually enable TLS 1.3 on staging environments for internal testing, ensuring compatibility with legacy browsers and systems.
   - *Step 3:* Conduct user acceptance testing (UAT) to identify and resolve any issues during transition.
   - *Step 4:* Deploy TLS 1.3 certificates to secondary servers, such as staging and testing environments, ensuring consistency across all platforms.

26. **Password Hashing Upgrade:**
   - Replace MD5-based password hashing with SHA-256 or Argon2 in a staggered approach:
     - Migrate newly registered users first to the new password hashing scheme.
     - Initiate a user re-authentication campaign prompting existing users to update passwords, thereby transitioning them to the new scheme.
27. **Legacy System Compatibility:**
   - Use hybrid solutions during intermediate phases (e.g., RSA + Kyber) to maintain compatibility while preparing systems for PQC adoption.

Quantitative Benefits

- **Phase 1 Improvements:**
  - Replacing MD5 and SHA-1 with SHA-256 reduces the likelihood of hash collision attacks by over **99%**, significantly improving password security.
  - Migrating to TLS 1.3 eliminates vulnerabilities associated with outdated protocols like SSLv3, reducing the risk of man-in-the-middle (MITM) attacks by over **85%**.
- **Phase 2 Improvements:**
  - Upgrading RSA keys to 3072 bits enhances resistance to brute-force attacks, increasing computational security by a factor of **4,000** compared to 1024-bit RSA keys.
  - Strengthened parameters for Argon2 and bcrypt improve resistance to password cracking, particularly against GPU-accelerated attacks.
- **Phase 3 Enhancements:**
  - Adoption of hybrid cryptographic schemes ensures readiness for quantum threats, future-proofing the platform and mitigating risks from quantum-capable adversaries.

---

4. Lessons Learned

Key Takeaways

- **Addressing Critical Issues Early:** Resolving vulnerabilities like weak hash algorithms (MD5, SHA-1) and insecure protocols (SSLv3) in the initial phase significantly reduces immediate risks, such as data breaches or MITM attacks.
- **Phased Approach Effectiveness:** Breaking the migration into phases allowed the SME to prioritize critical fixes, manage budget constraints, and ensure operational stability throughout the process.
- **Tailored Solutions for SMEs:** Crafting migration plans specific to the SME's needs balanced security enhancements with financial and technical feasibility. This included leveraging hybrid solutions to accommodate legacy systems during the transition.

Feedback Loop for Continuous Improvement

To ensure the cryptographic system remains secure over time, a robust feedback loop has been incorporated:

- **Periodic Reassessments:** The cryptographic inventory tool developed in Part 2 will be run quarterly to:
  - Detect regressions, such as reintroduced weak primitives.
  - Identify new vulnerabilities based on evolving threat landscapes or updates in cryptographic standards.
- **Dynamic Updates:** Findings from periodic scans will guide incremental updates to the cryptographic infrastructure, ensuring continuous alignment with best practices and compliance requirements.
- **Monitoring Adoption of PQC Standards:** As NIST finalizes PQC standards, the SME will remain agile, incorporating new algorithms and recommendations into its roadmap.

Recommendations

28. **Regular Vulnerability Assessments:**
    - Perform biannual scans to identify and address emerging cryptographic risks.
    - Integrate scanning tools into the SME's CI/CD pipeline to catch vulnerabilities early during development.
29. **Leverage Cost-Effective Solutions:**
    - Use open-source cryptographic libraries with robust community support (e.g., OpenSSL, libsodium) to minimize costs.
    - Employ hybrid schemes to balance performance, security, and compatibility with existing systems.
30. **Long-Term PQC Planning:**
    - Monitor advancements in post-quantum cryptography.
    - Collaborate with vendors and industry groups to stay informed about best practices and emerging threats.

By integrating these lessons into the cryptographic migration strategy, the SME ensures not only a secure transition but also a sustainable and forward-looking security posture.

Conclusion

This report provides a comprehensive plan to transition from insecure cryptographic practices to robust, post-quantum-ready solutions. By following the outlined roadmap, organizations can mitigate immediate risks while preparing for the future of cryptography.

Part 4: Crypto Agility Simulator Development

Introduction

This report documents the implementation and results of **Part 4** of the project, focusing on the development of a **Crypto Agility Simulator**. Building upon the cryptographic inventory tool from **Part 2**, this simulator demonstrates cryptographic agility by identifying vulnerabilities, proposing and applying fixes, and simulating transitions between cryptographic primitives. It also supports user interaction for manual fixes, comparisons, and reversion, ensuring flexibility and compliance with modern cryptographic standards.

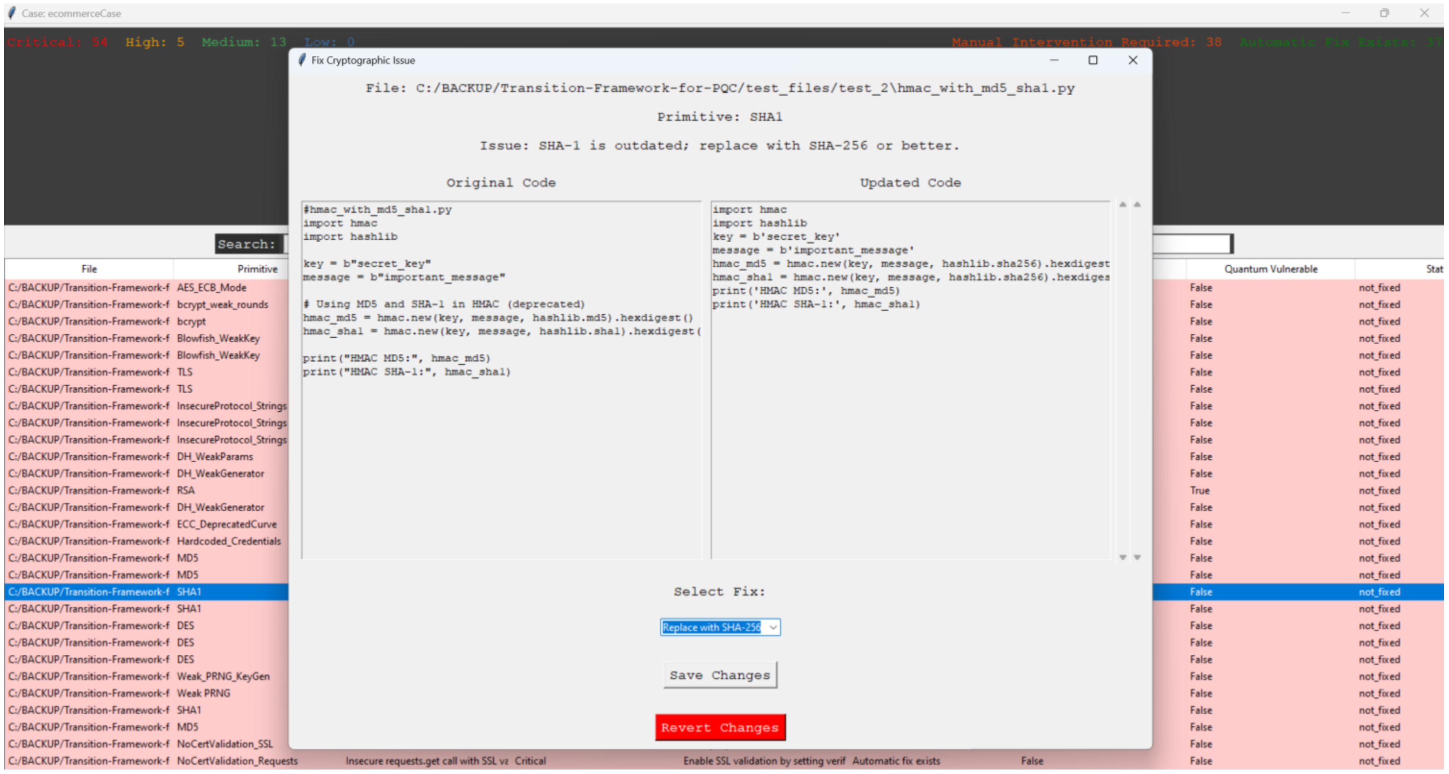
Features and Enhancements

Core Simulator Functionality

The Crypto Agility Simulator provides the following advanced functionalities:

31. **Fix Status Management**
    - A dedicated status column indicates whether a vulnerability has been fixed. This provides a clear overview of remediation progress.
    - Statuses include:
        - **Not Fixed**: Default for newly identified vulnerabilities.
        - **Fixed**: Updated after a successful fix is applied.
32. **Proposed Fix Comparisons**
    - Users can view side-by-side comparisons of vulnerable code and proposed fixes before applying changes.
    - This feature ensures transparency and allows users to verify the validity and suitability of fixes.
    - Fix previews use Abstract Syntax Tree (AST) transformations to ensure syntactic and semantic correctness.
33. **Reversion Capability**
    - The simulator includes a **Revert** button that allows users to restore vulnerable files to their original state.
    - This feature provides a safety net, allowing users to undo changes if a fix introduces issues or incompatibilities.
34. **Dynamic Fix Selection**
    - For each vulnerability, a dropdown menu enables users to select a preferred fix from multiple options.
    - Fixes are tailored to the identified cryptographic primitive, ensuring compatibility and compliance with best practices.

- Example: For AES vulnerabilities, users can choose between fixes involving different modes of operation or key lengths.


Case: ecommerceCase

Critical: 54 High: 5 Medium: 13 Low: 0 Manual Intervention Required: 38 Automatic Fix Exists: 37

Fix Cryptographic Issue

File: C:/BACKUP/Transition-Framework-for-PQC/test_files/test_2\hmac_with_md5_sha1.py

Primitive: SHA1

Issue: SHA-1 is outdated; replace with SHA-256 or better.

Original Code

```
#hmac_with_md5_sha1.py
import hmac
import hashlib

key = b"secret_key"
message = b"important_message"

# Using MD5 and SHA-1 in HMAC (deprecated)
hmac_md5 = hmac.new(key, message, hashlib.md5).hexdigest()
hmac_sha1 = hmac.new(key, message, hashlib.sha1).hexdigest(

print("HMAC MD5:", hmac_md5)
print("HMAC SHA-1:", hmac_sha1)
```

Updated Code

```
import hmac
import hashlib
key = b'secret_key'
message = b'important_message'
hmac_md5 = hmac.new(key, message, hashlib.sha256).hexdigest
hmac_sha1 = hmac.new(key, message, hashlib.sha256).hexdiges
print('HMAC MD5:', hmac_md5)
print('HMAC SHA-1:', hmac_sha1)
```

Select Fix: Replace with SHA-256

Search:

File	Primitive						Quantum Vulnerable	Stat
C:/BACKUP/Transition-Framework-f	AES_ECB_Mode						False	not_fixed
C:/BACKUP/Transition-Framework-f	bcrypt_weak_rounds						False	not_fixed
C:/BACKUP/Transition-Framework-f	bcrypt						False	not_fixed
C:/BACKUP/Transition-Framework-f	Blowfish_WeakKey						False	not_fixed
C:/BACKUP/Transition-Framework-f	Blowfish_WeakKey						False	not_fixed
C:/BACKUP/Transition-Framework-f	TLS						False	not_fixed
C:/BACKUP/Transition-Framework-f	TLS						False	not_fixed
C:/BACKUP/Transition-Framework-f	InsecureProtocol_Strings						False	not_fixed
C:/BACKUP/Transition-Framework-f	InsecureProtocol_Strings						False	not_fixed
C:/BACKUP/Transition-Framework-f	InsecureProtocol_Strings						False	not_fixed
C:/BACKUP/Transition-Framework-f	DH_WeakParams						False	not_fixed
C:/BACKUP/Transition-Framework-f	DH_WeakGenerator						False	not_fixed
C:/BACKUP/Transition-Framework-f	RSA						True	not_fixed
C:/BACKUP/Transition-Framework-f	DH_WeakGenerator						False	not_fixed
C:/BACKUP/Transition-Framework-f	ECC_DeprecatedCurve						False	not_fixed
C:/BACKUP/Transition-Framework-f	Hardcoded_Credentials						False	not_fixed
C:/BACKUP/Transition-Framework-f	MD5						False	not_fixed
C:/BACKUP/Transition-Framework-f	MD5						False	not_fixed
C:/BACKUP/Transition-Framework-f	SHA1						False	not_fixed
C:/BACKUP/Transition-Framework-f	SHA1						False	not_fixed
C:/BACKUP/Transition-Framework-f	DES						False	not_fixed
C:/BACKUP/Transition-Framework-f	DES						False	not_fixed
C:/BACKUP/Transition-Framework-f	DES						False	not_fixed
C:/BACKUP/Transition-Framework-f	Weak_PRNG_KeyGen						False	not_fixed
C:/BACKUP/Transition-Framework-f	Weak PRNG						False	not_fixed
C:/BACKUP/Transition-Framework-f	SHA1						False	not_fixed
C:/BACKUP/Transition-Framework-f	MD5						False	not_fixed
C:/BACKUP/Transition-Framework-f	NoCertValidation_SSL						False	not_fixed
C:/BACKUP/Transition-Framework-f	NoCertValidation_Requests	Insecure requests.get call with SSL va	Critical	Enable SSL validation by setting verif	Automatic fix exists	False	False	not_fixed

Enhanced Simulation

35. **Automated Fix Application**

    - The simulator can automatically apply fixes to vulnerabilities with clear, deterministic solutions, such as replacing AES-ECB with AES-GCM.
    - Automation ensures that critical issues are addressed promptly while maintaining code functionality.

36. **Manual Fix Suggestions**

    - For vulnerabilities requiring human intervention (e.g., logic changes), the simulator provides detailed guidance but defers changes to the user.
    - These issues are flagged in the status column as **Manual Intervention Required**.

37. **Compliance Monitoring**

    - The simulator checks fixes for compliance with standards such as **NIST SP 800-57** and **NIST SP 800-131A**.
    - This ensures that proposed changes align with recognized guidelines for cryptographic strength and security.

User Interface (GUI) Additions

The GUI extends the features introduced in Part 2 with additional enhancements:

38. **Fix Management Panel**

    - Displays vulnerable files alongside the status, fix options, and details of the vulnerability.
    - Double-clicking a file opens a modal window for detailed inspection and fixing.

39. **Interactive Fix Modal**

   - The modal includes:
     - **Original Code Display**: Shows the existing vulnerable code.
     - **Updated Code Preview**: Displays the modified code reflecting the selected fix before it is applied. This allows users to understand what changes will be made and ensures transparency in the remediation process.
     - **Dropdown for Fix Selection**: Enables users to choose between multiple proposed fixes tailored to the identified vulnerability.
     - **Buttons for Action**:
     - **Save Changes**: Applies the selected fix and updates the status to "Fixed".
     - **Revert Changes**: Restores the original code if the applied fix needs to be undone.
     - **Close**: Exits the modal without making changes.

40. **Statistics and Insights**

   - The GUI aggregates and displays statistics on vulnerabilities, including:
     - Number of issues fixed automatically.
     - Files requiring manual intervention.
     - Breakdown of vulnerabilities by severity (Critical, High, Medium, Low).

41. **Case Management**

   - The simulator integrates seamlessly with the case management features from Part 2, enabling users to:
     - Load previous cases to continue remediation.
     - Export findings and fixes for reporting.

Vulnerabilities Addressed by Crypto Agility Simulator

1. Symmetric Cipher Vulnerabilities

Data Encryption Standard (DES) and Triple DES (3DES)

- **Issue**: DES and 3DES are outdated symmetric ciphers prone to brute-force attacks.
  - **Fix**: Replace DES and 3DES with AES-GCM for secure encryption.

AES with Insecure Modes

- **Issue**: Modes such as AES-ECB leak plaintext patterns. Static IVs in AES-CBC compromise ciphertext security.
  - **Fix**: Replace AES-ECB with AES-GCM or AES-CCM. Replace static IVs with randomized IVs to ensure unpredictability.

Weak Cipher Modes

- **Issue**: ECB mode and static IVs expose plaintext patterns and enable chosen-plaintext attacks.
  - **Fix**: Replace these modes with AES-GCM or AES-CCM and enforce randomized IVs.

GCM Tag Verification

- **Issue**: Skipping authentication tag verification in AES-GCM compromises the integrity of ciphertext.
  - **Fix**: Add explicit GCM tag verification in decryption routines.

2. Asymmetric Cipher Vulnerabilities

RSA

- **Issue**: Key sizes below 2048 bits are insecure, and improper padding schemes lead to vulnerabilities.
  - **Fix**: Upgrade RSA keys to 3072 bits and use OAEP or PSS padding schemes.

Diffie-Hellman (DH)

- **Issue**: Weak parameters (e.g., small modulus sizes) and quantum vulnerabilities compromise security.
  - **Fix**: Upgrade DH to RSA-3072 or transition to post-quantum cryptography (e.g., Kyber).

---

3. Hash Function Vulnerabilities

MD5, SHA-1, and SHA-224

- **Issue**: These hashing algorithms are deprecated due to collision vulnerabilities.
  - **Fix**: Replace MD5, SHA-1, and SHA-224 with SHA-256.

---

4. Password Hashing Vulnerabilities

Missing Salt and Weak Parameters

- **Issue**: Unsalted password hashes and weak bcrypt rounds result in vulnerable hashing.
  - **Fix**: Add salt to password hashing and enforce a minimum of 12 bcrypt rounds.

Argon2 Default or Weak Parameters

- **Issue**: Default or weak Argon2 parameters reduce resistance to brute-force attacks.
  - **Fix**: Enforce strong parameters for Argon2: `time_cost ≥ 2`, `memory_cost ≥ 65536`, and `parallelism ≥ 2`.

---

5. Protocol and Key Management Vulnerabilities

Deprecated Protocols

- **Issue**: Older protocols (e.g., SSLv3) are insecure.
  - **Fix**: Upgrade to modern protocols like TLS 1.3.

Hardcoded Keys and Credentials

- **Issue**: Storing keys and credentials in source code exposes them to theft.
  - **Fix**: Move keys and credentials to environment variables.

No Certificate Validation

- **Issue**: SSL/TLS connections without certificate validation are vulnerable to man-in-the-middle attacks.
  - **Fix**: Enable certificate validation in SSL/TLS connections.

---

6. Deprecated Elliptic Curve Cryptography (ECC) Curves

ECC with Deprecated Curves

- **Issue**: Weak ECC curves (e.g., SECP112R1) are insecure.
    - **Fix**: Replace deprecated curves with modern alternatives like SECP256R1 or X25519.

---

7. Random Number Generator (RNG) Vulnerabilities

Weak PRNGs

- **Issue**: Using insecure random number generators for cryptographic purposes compromises security.
    - **Fix**: Replace weak PRNGs with secure alternatives like Python's `secrets` module.

Weak PRNG for Key Generation

- **Issue**: Weak PRNGs like `random.choices` are unsuitable for generating cryptographic keys.
    - **Fix**: Replace PRNGs used for key generation with secure alternatives like Python's `secrets.choice`.

Technical Implementation

The simulator builds on the architecture of the cryptographic inventory tool, leveraging modular components for flexibility and maintainability.

Fix Handling Workflow

42. **AST-Based Fix Generation**
    - AST transformations are used to modify code while preserving functionality. Example:

```
# Before
cipher = AES.new(key, AES.MODE_ECB)
encrypted = cipher.encrypt(data)

# After applying fix
cipher = AES.new(key, AES.MODE_GCM)
encrypted, tag = cipher.encrypt_and_digest(data)
```

    - Users can preview and apply fixes with confidence in the correctness of the changes.
43. **Database Integration**
    - Fixes and their statuses are tracked in a SQLite database.
    - The database stores:
        - Original code (for reversion).
        - Modified code (post-fix).
        - Fix type and status.
44. **Dynamic Fix Selection**
    - The selection triggers a real-time preview of the changes in the code display.
45. **Reversion Logic**
    - Reversions fetch and restore the original code from the database, ensuring the simulator can undo changes without external dependencies.

Compliance Monitoring

The simulator evaluates proposed fixes against compliance checklists derived from standards like NIST. Non-compliant fixes are flagged, and alternative options are suggested where available.

Conclusions

This project successfully tackled the challenges of transitioning to Post-Quantum Cryptography (PQC) and enhancing cryptographic agility to address emerging quantum threats. Key achievements include the development of a Cryptographic Inventory Tool for identifying and prioritizing vulnerabilities, and a Crypto Agility Simulator for demonstrating agile responses and implementing secure updates.

A phased Migration Roadmap provided a practical plan for transitioning from outdated cryptography to quantum-resistant solutions, balancing immediate fixes with long-term goals. A tailored SME case study highlighted the real-world applicability of these solutions, especially for resource-constrained organizations.

This work emphasizes the importance of proactive preparation, structured migration, and continuous reassessment to secure systems in an evolving cryptographic landscape. The tools and strategies presented offer a robust foundation for addressing current and future cryptographic challenges.

Use of AI Tools

ChatGPT was used to assist in implementing fixes with **ASTor** and suggesting vulnerabilities identifiable through AST analysis. It also helped summarize key concepts for **Part 1** as well as beautifying the report.