Group 2
Chenxi Li, Haoteng Wang, Liang Zhao, Jasneet Kaur, Ron Manosuthi
COMP 533
April 28, 2021

Encryption - Part 2

Encryption is a process of encoding information that converts “plaintext” to “ciphertext”; it ensures confidentiality, which is vital to data security especially with the rapid development of information technologies.
The encryption concern encapsulates the following functionalities:

- Encryption/decryption with different algorithms.
- Notifications of job status.

Features

No	Feature Name	Feature Description	Realization Model Detail
1	Encryption (Root Feature)	This feature defines the basic classes for the concern.	This model contains the following classes: User, Key, and Content. The User class and the Content class are partial, and the input attribute of Content is the data to be encrypted or decrypted which is also partial. They need to be mapped at the time of reuse.
2	Job (Mandatory Feature)	This feature defines the different Job from which the user can select.	This model contains the Job Class which contains an attribute JobType as the Enumeration and also contains boolean attribute success which records whether the job is successful or not. This class is linked to User and Key.
2.1	EncryptionJob (OR feature)	This defines the Job Type as Encryption	This model defines the JobType as Encryption. There are four other associated realization models with this (AESEncrypt, AESBoth, RSAEncrypt, RSABoth). These realization models are explained in the below features.
2.2	DecryptionJob (OR feature)	This defines the Job Type as Decryption	This model defines the JobType as Decryption. There are four other associated realization models with this(AESDecrypt, AESBoth, RSADecrypt, RSABoth). These realization models are explained in the below features.
3	Notification (Optional	This feature allows user to receive	This model contains the Job class that has a buildString operation which is used

	feature)	notification whether the encryption and decryption process is a success or not	to create the message based on the job type and the success status of the job. This class is linked to the Notification class with an association.
3.1	Email (OR feature)	This allows the user to receive notifications by email.	This model adds the email attribute to the User class and adds the EmailNotifier which contains a sendEmail operation to send a notification to the user. This also adds the emailEnabled attribute to the Notification class that enables the sendNotification operation to send notifications using EmailNotifier.
3.2	SMS (OR feature)	This allows the user to receive notifications by SMS.	This model adds the phoneNumber attribute to the User class and adds the SMSNotifier which contains the sendSMS operation to send a notification to the user. This also adds the smsEnabled attribute to the Notification class that enables the sendNotification operation to send notifications using SMSNotifier.
4	Algorithm (Mandatory Feature)	This feature represents the different algorithms used to encrypt and decrypt information. Users can select between symmetric and asymmetric algorithms. For the design model, we have worked on one symmetric (AES) and one asymmetric (RSA) algorithm. Both the algorithms can be selected together.	N/A
4.1	AES	This is a symmetric algorithm.	This contains three different realization model for modularization purposes - 4.1.1 AESEncrypt 4.1.2 AESDecrypt 4.1.3 AESBoth
4.1.1			**AESEncrypt** This model extends the EncryptionJob and contains the following

			implementation classes from *java.security* and *javax.crypto* which help with the encryption process. ● **KeyGenerator**: used to generate a SecretKey each time an encryption job is processed if the user did not provide a key to use. ● **SecretKey**: used for encrypting and decrypting the data. ● **IvParameterSpec:** used to generate an initialization vector, the IV is used to avoid repetition during the encryption process thus making the algorithm more secure. ● **SecureRandom:** used in the generation of IV which helps with the security of the algorithm. ● **Cipher**: provides the functionality of a cryptographic cipher for encryption and decryption. We used methods from this class to do the encryption and decryption. ● **Base64**: a binary-to-text encoding scheme that was used to translate the encrypted/decrypted byte array back to String. It also contains the following classes to implement the AES algorithm: ● The **AESKey** class is responsible for generating everything we need to use in the encryption process. The generateAESKey operation generates and returns a SecretKey, the generateIv operation generates and returns an IvParameterSpec, both are used in the encryption process. There is also a operation that converts a String to a SecretKey if the user provided a key to use for encryption. ● The **AES** class is associated with Content and Job. This class contains the operation that actually does the encryption - encryptAES - it checks if the AES algorithm is selected and if a key is provided, then it calls the methods in *javax.crypto.Cipher* to do encryption and returns an encrypted String, it also sets the success status of the job.

4.1.2			**AESDecrypt** See AESEncrypt for a description of the implementation classes. In this model, two classes are similar to the ones in AESDecrypt: ● The **AESKey** class does not have the generation operation for SecretKey because a SecretKey should be provided by the user for decryption. There is also an operation that converts the user-provided secret key that is in String format to a SecretKey. ● The **AES** class is associated with Content and Job. This class has the decryptAES operation to decrypt data, it checks if the AES algorithm is selected and decrypts the data using the methods in *javax.crypto.Cipher*, the success status of the job is also set.
4.1.3			**AESBoth** This realization model extends the AESEncrypt and the AESDecrypt in it.
4.2	RSA	This is an asymmetric algorithm	This Contains four different realization model for modularization purposes - 4.2.1 KeyStorage 4.2.2 RSAEncrypt 4.2.3 RSADecrypt 4.2.4 RSABoth
4.2.1			**KeyStorage** In this realization model, we used a HashMap to store the keyPair after generation. The PublicKey is used as the key for the HashMap to store the KeyPair, the key-value pair for the HashMap is PublicKey-KeyPair.
4.2.2			**RSAEncrypt** This model extends the EncryptionJob and contains the following implementation classes from *java.security* and *javax.crypto* which help the encryption process. ● **KeyPairGenerator**: used to generate a KeyPair each time an encryption job is processed if the user did not provide a PublicKey. The user must provide a

			PublicKey that has been saved in our key storage to use it for encryption. ● **KeyPair**: contains a PublicKey and a PrivateKey which are used for encrypting and decrypting the data. ● **PublicKey:** distributed among users and used for the encryption process. ● **PrivateKey:** stored and used for the decryption process. ● **Cipher:** provides the functionality of a cryptographic cipher for encryption and decryption. We used methods from this class to do the encryption and decryption. ● **Base64**: a binary-to-text encoding scheme that was used to translate the encrypted/decrypted byte array back to String. ● **KeyFactory** and **X509EncodedKeySpec** are used to convert a String to a PublicKey. It also contains the following classes to implement the RSA algorithm: ● The **RSAKey** class is responsible for generating everything we need to use in the encryption process. The generateRSAKey operation generates and returns a KeyPair. There is also an operation that converts the user-provided publiv key that is in String format to a PublicKey. ● The **RSA** class is associated with Content and Job. This class contains the operation that actually does the encryption - encryptRSA - it checks if the RSA algorithm is selected and if a key is provided, then it uses the methods in *javax.crypto.Cipher* to do encryption and returns a String, it also sets the success status of the job.
4.2.3			**RSADecrypt** See RSAEncrypt for a description of the implementation classes. In this model, two classes are similar to

			the ones in RSADecrypt: ● The **RSAKey** class does not have the generation operation for PublicKey in this model because a PublicKey should be provided by the user for decryption. The getPrivateKey operation first converts the input String to a PublicKey then use the PublicKey to fetch the PrivateKey from storage. ● The **RSA** class is associated with Content and Job. This class has the decryptRSA operation to decrypt data, it checks if the RSA algorithm is selected and decrypts the data using the methods in *javax.crypto.Cipher*, the success status of the job is also set.
4.2.4			**RSABoth** This realization model extends the RSAEncrypt and the RSADecrypt in it.

API Description

The user of the concern can select the job type between Encryption and Decryption. It is also possible to select the algorithm the user wants to use for the selected job. There is an additional attribute that can be used to enable or disable the algorithm. To reuse our concern, simply create the required mappings, and create an instance of an algorithm of choice, enable it by setting the boolean attribute then call the encryption operation.

Moreover, it provides the optional feature whether the user wants to get notified of the success and the failure of the job. The user of the concern can set the different values of the attributes. It can turn on and off email and SMS notifications. To use the Notification feature, create the required mappings, create a Notification instance and enable the notifier(s) of choice by setting the boolean attribute(s), then simply call the sendNotification operation.

Impact Model

There are four goals in our impact model:

- Decrease encryption time
- Decrease decryption time
- Increase Security
- Optimize memory usage
- Increase Informativeness —— SMS > Email
    - This goal model evaluates how fast the user can get the information. These values are based on the consideration of internet requirements to view the email, SMS will more likely to reach the user faster

International Journal of Computer Applications, April 2013 "A Study of Encryption Algorithms (RSA, DES, 3DES, and AES) for Information Security" was the main source for our algorithms' numeric rankings of memory usage, encryption/decryption time, and security level. According to this paper, we concluded the following, ranked from highest to lowest:

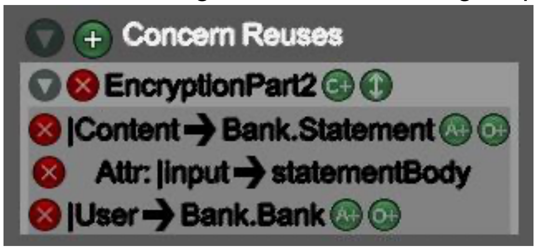
1. Decryption time —— RSA>>AES
2. Encryption time —— RSA>AES
3. Security level —— AES>RSA
4. Memory usage —— RSA>AES

Sample Application

For the sample application, we came up with a Bank domain in which we encrypt the statements of the Accounts. In the bank application, the statements are automatically encrypted by the bank before they are saved, the key is transmitted to the customer; when a customer wants to see his/her statements, he/she needs to provide a key to decrypt the statements. In the encryption scenario, the User would be the Bank; in the decryption scenario, the User would be the Customer. Our concern only supports a single user, thus, to demonstrate how to reuse our concern, two sample applications are provided:

- Bank_encryption

  In the encryption scenario, the Bank encrypts the statementBody automatically after they are created and before saving them. The following mappings are created.

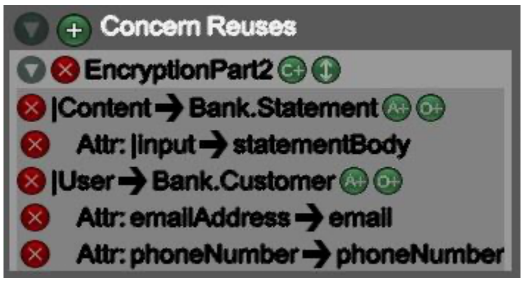


Concern Reuses
EncryptionPart2
|Content → Bank.Statement
Attr: |input → statementBody
|User → Bank.Bank

  In the constructor of the Statement class, we demonstrated the encryption process with the RSA algorithm by creating an instance of RSA and enabling it, then we simply encrypt the statementBody and save it. Notification is not meaningful in this scenario because the User is just another system.

- Bank_decryption

  In the decryption scenario, the User provides a key to the system if he/she wants to see the decrypted statements. The following mappings are created.



Concern Reuses
EncryptionPart2
|Content → Bank.Statement
Attr: |input → statementBody
|User → Bank.Customer
Attr: emailAddress → email
Attr: phoneNumber → phoneNumber

In the Account class, the displayStatements operation demonstrates the decryption process with the RSA algorithm by creating an RSA instance, enabling it and saving the user-provided key, then simply decrypt each statement saved in an ArrayList with a for-each loop. We also demonstrated the usage of the Notification feature in the displayStatements operation. Firstly, an instance of Notification is created, and we enable both email and SMS notifications, simply make a call to the sendNotification operation inside the for-each loop to send out notifications.

Follow the same logic to use the other algorithm.

Future Improvements

- Currently, we only support a single user in our concern, which can be inconvenient for many applications. In the future, support for multiple users should be considered.
- Our algorithm supports two algorithms, we have also considered adding support for more algorithms.
- Provide more information when a job fails to complete in the notifications.
- Further investigation is needed to evaluate the feasibility and meaningfulness of multiple encryption and combining encryption algorithms with other cryptosystems.