Cryptography in Process Workflow and Prompt Architecture for project-crypto

1. Introduction

This document provides a detailed explanation of the cryptographic mechanisms employed within the `project-crypto` web application, focusing on Hybrid Attribute-Based Encryption (ABE) and Digital Signatures. It also outlines a proposed prompt architecture for the system, detailing design principles, key components, and their interactions. The `project-crypto` project, available on GitHub at https://github.com/Hpgbao2204/project-crypto, aims to secure document management through advanced cryptographic techniques.

2. Cryptography in project-crypto's Workflow

The `project-crypto` application integrates two primary cryptographic functionalities: Hybrid ABE for data encryption and access control, and Digital Signatures for ensuring data integrity and authenticity. These mechanisms are crucial for maintaining the confidentiality and trustworthiness of documents within the system.

2.1. Hybrid Attribute-Based Encryption (ABE)

Hybrid ABE, as implemented in `project-crypto` (specifically in `src/encryption/hybrid_abe.py`), combines the strengths of symmetric-key encryption with the flexibility of attribute-based access control. This approach allows for efficient encryption of large data files while enabling fine-grained access control based on user attributes.

2.1.1. Core Principles and Implementation

The `HybridABE` class utilizes AES (Advanced Encryption Standard) in GCM (Galois/Counter Mode) for symmetric data encryption. This mode provides both confidentiality and authenticity. Key derivation is performed using PBKDF2 (Password-Based Key Derivation Function 2) with SHA256, ensuring that encryption keys are securely derived from passwords and salts.

Key components and their roles:

- `_derive_key(password, salt)`: This method securely derives a cryptographic key from a given password and a unique salt. The use of PBKDF2 with a high iteration count (100,000) makes brute-force attacks computationally expensive, enhancing key security.
- `_encrypt_data(data, key)` and `_decrypt_data(encrypted_data, key)`: These methods handle the symmetric encryption and decryption of data using AES-GCM. AES-GCM is chosen for its efficiency and its ability to provide authenticated encryption, meaning it can detect if the ciphertext has been tampered with.
- `setup()`: Initializes the global parameters for the encryption scheme, primarily by generating a master salt used in key derivation processes across the system.
- `authsetup(gp, name)`: Simulates the setup of an attribute authority, generating a public and secret key pair for that authority. In a real-world multi-authority ABE system, multiple such authorities would exist, each managing a set of attributes.
- `keygen(gp, sk, gid, attribute)`: Generates a user-specific secret key for a given attribute. This key is derived using the global parameters, the authority's secret key, a global user identifier (GID), and the specific attribute. The user's attribute key is then encrypted with the authority's key, ensuring that only the legitimate authority can decrypt and issue these keys.
- `encrypt(gp, pks, message, policy_str)`: This is the core encryption function. It encrypts the actual message (document data) using a randomly generated data key. Crucially, this data key is then re-encrypted multiple times, once for each attribute specified in the access policy. The re-encryption uses keys derived from the attributes themselves, ensuring that only users possessing the correct set of attributes can reconstruct the data key and subsequently decrypt the message.
- `decrypt(gp, sk, ct)`: The decryption process involves a user presenting their set of secret keys (corresponding to their attributes). The system then attempts to find a subset of these attributes that satisfies the access policy embedded within the ciphertext. If a satisfying set is found, the corresponding attribute key is used to decrypt the data key, which then decrypts the actual message.
- `_parse_policy(policy_str)` and `_find_satisfying_attributes(policy, user_attributes)`: These utility methods are responsible for parsing the human-readable access policy string into a structured format and for determining if a user's attributes satisfy the defined policy, respectively. The policy can include logical AND and OR operations, allowing for complex access rules.

2.1.2. Workflow Integration

In `project-crypto`, Hybrid ABE is integrated into the document management workflow as follows:

1. **Document Upload and Encryption**: When a user (typically a

Data Owner) uploads a document, they can choose to encrypt it using Hybrid ABE. During this process, the Data Owner defines an access policy (e.g., "Doctor@Hospital AND Department@Cardiology"). The system then encrypts the document and the data key under this policy. 2. **Access Control and Decryption**: When another user attempts to access an encrypted document, the system checks their attributes against the document's access policy. If the user possesses the necessary attributes to satisfy the policy, their secret keys are used to decrypt the data key, and subsequently, the document itself. If the attributes do not match the policy, decryption is denied.

2.2. Digital Signatures

Digital signatures, implemented in `src/encryption/digital_signature.py`, provide a robust mechanism for ensuring the authenticity, integrity, and non-repudiation of documents. They guarantee that a document has not been altered since it was signed and that the signature originates from a specific, verifiable source.

2.2.1. Core Principles and Implementation

The `DigitalSignature` class leverages RSA (Rivest–Shamir–Adleman) for generating key pairs and performing signing and verification operations. RSA is a widely used public-key cryptosystem suitable for digital signatures due to its strong security properties.

Key components and their roles:

- **`generate_key_pair(key_size=2048)`**: This method generates a unique RSA public-private key pair. The private key is kept secret by the signer and used to create the digital signature, while the public key is distributed widely and used by anyone to verify the signature.
- **`encrypt_private_key(private_key_pem, password)` and `decrypt_private_key(encrypted_key_base64, password)`**: These methods handle the encryption and decryption of the private key using AES in CBC (Cipher Block Chaining) mode, with key derivation via PBKDF2. This ensures that the private key, even if stored on a system, is protected by a password, adding an extra layer of security.

- **`sign_document(document_data, private_key_pem)`**: To sign a document, the system first computes a cryptographic hash of the document's content. This hash (a fixed-size string of characters) acts as a unique digital fingerprint of the document. The signer's private key is then used to encrypt this hash, producing the digital signature. The use of PSS (Probabilistic Signature Scheme) padding with SHA256 enhances the security of the signature.
- **`verify_signature(document_data, signature, public_key_pem)`**: To verify a signature, the verifier uses the signer's public key to decrypt the received digital signature, which yields the original hash of the document. Concurrently, the verifier independently computes the hash of the document. If the two hashes match, the signature is deemed valid, confirming the document's integrity and the signer's authenticity.

2.2.2. Workflow Integration

Digital signatures are integrated into the `project-crypto` workflow to provide verifiable proof of origin and integrity:

1. **Document Signing**: After a document is uploaded (and potentially encrypted), a user can choose to sign it. The system uses the user's private key (protected by their password) to generate a digital signature for the document. This signature is then stored alongside the document.
2. **Signature Verification**: Any user with access to the document and the signer's public key can verify the document's signature. This process confirms that the document has not been tampered with since it was signed and that it indeed originated from the claimed signer. The `test_scenarios.md` file explicitly outlines test cases for both valid and invalid signature verification, highlighting the importance of this feature.

2.3. Interplay of Hybrid ABE and Digital Signatures

The `project-crypto` application allows for both encryption and signing of documents, offering flexibility in how security is applied. The `test_scenarios.md` file details scenarios where documents are first encrypted and then signed, or vice-versa. This demonstrates a layered security approach:

- **Encrypt then Sign**: The document is first encrypted to ensure confidentiality (only authorized users can decrypt it). Then, the encrypted document (or its hash) is signed. This ensures that even if the encrypted document is intercepted, its authenticity can still be verified, and any tampering would be detected.
- **Sign then Encrypt**: The document is first signed to establish its authenticity and integrity. Then, the signed document is encrypted. This ensures that the original,

signed content is protected by encryption, and only authorized users can access it. The signature remains valid after decryption.

This dual approach provides comprehensive security, addressing both confidentiality and integrity/authenticity requirements within the document management system.

3. Prompt Architecture for project-crypto

Given the nature of `project-crypto` as a web application with complex cryptographic operations, a well-defined prompt architecture is crucial for user interaction, system feedback, and error handling. This architecture focuses on guiding users through cryptographic workflows, providing clear explanations, and ensuring a smooth user experience.

3.1. Design Principles

The prompt architecture for `project-crypto` should adhere to the following design principles:

- **Clarity and Simplicity**: Prompts should be easy to understand, avoiding overly technical jargon where possible. Complex cryptographic concepts should be explained in an accessible manner.
- **Guidance and Feedback**: Users should be guided through each step of cryptographic operations (encryption, decryption, signing, verification). The system should provide clear feedback on the success or failure of operations, along with actionable advice for errors.
- **Security Awareness**: Prompts should subtly reinforce security best practices, such as strong password usage for private key protection and understanding access policies.
- **Consistency**: The language, tone, and format of prompts should be consistent across the entire application to build user familiarity and trust.
- **Contextual Relevance**: Prompts should be relevant to the current user action and system state, providing information precisely when and where it is needed.

3.2. Key Components of Prompt Architecture

3.2.1. User Onboarding and Key Management Prompts

- **Registration**: When a new user registers, prompts should explain the importance of selecting appropriate roles and attributes, as these will directly impact their access rights in ABE. For example, "Chọn vai trò và thuộc tính của bạn. Các thuộc tính này sẽ xác định quyền truy cập của bạn vào các tài liệu được mã hóa."

- **Key Pair Generation (Digital Signature)**: When a user generates an RSA key pair, prompts should explain the purpose of the private and public keys and emphasize the importance of securing the private key with a strong password. For example, "Tạo cặp khóa chữ ký số của bạn. Khóa riêng tư của bạn sẽ được bảo vệ bằng mật khẩu. Hãy chọn một mật khẩu mạnh và ghi nhớ nó."
- **Private Key Encryption/Decryption**: When encrypting or decrypting a private key with a password, prompts should clearly indicate the action being performed and the need for the correct password. For example, "Nhập mật khẩu của bạn để mã hóa/giải mã khóa riêng tư."

3.2.2. Document Workflow Prompts

- **Document Upload**: When uploading a document, prompts should guide the user on available options, such as whether to encrypt or sign the document. For example, "Tải lên tài liệu của bạn. Bạn có muốn mã hóa hoặc ký tài liệu này không?"
- **Encryption Policy Definition (Hybrid ABE)**: This is a critical area for prompts. When a Data Owner defines an access policy, the system should provide a user-friendly interface for constructing the policy string (e.g., by selecting attributes and logical operators). Prompts should explain the syntax and implications of the policy. For example, "Xây dựng chính sách truy cập cho tài liệu này. Chỉ những người dùng có thuộc tính phù hợp với chính sách này mới có thể giải mã tài liệu."
- **Decryption Attempts**: When a user attempts to decrypt a document, prompts should inform them about the status of the decryption. If decryption fails due to insufficient attributes, a clear and informative error message should be displayed. For example, "Giải mã thất bại. Bạn không có đủ thuộc tính để truy cập tài liệu này. Vui lòng liên hệ với chủ sở hữu tài liệu để được cấp quyền."
- **Signing and Verification**: Prompts for signing should confirm the action and the use of the private key. For verification, prompts should clearly state whether the signature is valid or invalid and, if invalid, suggest potential reasons (e.g., document alteration). For example, "Chữ ký hợp lệ: Tài liệu này chưa bị thay đổi và được ký bởi [Tên người ký]." hoặc "Chữ ký không hợp lệ: Tài liệu này có thể đã bị thay đổi hoặc chữ ký không đúng."

3.2.3. Error Handling and Security Prompts

- **Generic Error Messages**: All error messages should be user-friendly, avoid technical jargon, and suggest actionable steps. Instead of a generic "Error occurred," provide specific messages like "Tải lên tài liệu thất bại. Vui lòng kiểm tra định dạng file và thử lại."
- **Security Warnings**: For actions with security implications (e.g., exporting a private key), prompts should include clear warnings about the risks involved. For example,

"Cảnh báo: Xuất khóa riêng tư có thể gây rủi ro bảo mật nếu khóa bị lộ. Bạn có chắc chắn muốn tiếp tục không?"
- **Password/Key Compromise**: In scenarios where a password or key might be compromised, prompts should guide the user on recovery or mitigation steps. For example, "Mật khẩu của bạn có vẻ đã bị lộ. Vui lòng đổi mật khẩu ngay lập tức và xem xét tạo lại cặp khóa."

3.3. Prompt Implementation Considerations

- **Internationalization (i18n)**: Given the user's request in Vietnamese, the prompt architecture should support easy internationalization to display messages in multiple languages.
- **Dynamic Content**: Prompts should be able to dynamically insert relevant information, such as usernames, document names, or specific attributes, to make messages more personalized and informative.
- **Logging**: All user interactions with prompts and system responses should be logged for auditing and debugging purposes, especially for cryptographic operations.
- **User Interface Integration**: Prompts should be seamlessly integrated into the web application's UI, appearing as modal dialogs, inline messages, or toast notifications, depending on their urgency and importance.

4. Conclusion

The `project-crypto` application effectively utilizes Hybrid ABE and Digital Signatures to provide a secure document management system. The Hybrid ABE ensures fine-grained access control based on attributes, while digital signatures guarantee the integrity and authenticity of documents. A thoughtfully designed prompt architecture, emphasizing clarity, guidance, and security awareness, is essential for maximizing the usability and trustworthiness of such a cryptographic system. By adhering to these principles, `project-crypto` can offer a robust and user-friendly solution for secure document handling.