Encryption Mechanism
Software Requirement Specifications
changed to Dokumentation
Version 1.3.0

Table of Contents

User Requirement Document

1.Introduction

Security of data, while transferring over a network and stored on a shared location, is arguably the vital aspect to be think about.

In the software, BME for example, student exam data need to be transfer over a network and will be stored at ASW S3 or FTP. It is by law in many organizations or countries that data must be stored securely on shared location.

Encryption is the typical way of security to prevent falling data into wrong hand. The POC presented below uses intuitively a simple but a strong idea of encryption mechanism to secure the data.

Basically, there are two types of encryption are being used: Symmetric Encryption and Asymmetric Encryption (also called public-key encryption). Symmetric key uses same key to encrypt-decrypt the data whereas asymmetric key requires key pair (public-key and private-key). Though asymmetric encryption is bit complex and slower than symmetric encryption, it is more secure. That is the key factor to use asymmetric encryption in the POC.

Data encryption alone is not enough. Data in BME software also traverse from one place to another which brings another aspect to think about – authentication. Digital Signature is a method that demonstrate the authenticity of data/message.

Primary objective of this software is to implement proof of concept, using the core security aspect - Public Cryptography and Digital Signature - to store data securely and verify the authenticity of sender.

2.System work flow and terminology

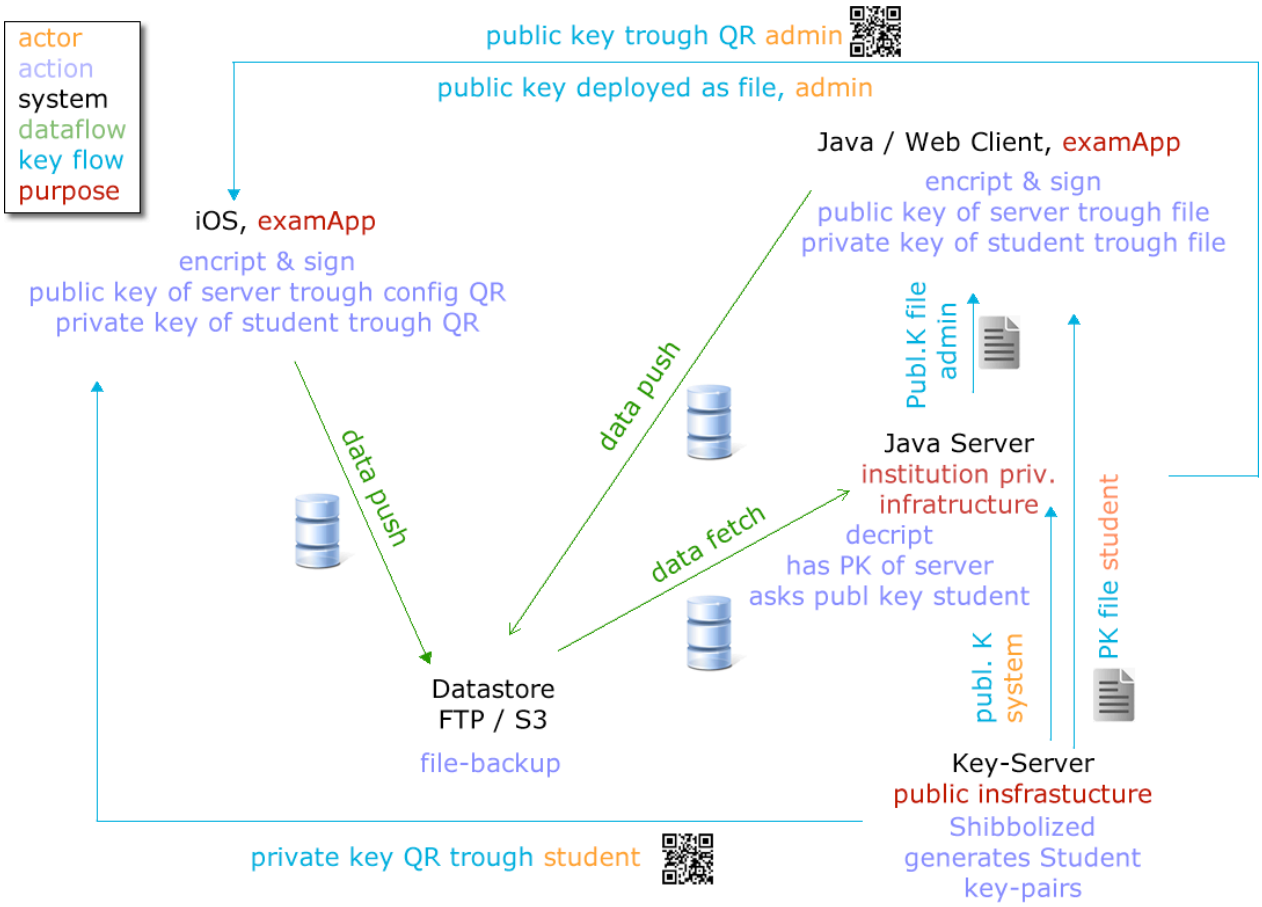
Overall, system together consists of two server (Key-Server and Decryption-Server) and three Client modules (Web-Client, Desktop-Client and iOS Client).

Both mechanisms - Public Cryptography and Digital Signature requires public-private key pair. One is used for data encryption – managed by Decryption-Server, the other one for signing the data – managed by Key-Server.

The keys can either be transferred as file or as QR code. Student private-key can also be transferred through web-request authenticated via Shibboleth. The encryption key is to be used by the admin whereas students use the signing key.

The Key-Server's key pair is used for signing and verifying the data. One of the Java Server is used to encrypt the exam data. Idea behind develop 3 different clients is to demonstrate the concept on different platform / environment.

Please find the below screenshot illustrating overall system work flow and integration of different modules:



Key-Server

There is a central server hosting a Shibboleth Authentication component that manages key pairs. Once student logs in with his credentials, he will see a frontend where he can generate a new key pair. Once a new key pair is generated, the old will be deactivated (but is still present). This mechanism should enable the possibility to create a new pair if the old one was lost on a USB stick or QR code.

The server has a two-way login mechanism: 1) Shibboleth and 2) User management. The user management surely can't use SSO.

The key consists of two files, the public and the private key. The public key may be downloaded from applications to ensure that the correct student signed the data. The private key will be kept secret and may be seen and downloaded only by the student after a successful login. The private key must be protected by a password. This feature prevents the key from being used in case the QR code or the file was lost.

The user can obtain the private key in three ways:

1. He logs in to an online application with his Shibboleth credentials. After a session is started, the key will be delivered. (Shibboleth is SSO, so in general it should work). The private key is secured by password, so once fetched, it is required to provide password while using in decryption. If he has forgotten the symmetric password, he may generate a new key from key-server.

2. He may store his key on his local storage device (USB, CD), secured by his password. There is also a file that contains information like email, study branch, name and first name.

3. He may print a QR code that contains his private key, secured by his password. This QR code may also contain his shibboleth email, study branch, name and first name.

It may happen that user has forgotten his login credentials. There should be a possibility to create a temporary key pair with the creation of a temporary user. This user may be merged to another account if the credentials are recollected again. With this approach a new key pair may be generated short before an exam. The only requirements for this are a notebook and a photo-printer.

Decryption-Server (Java-Server)

The exam client has two possibilities to get the public key from the decryption server. He may ask for a QR code (iOS and Android devices) or load the key from a file. This key is used for securing data transfer only. The key is preloaded to the exam client because the servlet / request URL is password protected. The handshake procedure might be redirected and mis-used.

Exam Apps (Java web/desktop and iOS client)

Generally, exam apps are configured for each student. As explained above, each exam client preload the public key for encryption (generated on Decryption-Server) and private key of respective student to digitally sign data (generated by student on Key-Server).

If keys are properly configured, user can upload encrypted and digitally signed student data on AWS S3 or FTP. Decryption-Server fetches this encrypted data and decrypt the files if successfully verifies the students digitally signed data.

3.System Requirements

To demonstrate the system workflow as described above, following are the essential modules/systems to be implemented and should be well integrated with each other.

1. Key-Server (Web Application to manage student key pair)
2. Decryption-Server/ Java Server (Web Application to manage admin key pair)
3. Web Client Application (Java) (examApp)
4. Desktop Client Application (Java) (examApp)
5. iPad Client Application (examApp)

3.1 Key-Server (Web Application to manage student key pair)

This application would act as a central server application for different client application. This application has its own web front-end to manage user management and key pair generation and download. This server application may probably be deployed on public infrastructure, so that it can be easily accessible to all. The application is only accessible through 443 and Shibboleth from the internet. This application will have following functionality:

3.1.1 Login Mechanism.

Log in

It would be implemented with two kind of login mechanisms: 1) Shibboleth and 2) Application's own User management.

When user accesses web front-end of server application, system will show the login page if user is yet not logged in either via shibboleth or via application's own log-in mechanism.

Login page contains the login component; one to login via application's own login mechanism and another to login via Shibboleth.

If user want to be authenticated via application's own log-in credentials, then he can enter user name, password and click on the “Login” button. Upon clicking on Login button, system will authenticate user with given credentials and redirect to main page if successfully authenticated.

If user clicks on “Login with shibboleth” component, then system will redirect user to their home organization to be authenticated.

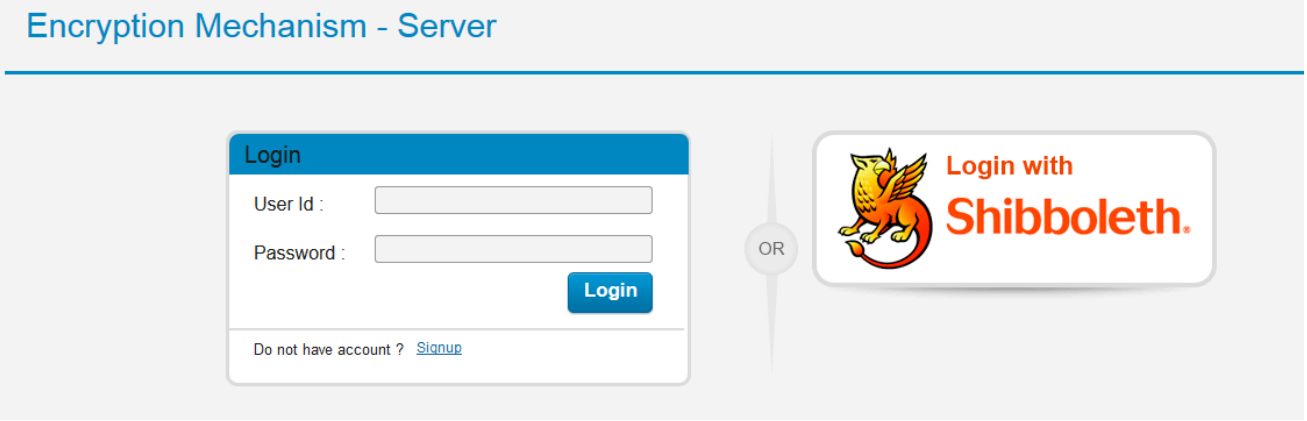


Fig.1 Log in page (Key-Server Application)

Sign up

If user could not able to sign in using shibboleth credentials for whatever reason, he can create his own account using “Signup” functionality. We provide a “Signup” link on the log in page to create a new fresh account.

User has to provide following information to create a new account: User Id, Password, Pre-name, Name. System maintains email-id as User Id. In addition, user may also specify Address, City, State etc. There is also a Password policy as below:

- Minimum length must be 8 digits,
- Should contain lower case, upper case, number/special character.
- Password must be stored as MD5 hash with encoded to Hex

If user logs in to the system first time via shibboleth, system will store the user information (User Id, Pre-name and Name) into User table.

System maintains unique record for both shibboleth user account and application's own user account. If user is already having shibboleth credentials, then his entry will be stored/maintained into the system whenever s/he logs-in for the first time. Now, as s/he is shibboleth user and their entry is already done in the system, s/he would not able to create a non shibboleth account using sign up functionality with same email address.

Suppose, user is shibboleth user, but haven't logged in to the system yet, then their record is not available in the system. Now, instead of login via shibboleth authentication, he can create an account via sign up functionality and can log in via non-Shibboleth authentication (application's own login component). However, whenever user login via shibboleth authentication and system finds his entry in the system as non-shibboleth user, then system will convert this account as shibboleth user. Hence, user will not able to log-in via non-shibboleth credentials afterwards. A warning is shown to accept the conversion.

User will able to create the new account via e-mail id other than shibboleth e-mail id. In future, user will able to merge

this account with his shibboleth account. To do so, user first has to login with his non-shibboleth credentials. Once login successful, there will be an option to merge the account with shibboleth account. Here, user has to provide his shibboleth email id. User will be allowed to merge account with his shibboleth account, only if entry with his shibboleth e-mail id is available in system (i.e. user is already log-in into the system via shibboleth id in the past). Once merged, old account (non-Shibboleth) will be deactivated and password will be erased. System will ask user to copy the generated key pairs of old account. If user selects “yes” then the key pair of old account will be copied to the account where old account is being merged. However, user can also generate new key pairs if required. In this case, existing key pairs will be deactivated/removed.

3.1.2 Detail Page.

Once logged in successfully, user will be redirected to detail page where he is shown user information and have access to these functionalities: 1) Generate key pair, 2) Download private key as file, 3) Download private key as QR Code.

Generate key pair

Upon clicking on “Generate key pair” button, system will ask to specify symmetric key/password. Private-key is protected with the password, so same password must be provided while using the private key. Public-key, however, is not protected with any password and may be used directly. Generated key pairs are persisted and can be downloaded upon request.

Download private key as file

Upon clicking on the “Download private key as file” button, system downloads private key as a file. Downloaded private key is secured with the password specified by user at time of key generation. So, it can not be directly used without knowing password and prevent the misuse.

In the exam application, user has to specify respective private key password before digitally signing the data.

There is also a check-box “Include user information”. If this box is selected then user information would also be included in the file.

Download private key as QR code

Upon clicking on the “Download private key as QR code” button, system downloads private key as QR Code in PDF file. Private key downloaded in the QR code is secured with password provided while generating key pair.

So whenever user uses in the client application to digitally sign the student data, he has to provide the password before use.

There is also a check-box “Include user information”. If this box is selected then user information would also be included in QR Code.

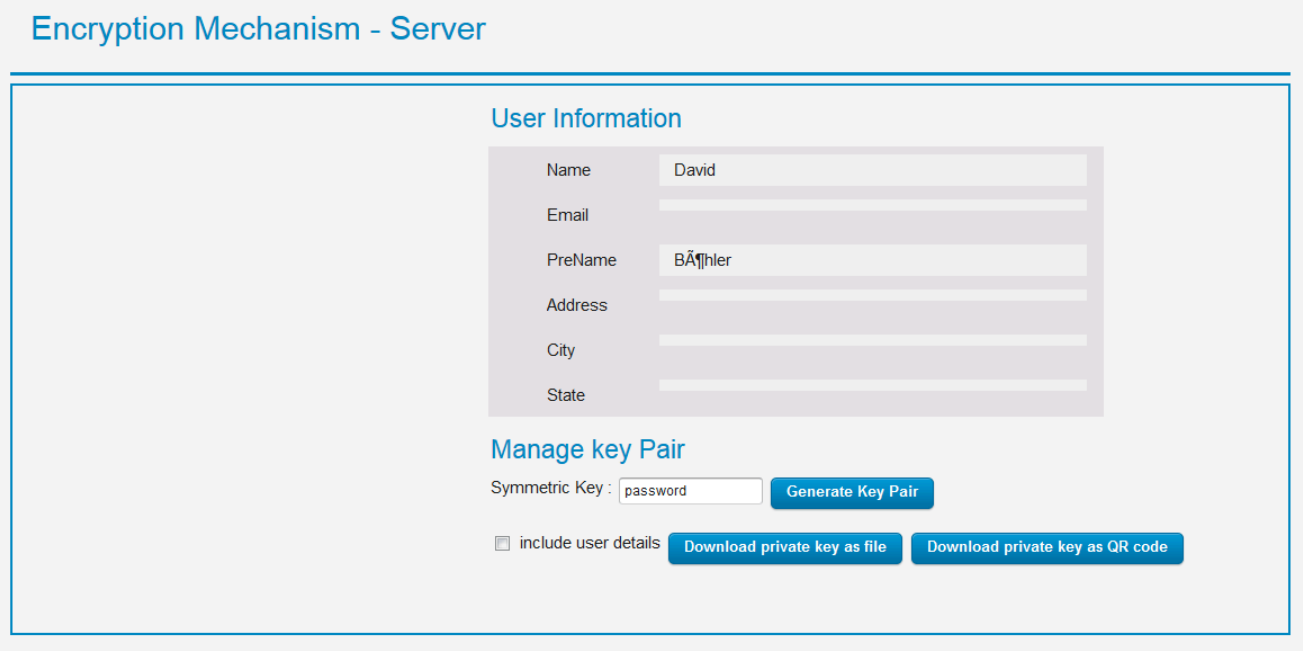


Fig. 2 Detail Page (Server Application)

Other services

For the web client using shibboleth authentication, there will be need to provide a servlet which can be requested by web client to get the private key. This servlet is only accessible if user is successfully logged in via shibboleth. This servlet will return private key as a response and will be used in web-client. This features is only available for web-client.

There would be another servlet to get public key. Decryption-Server can request this servlet to get the public key to verify the digitally signed data of particular student. System has to provide respective student's email id while requesting the servlet to fetch public key. No other login credentials needs to be checked while requesting this servlet.

3.2 Sever Application (Web Application) for admin

There would be separate admin server which would be similar to student server. This server is internal in every institution and accessible only trough trusted admin. The data for decryption are fetched from external file storage on demand.

Once login successfully, user can generate new public-private key pair. There would be only one key pair across the application. If user generates again, then old key pair will be deactivated.

Public key would be downloaded as QR code or as a file by trusted admin.

There will be two buttons: 1) to download public key as QR code image (in PDF) and 2) to download public key as file.

Private key generated on this server, however, will be kept secret in admin server and would not be sent outside.

Decrypt functionality

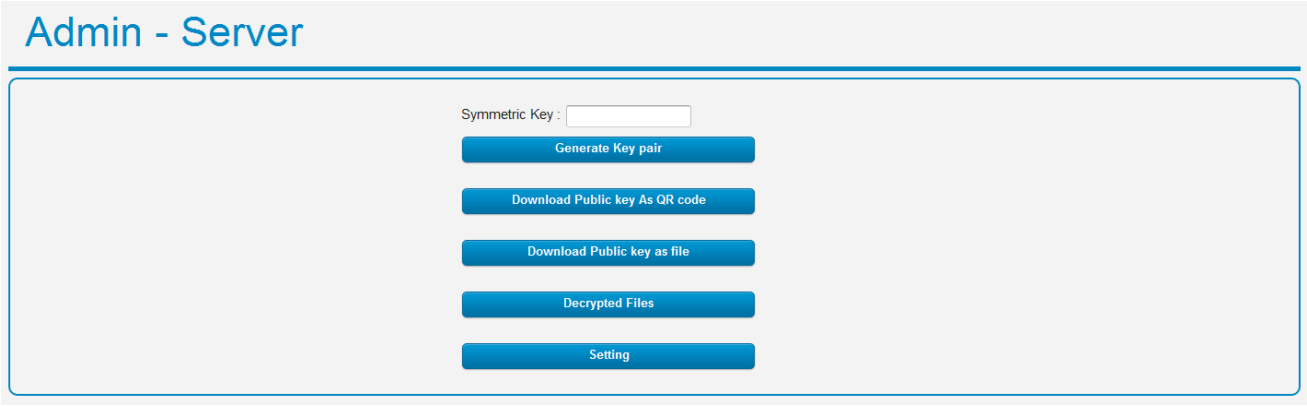
Admin server will also perform the decryption and verification. There will be one button labeled as “Decrypt file”. Upon click on “Decrypt file” button, system will fetch encrypted file from external storage system (e.g. FTP/S3) and decrypt them.

System first decrypts the file using own admin private key. Once decrypted, system will find the student email-id from

the file. Based on the student email-id, system will fetch public key of the student from student server and verify the digitally signed data.

Once all files are processed, system will show the list of the successfully processed file on web front-end.

System will also store all successfully processed file on the server temporarily, so user can download the file and verify the content.



3.3 Web Client

Web client would have Shibboleth authentication and should be deployed on Shibboleth enabled server. Whenever user accesses this application, he will be redirected for shibboleth authentication if no session is yet generated/available.

Once successfully authenticated via shibboleth, user will be redirected to client web application main page. There would be only one page.

An encryption panel on this page would have one file selection component and one text field to enter password along with “Encrypt” and “Setting” buttons.

Settings

User must configure the external storage attributes (S3/FTP), admin public-key and student private-key before performing encryption. Upon clicking on “Setting” button, a pop-up will be displayed where user can configure different attributes.

Admin public-key and Student private-key can be uploaded from QR Code or file downloaded from respective server application. Student private-key can also be fetched via web request. However, it does require shibboleth authentication.

Keys are stored in local storage and will be fetched whenever required.

Encryption

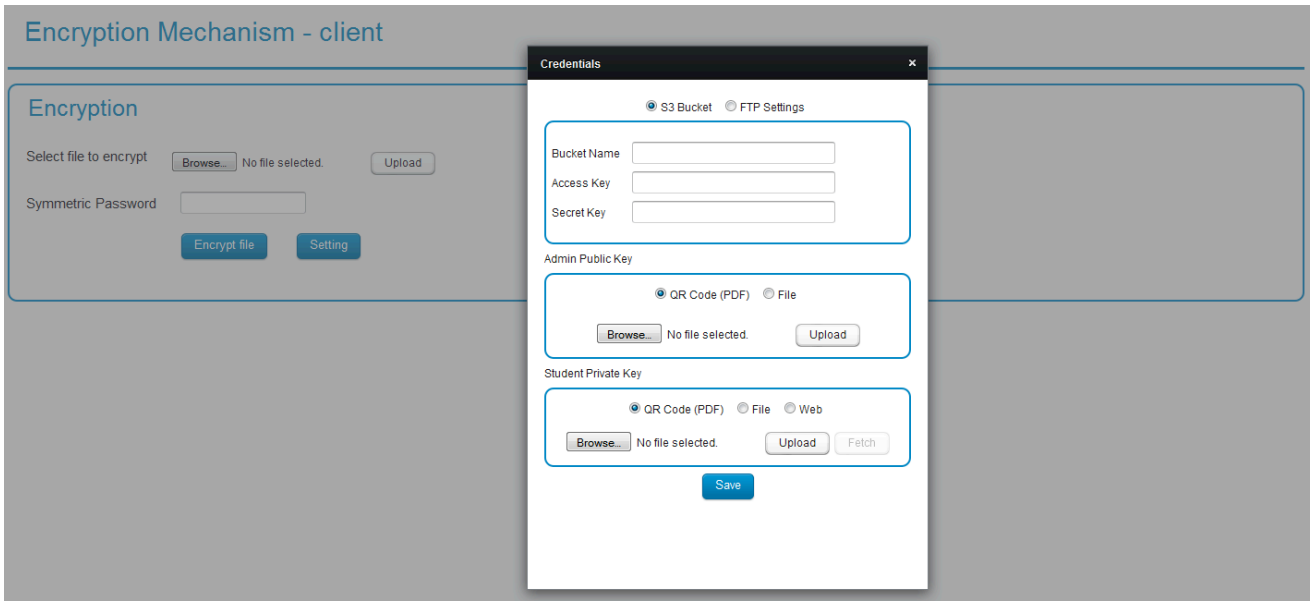
To encrypt any file, user has to first select the file from the local storage (we will use .txt file for testing) and must specify the symmetric key of respective student's private key.

Upon clicking on “Encrypt file” button, system performs following steps:

- System will digitally sign the data using student's private key and include in the plist file.
- System then encrypts the data using admin public-key and include in the plist file
- System also includes student's email id in the plist file, so, later in the decryption server it will be used to fetch the public-key from Key-Server.
- Once entire plist file will be generated, it will be uploaded on external storage (e.g. S3/FTP).

System will generate a plist file with following format:

```
<plist version="1.0">
<dict>
        <key>
                Email-ID
        </key>
        <data>
                Student's email id will be included here
        </data>
        <key>
                DigitalSignature
        </key>
        <data>
                Digitally signed selected file data hash (will be signed with student's private key) will be placed here.
                So it will be verified by client application on receiving.
        </data>
        <key>
                EnctyptedData
        </key>
        <data>
                Encrypted file data will be placed here. So it will be decrypted whenever requested.
        </data>
</dict>
</plist>
```



3.4 Desktop Client

For desktop client, there will NOT be any login mechanism. However, user has to provide email address to move on to the next screen. Once provided, next screen would have encryption panel having one file selection component and one text field to provide password along with “Encrypt” and “Setting” buttons.

Desktop client would have similar functionality as described above under web-client part. However, fetching student's private key using web request is not possible here.

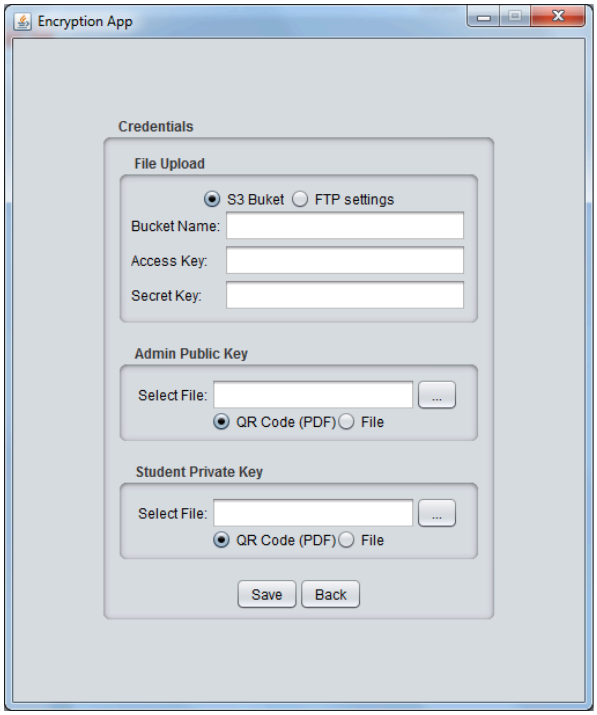


Fig. 4 Desktop Client Application (Setting screen)

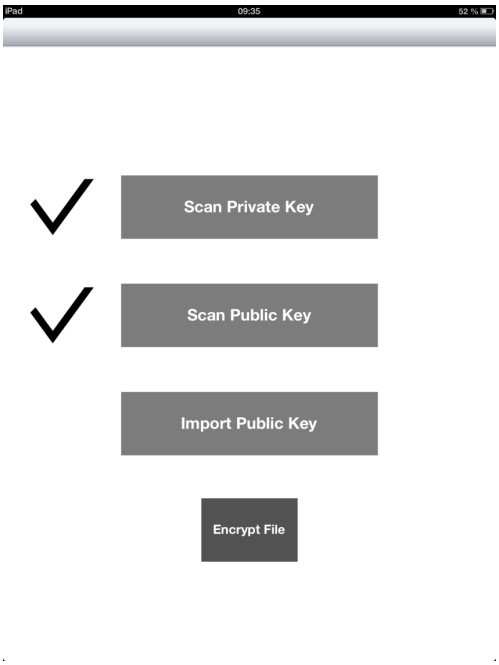
3.5 iPad Client

The purpose of iPad client application is to perform encryption of predefined file with public and private key and upload it to FTP location.

On application launch, Main screen will have “Scan Private Key”, “Scan Public Key” and “Import Public Key” buttons. The public key may already been configured or one can configure using Apple Configurator in App Datastore: Sho info: Current public Key

When user taps on “Scan Private Key” button application opens device camera to scan QR code image. QR code contains student Private key and student information.User can download QR code image from student web application.

When user taps on “Import Public Key” button, application finds admin public key file from application bundle to fetch admin public key. Admin public key file will be a text file with admin public key which will be preloaded inside the document folder of application bundle. Admin public key file can be loaded in application during device configuration using Apple Configurator or iTunes.



Once application gets student private key and admin public key (either using QR code or file) successfully, it shall move to next screen where user can actually perform encryption function using “Encrypt” button.

If scanning shall not find student private key or admin public key (which is mandatory), app shall show appropriate message like ‘QR code not valid’ or “File not found” and user will stays on same screen.

Due to length limitation the QR Code is ISO-8859-1 encoded.

Encryption Functionality:

On tap of “Encrypt” button, Application will show popup to enter Symmetric password. If user selects to continue, then user will be asked to enter Symmetric password. Here user has to enter the same password that s/he had used to

generate private key.

If Symmetric password will be valid, application shall be able to decrypt private key else shall show appropriate message to the user.

Once user enters correct symmetric password, the next screen will appear with list of text files. These files will be pre loaded in the demo application. Option is available for user to view any file content before encryption.

User has to select a file to encrypt. On successful encryption of selected file, system will show appropriate message e.g. 'Selected file has encrypted successfully'. Selected file will be encrypted using Admin public key and then digitally sign the data using student's private key. Application will include digital signature and encrypted data in plist file and will upload on FTP/S3. File format of plist file would be same as mentioned above in section 3.3.

Note:
There will be preloaded text file having FTP detail in document folder of application bundle. This file can be modified during iPad application configuration. iOS application will fetch FTP detail from this file while uploading plist file. If application does not found required FTP detail from file then it will show appropriate message to user.

For the scope of POC, encrypted text files will be stored within the application. Mobile application will not download files for encryption from server.

4.User Descriptions

Sr No	User Name/Type/Actor	Activities done by User/Actor in the system
1	Normal User (Mostly Student)	Transport the private key for signing
2	Admin	Transport the public key for encryption, triggers the decryption from admin server

5.Technology

- Apache tomcat web server
- Vaadin + Spring framwork
- Swing for desktop application
- Objective-C for iOS application
- MySQL database

6.Project Deliverables

- Key-Server, Decryption-Server and Web-Client - war files of each module and will be deployed on web server.
- Jar file for desktop client application.
- Executable file for iPad device
- Source code of the application – will be maintained on bitbucket.
- Installation Document