Software Design Document (SDD)

Rev: A0-02

4th May, 2017

"Secure Messaging"

From Concept to Customer

Proficient In Application Development Services
We help you deliver real solution for real business problem
Proven Expertise In Networking
Our Latest Technology Standards helps you achieve faster time to market
Excellence In Embedded Design
VVDN is a technology innovation company which stands for Voice Video Data Networking
Pioneers In Video Technology
We drive innovative approach using latest Video technology standards

Revision History:

Date	Rev No.	Description	By
25-03-2017	A0-01	Initial draft	VVDN
04-05-2017	A0-02	Second draft	VVDN

Table of Contents

Figures

1 Introduction

This document describes SW design details of the “Secure Message Application”.

This SDD is made for the reference of

- Product managers at VVDN & Embedded Downloads to confirm the SW design & Architecture before implementation
- Engineering Team at VVDN for implementation and validation of the Secure Message Application.
- System Integration and Verification team at VVDN/ Embedded Downloads.

1.1 Product Overview

This project is to design and develop a secure messaging application that will act as an interface between the sender and receiver on BitVault. Following are the key elements of the complete system proposed:

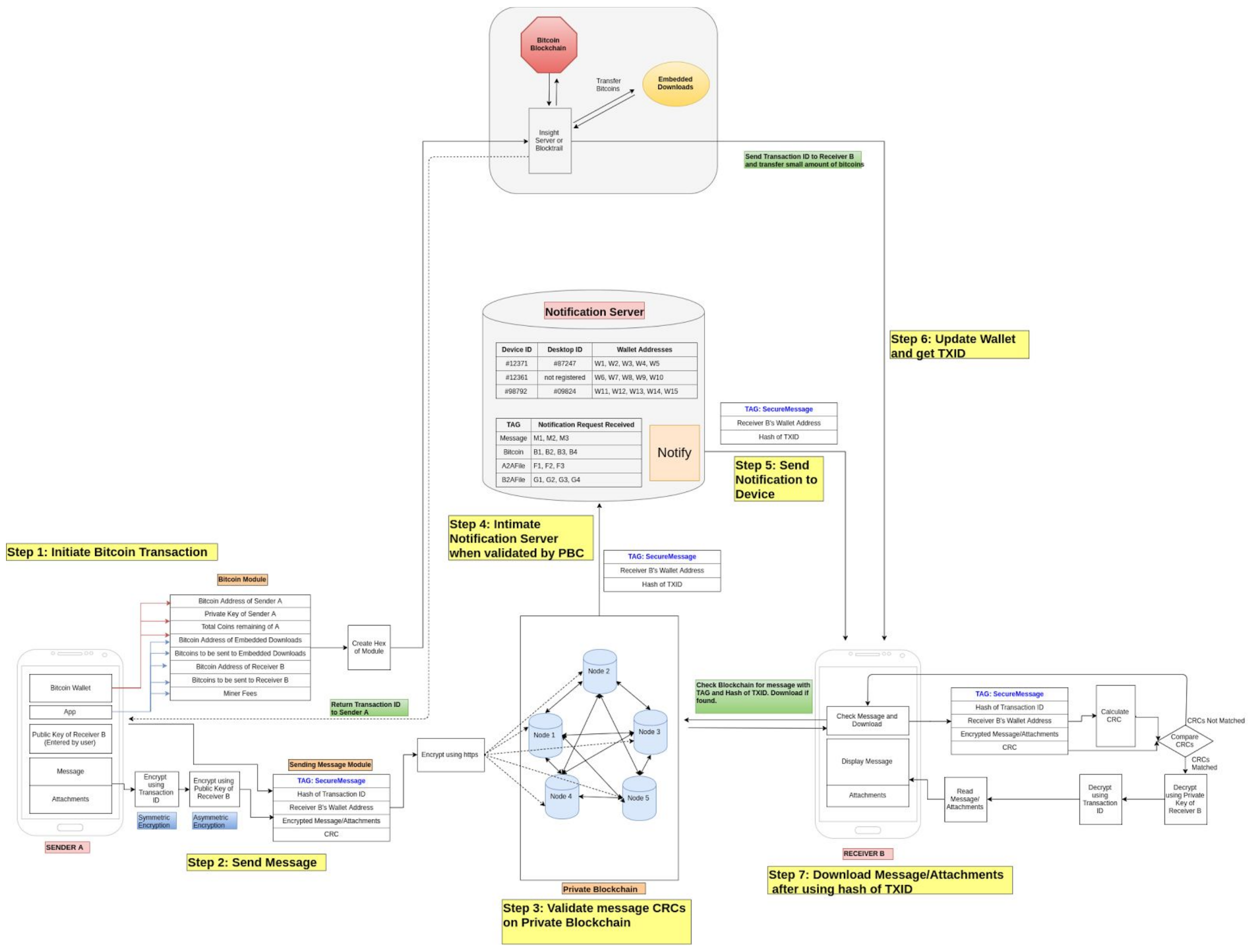
- Message content
- Bitcoin transaction for sending message/attachment to receiver
- Paying fees for sending message
- Sending message via private blockchain
- Check for new messages on private blockchain
- Download message
- Save or delete message
- Sending session key for BitVault Attachment desktop application.
- Sending files to desktop application from messaging app.

1.2 Scope of the Project

The product is a Secure Messaging application for BitVault. Secure Messaging Application is a native application of BitVault. User can send message to another user having the same application in BitVault. In order to send the message, user has to make a bitcoin transaction. User can check for new messages, download the message, save or delete the messages. User can also save receiver’s address to further send the message to the same user.

The user can also use the application for sending the session key for desktop to desktop attachments. Files can be send from the application to the user’s desktop as well.

2 Project Architecture


Bitcoin Blockchain
Transfer Bitcoins
Embedded Downloads
Insight Server or Blocktrail
Send Transaction ID to Receiver B and transfer small amount of bitcoins
Step 6: Update Wallet and get TXID
Notification Server
Notify
TAG: SecureMessage
Receiver B's Wallet Address
Hash of TXID
Step 5: Send Notification to Device
Step 4: Intimate Notification Server when validated by PBC
Step 1: Initiate Bitcoin Transaction
Bitcoin Module
Bitcoin Address of Sender A
Private Key of Sender A
Total Coins remaining of A
Bitcoin Address of Embedded Downloads
Bitcoins to be sent to Embedded Downloads
Bitcoin Address of Receiver B
Bitcoins to be sent to Receiver B
Miner Fees
Create Hex of Module
Return Transaction ID to Sender A
Bitcoin Wallet
App
Public Key of Receiver B (Entered by user)
Message
Attachments
SENDER A
Encrypt using Transaction ID
Symmetric Encryption
Encrypt using Public Key of Receiver B
Asymmetric Encryption
Sending Message Module
TAG: SecureMessage
Hash of Transaction ID
Receiver B's Wallet Address
Encrypted Message/Attachments
CRC
Step 2: Send Message
Encrypt using https
Node 1
Node 2
Node 3
Node 4
Node 5
Private Blockchain
Step 3: Validate message CRCs on Private Blockchain
Check Blockchain for message with TAG and Hash of TXID. Download if found.
Check Message and Download
Display Message
Attachments
RECEIVER B
TAG: SecureMessage
Hash of Transaction ID
Receiver B's Wallet Address
Encrypted Message/Attachments
CRC
Calculate CRC
Compare CRCs
CRCs Not Matched
CRCs Matched
Decrypt using Private Key of Receiver B
Decrypt using Transaction ID
Read Message/ Attachments
Step 7: Download Message/Attachments after using hash of TXID

Device ID	Desktop ID	Wallet Addresses
#12371	#87247	W1, W2, W3, W4, W5
#12361	not registered	W6, W7, W8, W9, W10
#98792	#09824	W11, W12, W13, W14, W15

TAG	Notification Request Received
Message	M1, M2, M3
Bitcoin	B1, B2, B3, B4
A2AFile	F1, F2, F3
B2AFile	G1, G2, G3, G4

Figure 1: Project Architecture

3 Software Layer Architecture

Software layer architecture contains the three layers that are described in following diagram.

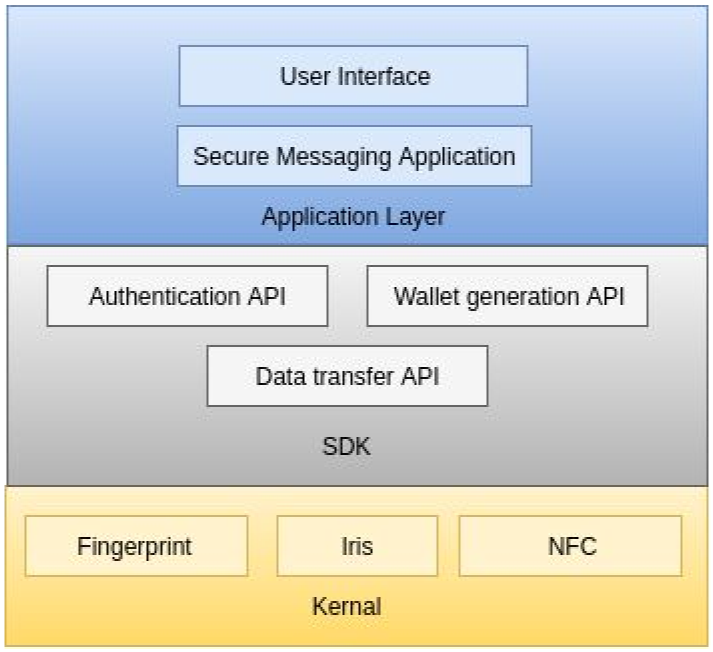


Figure 2: Software Architecture

There are three layers.

- Application Layer
- SDK Layer
- Kernel Layer

3.1 Application Layer

Application layer comprises of user interface, the Secure messaging application and all other application that will use the SDK.

3.2 SDK Layer

This layer will work as an interface between Application layer and Kernel. SDK will be having various sets of modules. Each module has different functionality, API and methods. SDK will perform the following functions:

1. UI for Iris, Fingerprint and NFC authentication.
2. Send/Receive transaction by using wallet generation API.

3. Data transfer using Data Transfer API.

3.3 Kernel Layer

This layer provides the access to fingerprint, iris, NFC authentication and secure storage.

1. Authenticate user using the provided Fingerprint, Iris or NFC card.
2. Key generation.
3. Decrypting the received encrypted data. First layer of decryption will be done by kernel.

3.4 Data Transfer API

The data transfer API interacts only with the private blockchain and is a custom implementation for secure sending and receiving of messages. It has the following APIs.

Response	Methods	Layer	Description
Success/Fail	SendData (Data, Receiver public key)	Application	Sends data using receiver public key.
Success/Fail	SendDataOnPrivateBlockChain (TAG, Encrypted data, transaction ID, Receiver address, CRC)	SDK	Sends message module to private block chain.
True/False	CheckDataOnBlockChain (TAG, Transaction ID)	SDK	Checks the inbox in the private blockchain for messages with TAG and hash of TXID.
Success/Fail	DownloadMessage ()	SDK	Downloads messages from private blockchain to the local message database.

4 Application Flow

The application flow has the following parts:

- Application Initialization
- Sending Message
- Receiving messages in Inbox
- Sent Messages
- BitVault Attachment

4.1 Application Initialization

Once the user starts the application, the user's iris, fingerprint and NFC card are authenticated. The application then loads all the available wallets. The user can choose the primary wallet to use for the secure messaging application. Wallet's public key and public address is displayed to the user with QR code of public key for sharing.

On continuing, the application transitions to the Home Screen with the options of creating a message, checking inbox, seeing sent messages, sending BitVault Attachment key and sending to to BitVault Attachment(desktop).

All new and pending messages are downloaded from Private Blockchain using the TAG and hash of received Transaction IDs and stored in a local database. The downloading process is explained in later section.

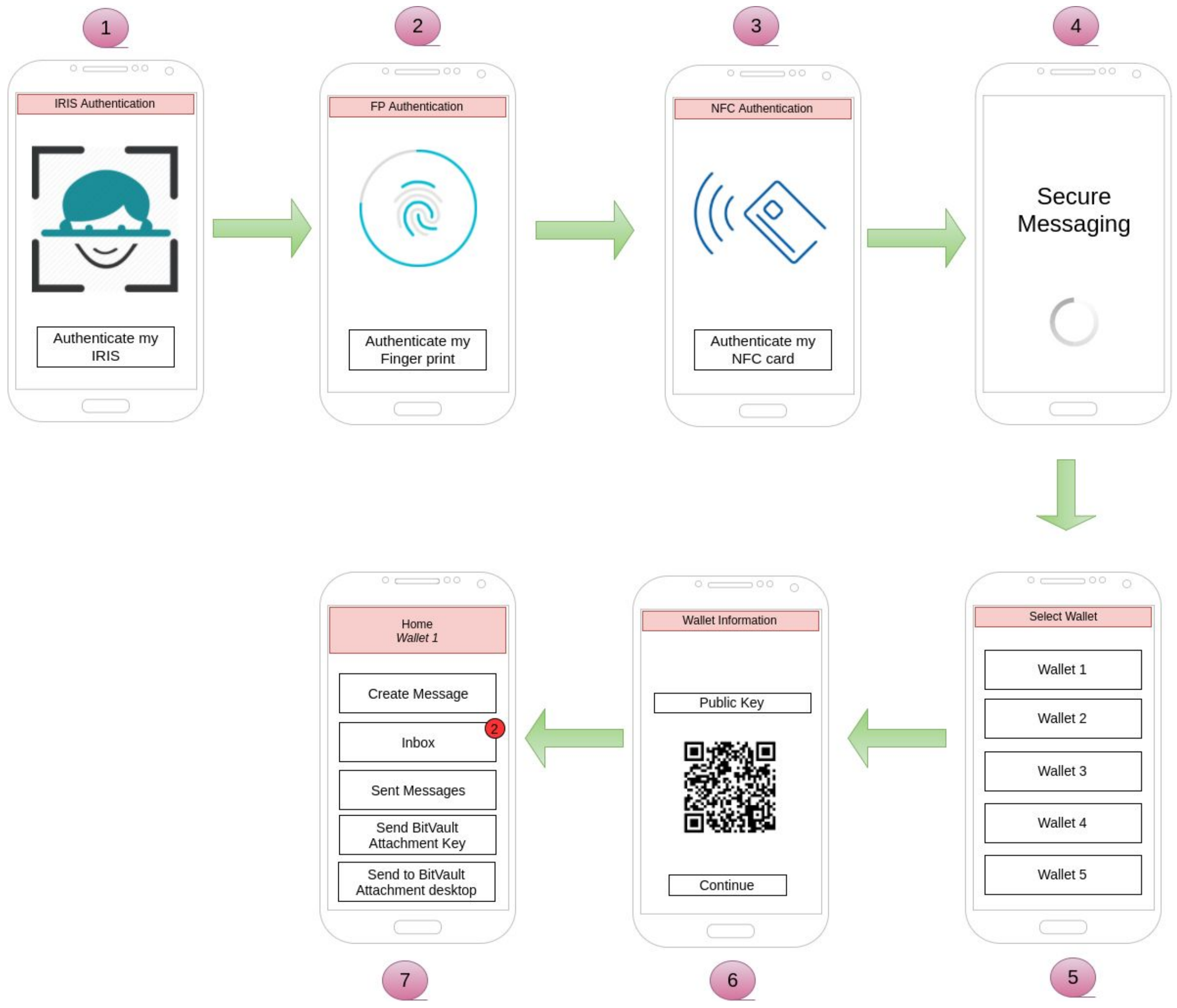


Figure 3: Application Initialization UI sequence

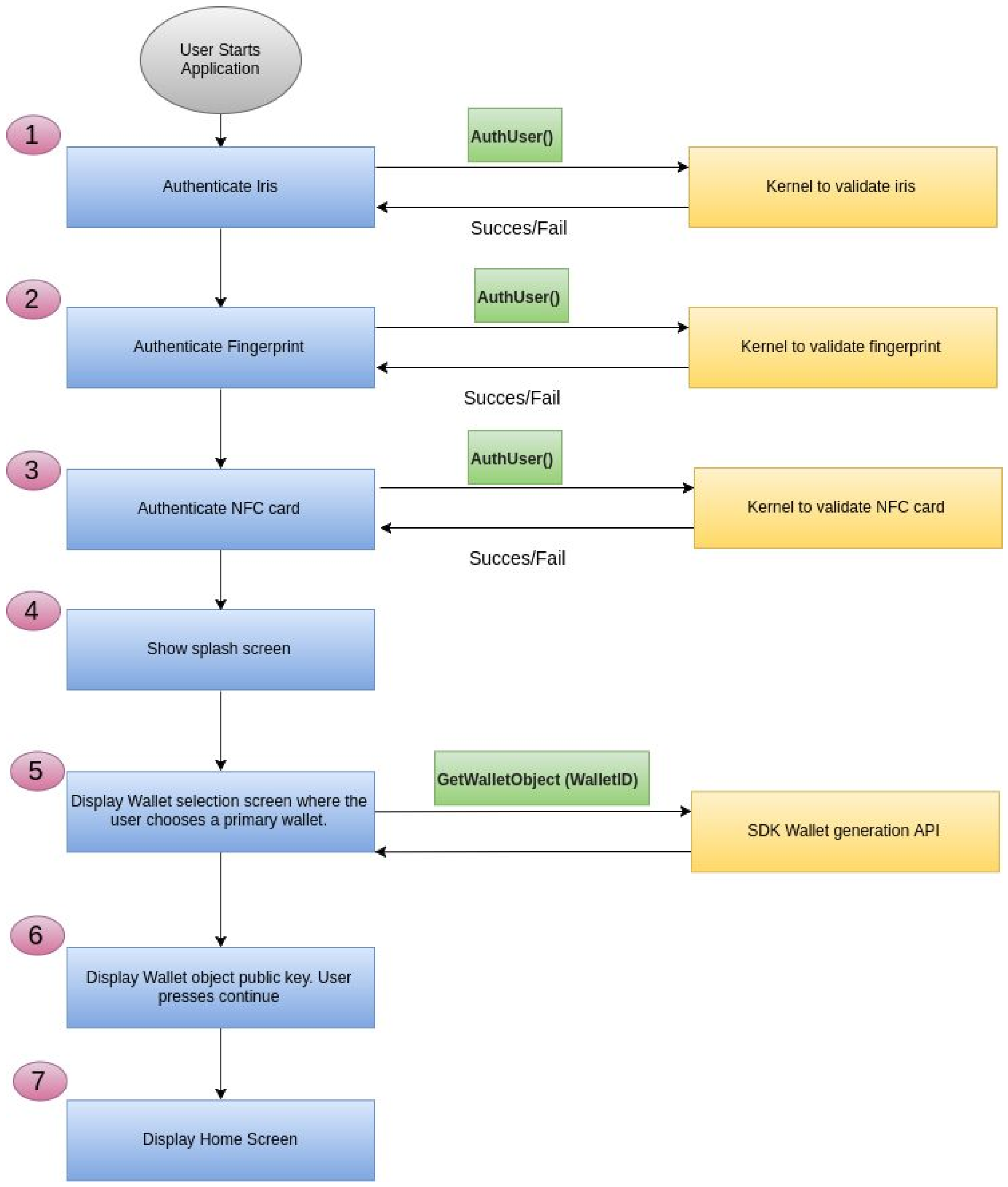

User Starts Application
1
Authenticate Iris
AuthUser()
Kernel to validate iris
Succes/Fail
2
Authenticate Fingerprint
AuthUser()
Kernel to validate fingerprint
Succes/Fail
3
Authenticate NFC card
AuthUser()
Kernel to validate NFC card
Succes/Fail
4
Show splash screen
5
Display Wallet selection screen where the user chooses a primary wallet.
GetWalletObject (WalletID)
SDK Wallet generation API
6
Display Wallet object public key. User presses continue
7
Display Home Screen

Figure 4: Application Initialization Flow

4.1.1 Application Authentication

Authenticating the application will be the first step towards unlocking and accessing the application. Like any other app lock application where user sets either a pattern or set of numbers for unlocking the application, here user will have to provide all the three choices that are: iris, fingerprint and NFC card authentication.

4.1.1.1 Iris Authentication

SDK will return the UI of the Iris. As soon as the user provides the iris, kernel will read the signature and will match with the stored iris signature. The output by the kernel will be success/failure. Signature will not be read/ stored in application layer or SDK layer.

Description	Authenticates the user's iris with the signature stored in Kernel Layer
Method	AuthUser ()
Input Parameters	None
Output Parameters	True/False Type: boolean

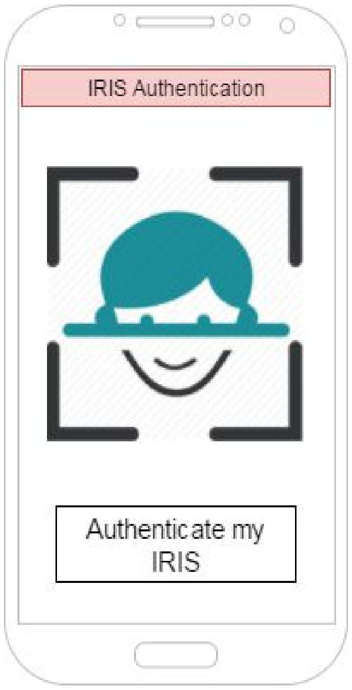


Figure 5: IRIS authentication screen

4.1.1.2 Fingerprint Authentication

This screen requires input of the user's Fingerprint. SDK layer will return UI of this screen. Kernel layer will use the fingerprint sensor to capture the fingerprint and will compare with the stored signature. This will return a success or failure.

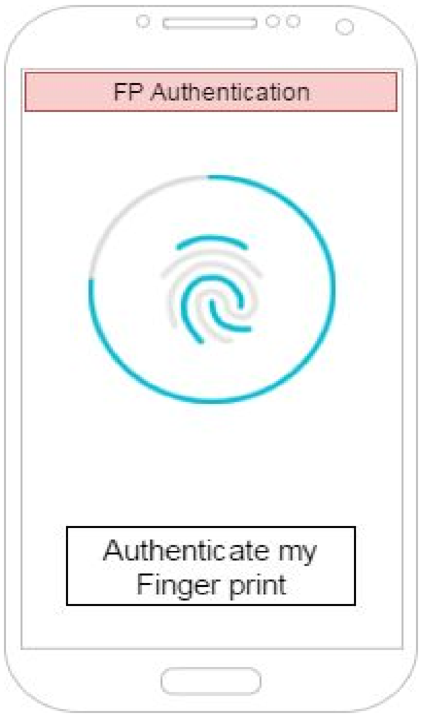


Figure 6: Fingerprint authentication screen

Description	Authenticates the user's fingerprint with the signature stored in Kernel
Method	AuthUser ()
Input Parameters	None
Output Parameters	True/False Type: boolean

4.1.1.3 Near Field Communication (NFC) Card

The user is prompted for a NFC card to complete the final security layer. As soon as the user authenticates using NFC card, the screen navigates to the Home Screen to access the application.

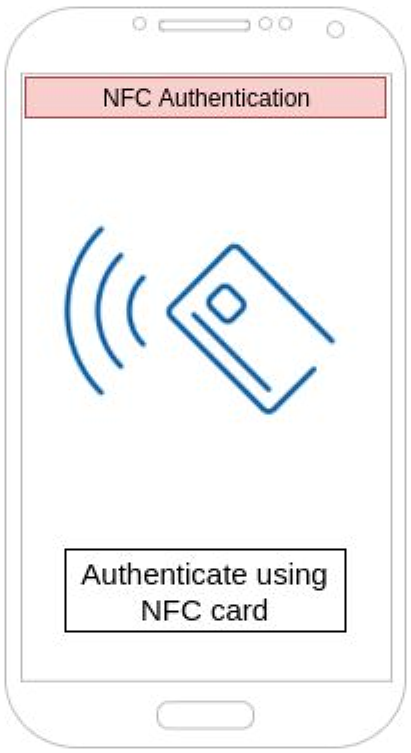


Figure 7: NFC card authentication screen

Description	Authenticates the NFC card
Method	AuthUser ()
Input Parameters	None
Output Parameters	True/False Type: boolean

4.1.2 Splash Screen

This is the first screen that appears after successful authentication and displays the name of the application. As soon as the splash screen appears, user is navigated to the Home Screen.

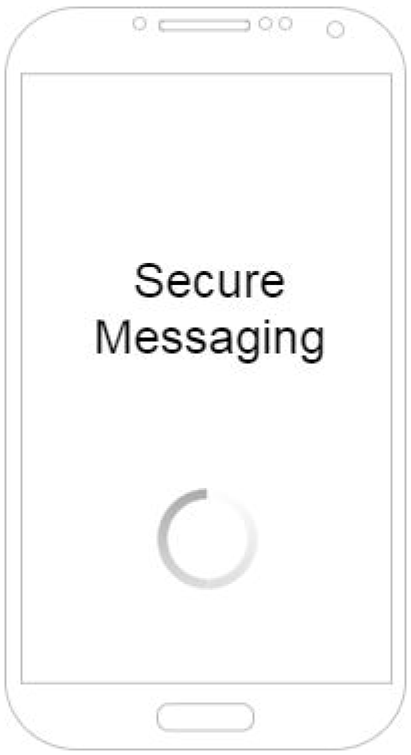


Figure 8: Secure Messaging Splash screen

4.1.3 Home Screen

Public key is shown on the screen to the user. Clicking proceed takes the user to the Home screen. It has the following buttons.

- Create Message – user can send new messages
- Inbox – user can check unread or saved messages
- Sent Messages – message that were sent successfully and saved.
- Send BitVault Attachment Key - send the key for desktop to desktop
- Send to BitVault Attachment desktop - send files from phone to desktop

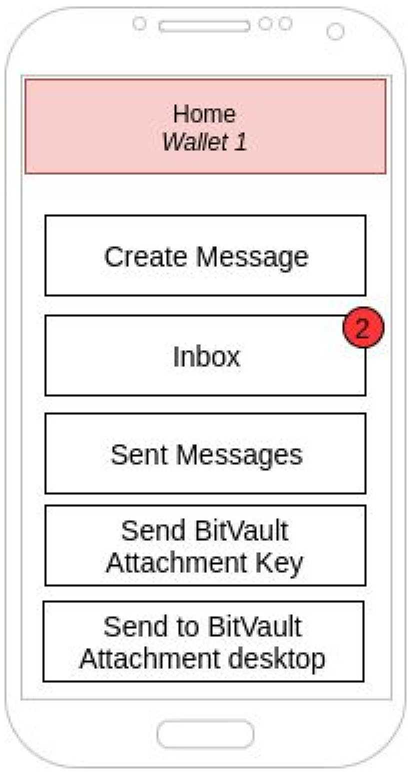

Home
Wallet 1
Create Message
Inbox
2
Sent Messages
Send BitVault Attachment Key
Send to BitVault Attachment desktop

Figure 9: Home screen

1.1.1 Wallet Selection

All the five wallets will be displayed on the screen from which user can choose any one as primary wallet to make a bitcoin transaction for sending the message.

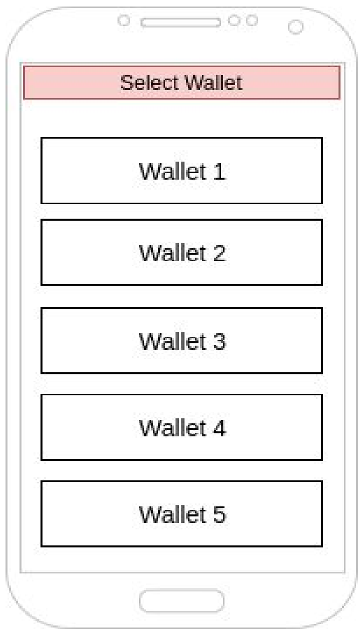


Figure 10: Wallet selection screen

All the wallets available to the user are displayed in this screen and they are updated using the API *GetWalletBalance ()*. The user then selects any one wallet that user wants to use for the messaging application using *GetTransactionHistory (Wallet Address)*. This downloads all the new transactions from the bitcoin blockchain for the wallet with the list of transaction IDs. Using these transaction IDs, the App tries to check the blockchain if new messages have been received using *CheckDataOnBlockChain (TAG, Transaction ID)*. If the response is a success, then the messages are downloaded using *DownloadMessage ()* and stored in their encrypted format in the local message database. This is done until all TxIDs corresponding to unread messages have been checked.

Description	Updates wallet balance by contacting the bitcoin blockchain
Method	GetWalletBalance ()
Input Parameters	Wallet Type: Object
Output Parameters	Bitcoins Type: long

Description	Gets all the new transactions for the wallet address from the bitcoin

	blockchain and returns the TxIDs
Method	GetTransactionHistory (Wallet Address)
Input Parameters	Wallet Address Type: String
Output Parameters	TxID Type: Object

Description	Checks the blockchain for new messages with the TAG and hash of TxID
Method	CheckDataOnBlockChain (TAG, Transaction ID)
Input Parameters	TxID Type: String
Output Parameters	True/False Type: boolean

Description	Downloads the messages from the private blockchain
Method	DownloadMessage ()
Output Parameters	True/False Type: boolean

1.2 Send Message

User can type a message and attach documents, images, videos and other files. The public key of the receiver can be typed, copied or scanned by QR code and entered in the recipient field.

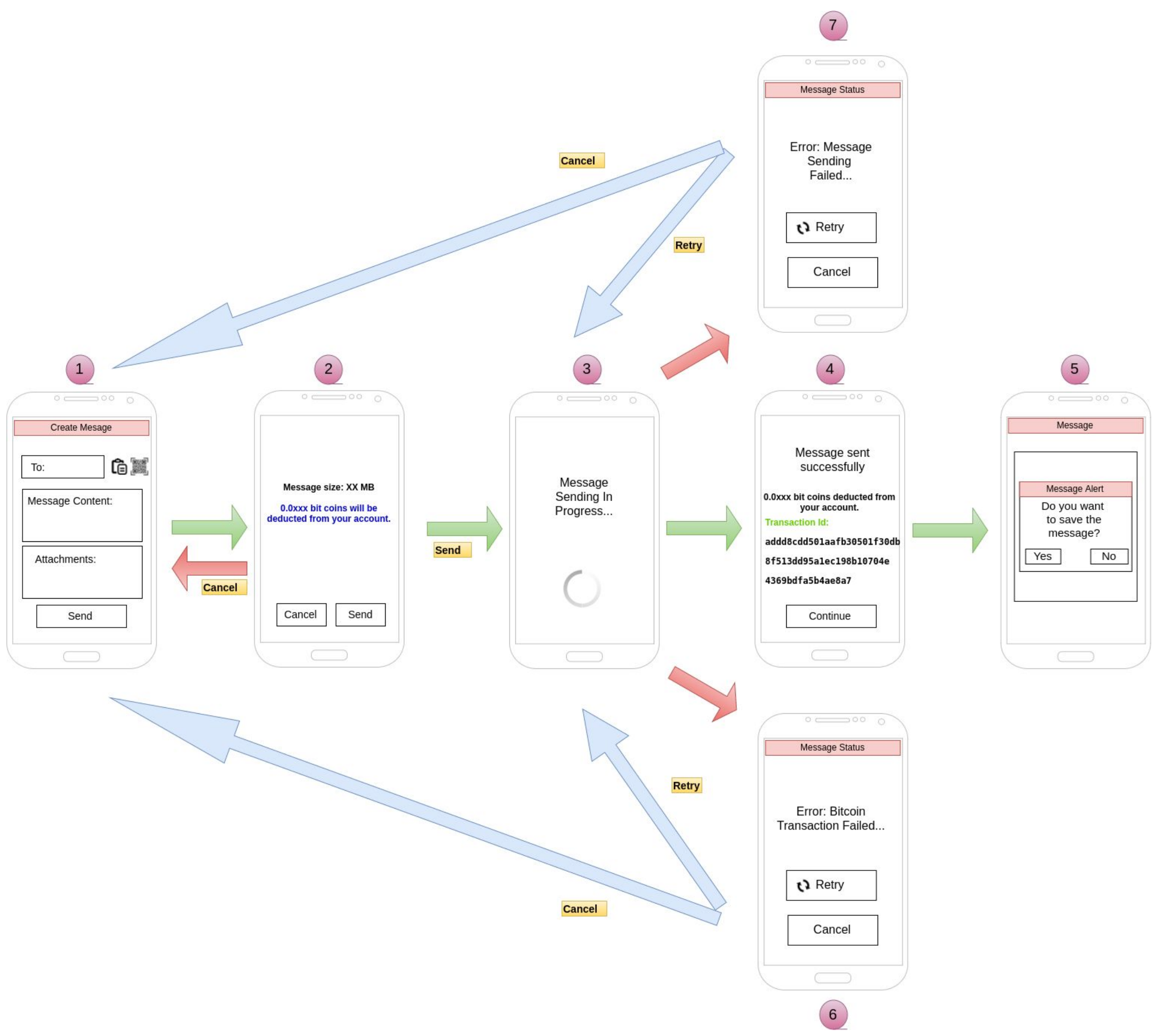
Bitcoins are sent to Embedded Downloads as fee and a very small amount to the receiver of the message. A small amount of bitcoins is also fixed as mining fees. A unique transaction ID is returned to the sender. The receiver gets the same TXID only after receiving a notification and wallet updation. Embedded Downloads charges bitcoins to the user based on the size of the message sent. The fees table for this is stored with the application.

In the second step, the message is encrypted using the TXID (symmetrically) and the Public Key (asymmetrically) of the receiver. A message module with the following fields is constructed:

- TAG: SecureMessage
- Hash of TXID
- Receiver's wallet address
- Encrypted Message/Attachments
- CRC

This is sent over https to the private blockchain (PBC). If the message is validated, it is added to the blockchain and an intimation is sent by PBC to the MQTT notification server containing:

- TAG: SecureMessage
- Hash of TXID
- Receiver's wallet address



Message size: XX MB
0.0xxx bit coins will be deducted from your account.

Message Sending In Progress...

Message sent successfully
0.0xxx bit coins deducted from your account.
Transaction Id:
addd8cdd501aafb30501f30db
8f513dd95a1ec198b10704e
4369bdfa5b4ae8a7

Do you want to save the message?

Error: Bitcoin Transaction Failed...

Error: Message Sending Failed...

Figure 11: Sending Messaging UI Sequence

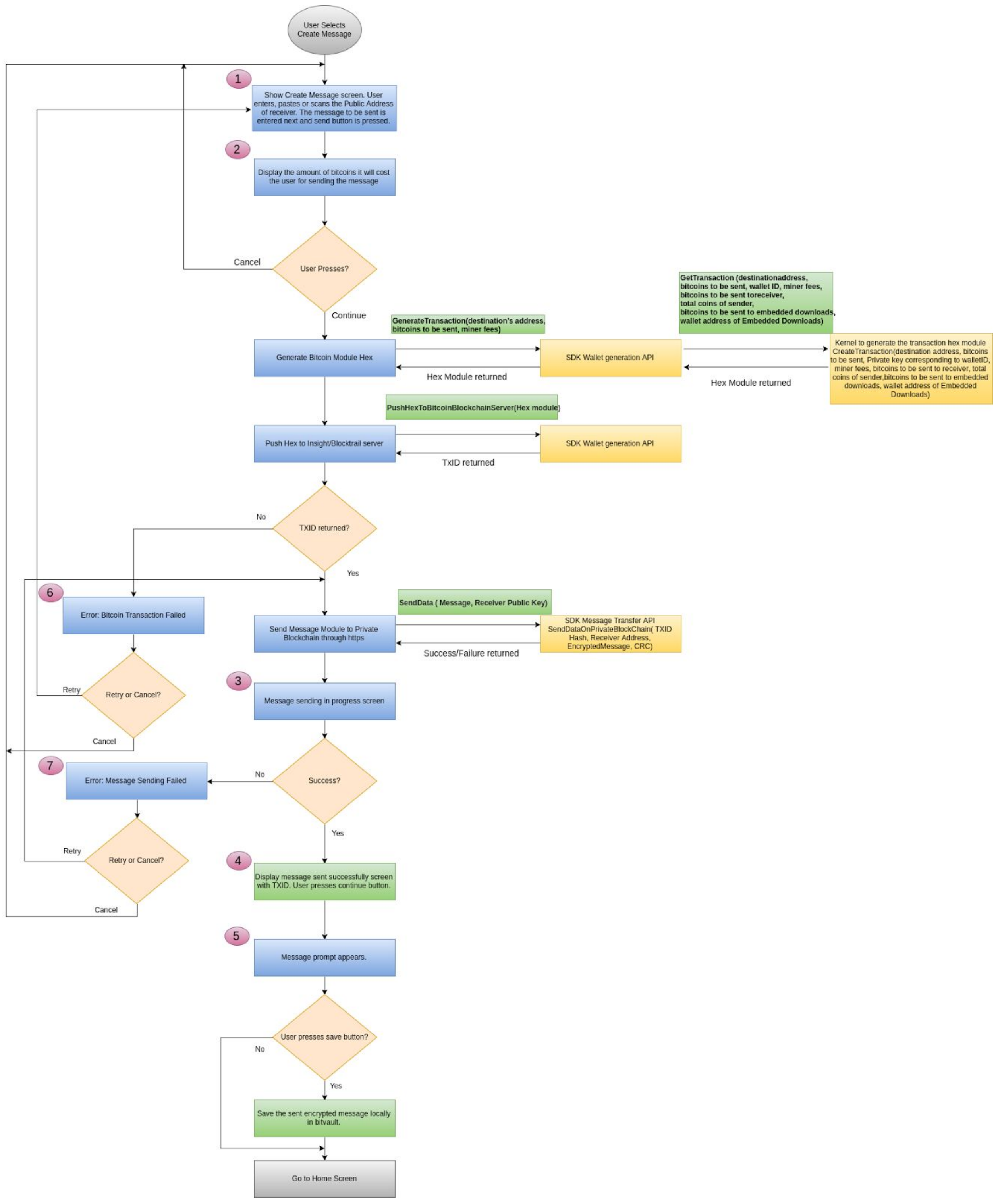

User Selects Create Message
1
Show Create Message screen. User enters, pastes or scans the Public Address of receiver. The message to be sent is entered next and send button is pressed.
2
Display the amount of bitcoins it will cost the user for sending the message
User Presses?
Cancel
Continue
Generate Bitcoin Module Hex
GenerateTransaction(destination's address, bitcoins to be sent, miner fees)
SDK Wallet generation API
Hex Module returned
GetTransaction (destinationaddress, bitcoins to be sent, wallet ID, miner fees, bitcoins to be sent toreceiver, total coins of sender, bitcoins to be sent to embedded downloads, wallet address of Embedded Downloads)
Kernel to generate the transaction hex module CreateTransaction(destination address, bitcoins to be sent, Private key corresponding to walletID, miner fees, bitcoins to be sent to receiver, total coins of sender,bitcoins to be sent to embedded downloads, wallet address of Embedded Downloads)
Hex Module returned
PushHexToBitcoinBlockchainServer(Hex module)
Push Hex to Insight/Blocktrail server
SDK Wallet generation API
TxID returned
TXID returned?
No
Yes
6
Error: Bitcoin Transaction Failed
Retry or Cancel?
Retry
Cancel
SendData (Message, Receiver Public Key)
Send Message Module to Private Blockchain through https
SDK Message Transfer API SendDataOnPrivateBlockChain(TXID Hash, Receiver Address, EncryptedMessage, CRC)
Success/Failure returned
3
Message sending in progress screen
Success?
No
7
Error: Message Sending Failed
Retry or Cancel?
Retry
Cancel
Yes
4
Display message sent successfully screen with TXID. User presses continue button.
5
Message prompt appears.
User presses save button?
No
Yes
Save the sent encrypted message locally in bitvault.
Go to Home Screen

Figure 12: Sending Messaging Flow

1.2.1 Compose Message

This screen allows the user to enter, paste or scan the QR code of the receiver public key. The message is typed and documents, images, videos are other files can be attached.

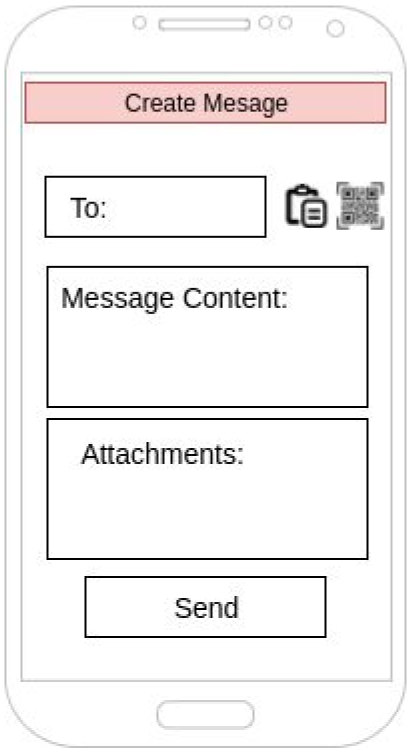
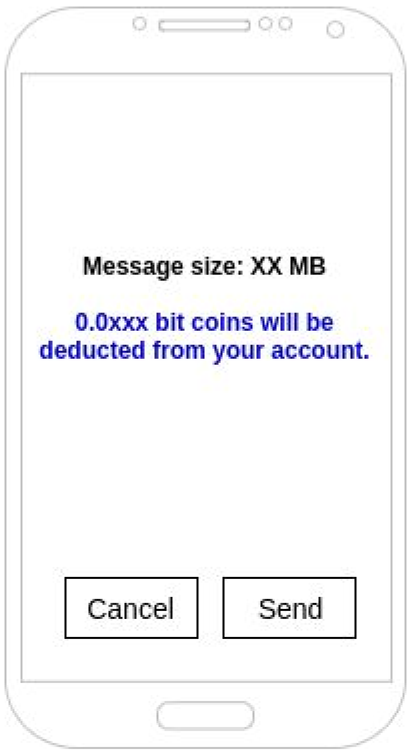


Figure 13: Create Message

On pressing send, the Message Fees UI opens which tells the user the total message and attachment size and amount of bitcoins that will be charged as fees for sending the message. This is retrieved from the message fees table stored with the application. If the user presses send again, message sending process begins. On pressing cancel, the UI goes back to the compose message screen and no message is sent.


Message size: XX MB

0.0xxx bit coins will be deducted from your account.

Figure 14: Message Fees

The sending message screen is then displayed. In the background, the application proceeds in these steps:

- Bitcoin transaction
- Encryption
- Sending to Private Blockchain

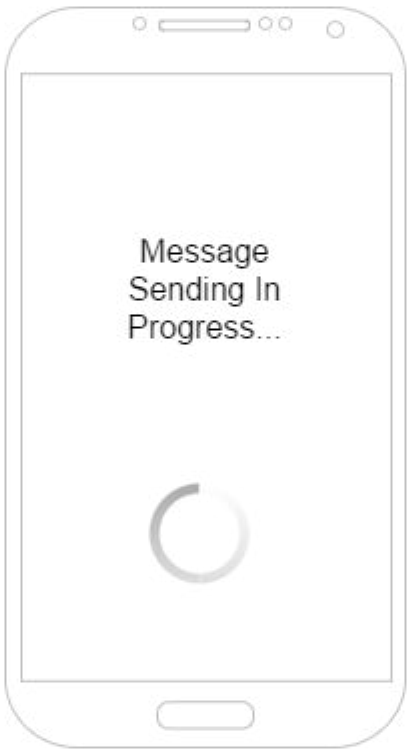


Figure 15: Message Sending

1.2.2 Bitcoin Transaction

The device finds the bitcoin parameters from the application and create a hex using GenerateTransaction(<parameters>) API. SDK will perform the bitcoin transaction.

The <parameters> are:

1. Bitcoin address of Sender A
2. Private Key of Sender A (encrypted)
3. Total coins remaining of A
4. Bitcoin address of Embedded Downloads
5. Messaging fees to be sent for messaging
6. Bitcoin to be sent to receiver B
7. Bitcoin Address of receiver B
8. Miner Fees

Following table shows the message fees respective to the size of the message.

Message Size	**Message fees (Paid to Embedded Downloads Wallet)**	**Miner fees**
Message < 1kB	0.0001BTC	0.0001BTC
1kB < Message < 10kB	0.0005BTC	0.0001BTC
10kB < Message < 100kB	0.0010BTC	0.0001BTC
100kB < Message < 1MB	0.0015BTC	0.0001BTC
1MB < Message < 10MB	0.002BTC	0.0001BTC
10MB < Message < 100MB	0.005BTC	0.0001BTC

When the application requests to send the message to the blockchain, SDK will first initiate the bitcoin blockchain.

Generate the transaction:

Application will perform the following functions:

Description	Generates a HEX for the bitcoin transaction
Method	GenerateTransaction (destination's address, bitcoins to be sent, miner fees)
Input Parameters	Bitcoin to be sent to receiver B Bitcoin Address of receiver B Miner Fees Type: String
Output Parameters	Hex Module Type: Object

Once the application layer has requested to make the transaction, SDK layer will receive the parameters and add more parameters. SDK will send this to Kernel layer for getting the final parameter which is the private key information.

SDK will use the following method in this case:

Description	Generates a HEX for the bitcoin transaction
Method	GetTransaction (destination address, bitcoins to be sent, wallet ID, miner fees, bitcoins to be sent to receiver, total coins of sender, bitcoins to be sent to embedded downloads, wallet address of Embedded Downloads)
Input Parameters	Bitcoin to be sent to receiver B Bitcoin Address of receiver B Miner Fees Wallet ID corresponding to the app ID Total bitcoins of sender Bitcoins to be sent to Embedded Downloads Wallet address of Embedded Downloads Type: String
Output Parameters	Hex Module Type: Object

Kernel layer will finally create the full transaction by adding the last parameter to the transaction module.

Description	Generates a HEX for the bitcoin transaction
Method	CreateTransaction (destination address, bitcoins to be sent, Private key corresponding to wallet ID, miner fees, bitcoins to be sent to receiver, total coins of sender, bitcoins to be sent to embedded downloads, wallet address of Embedded Downloads)
Input Parameters	Bitcoin to be sent to receiver B Bitcoin Address of receiver B Miner Fees Private key of Wallet ID corresponding to the app ID Total bitcoins of sender Bitcoins to be sent to Embedded Downloads Wallet address of Embedded Downloads

	Type: String
Output Parameters	Hex Module Type: Object

SDK layer will receive the hex module and push it to the bitcoin blockchain using Insight server.

Description	Pushes the hex to the blockchain using insight/blocktrail
Method	PushHexToBitcoinBlockchainServer (Hex module)
Input Parameters	Hex Module Type: Object
Output Parameters	Transaction ID Type: String

This transaction ID will be returned to the application layer.

1.2.3 Send to Private Blockchain

The message will be encrypted twice. TXID will be used for symmetric encryption since both the sender and receiver will receive it. The public key will be used for asymmetric encryption over the previous output. Encryption will be performed by the SDK layer.

The message/attachments will be sent to the PBC using the API *SendData (Data, Receiver public key)* at application layer. The message/attachment is sent to the SDK layer that will call *SendDataOnPrivateBlockChain (TAG, Encrypted data, transaction ID, Receiver address, CRC)*

The Message Module has the TAG:SecureMessage, hash of TXID, Receiver Address, Encrypted Message and CRC. This is sent to the private blockchain after https encryption. Each message is uploaded to the nodes of the blockchain and validated. If uploaded, message sent successfully screen is shown. The user can choose to save the message to the sent messages folder.

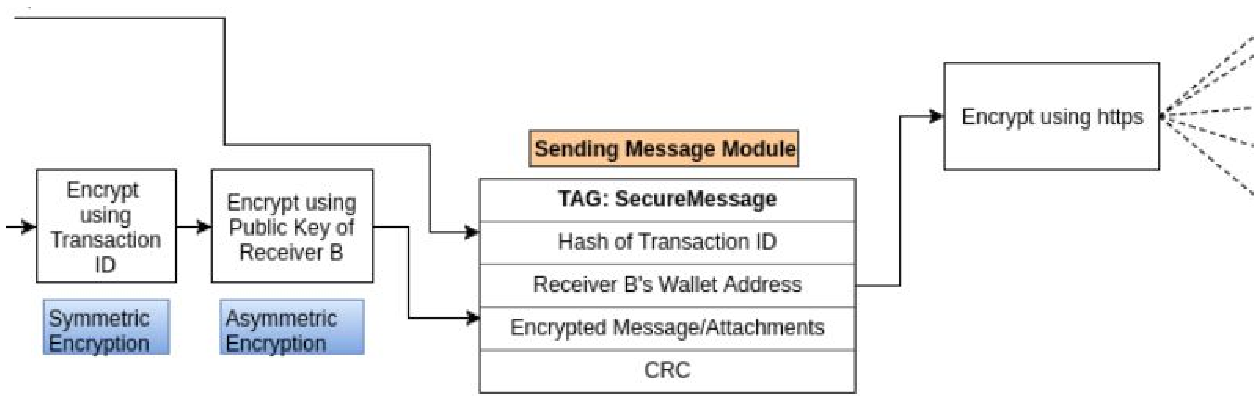
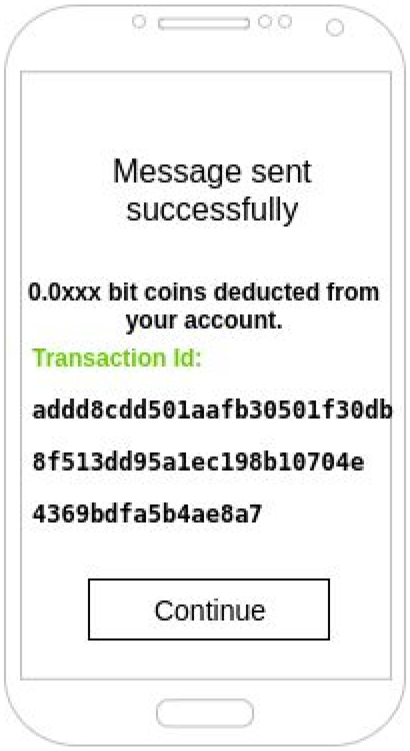


Figure 16: Encrypt and send Message

Pseudo code:

```
CipherSym = SymmetricEncryption (Message, TXID, SymEncryAlgorithm)
CipherAsym = AsymmetricEncryption (CipherSym, PublicKeyReceiver, AsymEncryAlgorithm)
MessageBody = TAG_SecureMessage + HashOfTXID + ReceiverWalletAddress + CipherAsym
CipherCRC = CRC32 (MessageBody)
HttpsEncryptSend ( MessageBody + CipherTxidCRC )
```



Message sent successfully

0.0xxx bit coins deducted from your account.

Transaction Id:

addd8cdd501aafb30501f30db

8f513dd95a1ec198b10704e

4369bdfa5b4ae8a7

Figure 17: Message sent

Application layer will perform the following method:

Description	Encrypts the message using symmetric encryption
Method	SymmetricEncryption (Message, TXID, SymEncryAlgorithm)
Input Parameters	Message, TXID Type: String SymEncryAlgorithm Type: Algo Type (e.g. AES)
Output Parameters	Cipher Type: String

SDK will encrypt the message and will send it to the private blockchain.

Description	Encrypts the message using asymmetric encryption
Method	AsymmetricEncryption (Message, PublicKeyReceiver, AsymEncryAlgorithm)
Input Parameters	Message Type: String PublicKeyReceiver Type: String AsymEncryAlgorithm Type: Algo Type (e.g. RSA)
Output Parameters	Cipher Type: String

Description	Calculates the CRC32 of the text
Method	CRC32 (Text)
Input Parameters	Text Type: String
Output Parameters	CRC output Type: String

Description	Encrypts the text to https standard
Method	HttpsEncryptSend (text)
Input Parameters	Text

	Type: String
Output Parameters	Https encrypted Type: String

Description	Sends the message to blockchain
Method	SendDataOnPrivateBlockChain (TAG, TXID, Receiver Wallet Address, EncryptedMessage, CRC)
Input Parameters	TAG, TxID, Encrypted Message, Receiver Address, CRC Type: Hex module
Output Parameters	Success/Failure Type: boolean

1.3 Receive Messages in Inbox

When the private blockchain has validated a sent message and added it to the blockchain, the notification server will be intimated by it with the TAG:SecureMessage, Receiver's public address and Hash of TXID. Server has a list of devices identified with their unique device ID and corresponding wallet public addresses. The list is added when devices first connect to the notification server or update their wallets. This device ID is a token for sending push notifications. Notification server finds the mapped device ID using the receiver's wallet public address. Push notification is sent to that device for which a new secure message has been received on the wallet public address.

The App will update the wallet on the bitcoin blockchain with its public address to get the list of latest transaction IDs. Hash values of these transaction IDs are found and one which matches with the received hash confirms the actual TXID corresponding to the Secure Message. The TXID hash will be used to poll the private blockchain to receive the encrypted message/attachment along with CRC from PBC.

For BitVault Attachment keys, the application will display a QR code of the decrypted Session Key and TXID for scanning on desktop.

The Inbox screen shows the unread messages and the saved messages.

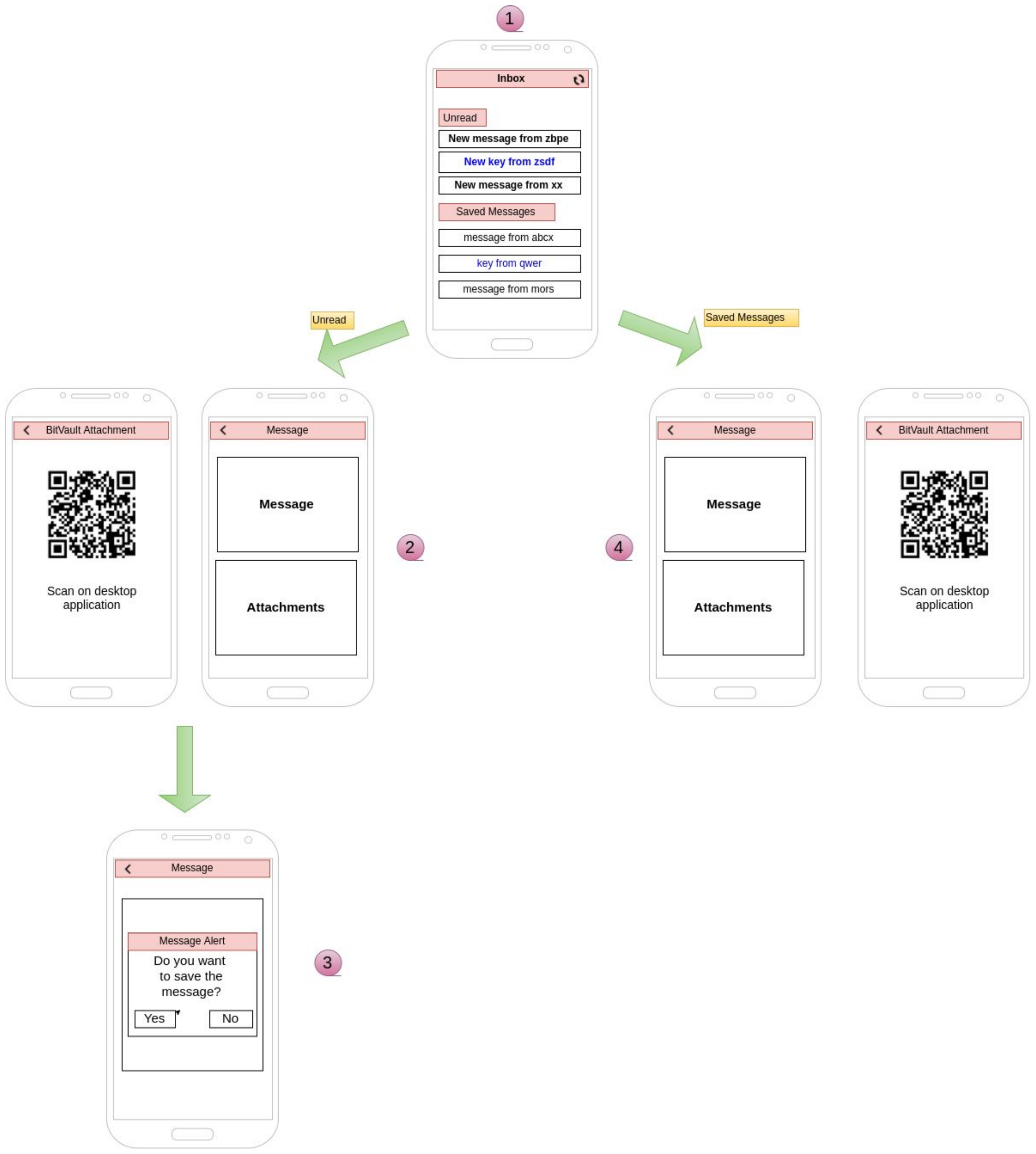


Figure 18: Inbox UI Sequence

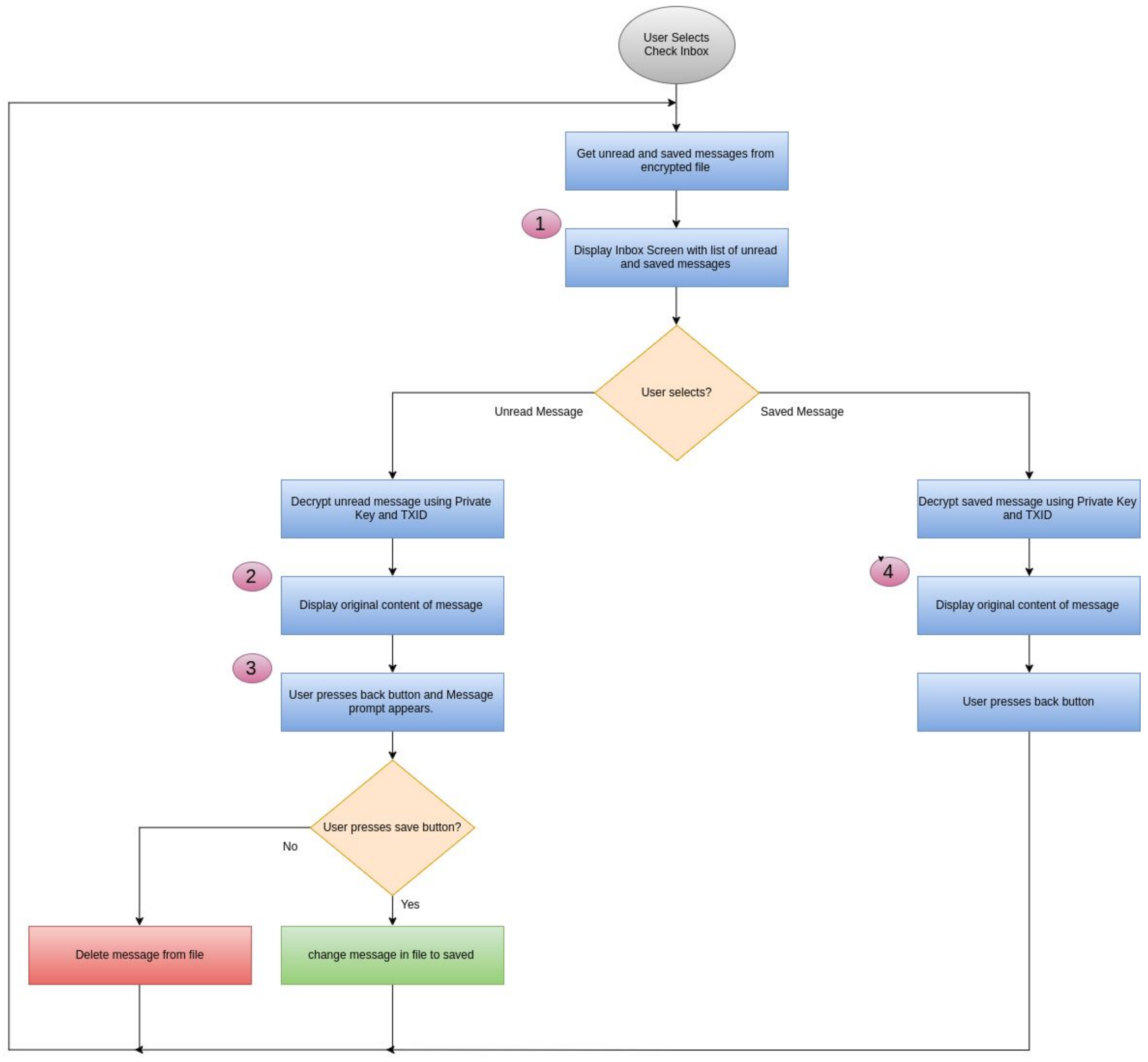

User Selects Check Inbox
Get unread and saved messages from encrypted file
1
Display Inbox Screen with list of unread and saved messages
User selects?
Unread Message
Saved Message
Decrypt unread message using Private Key and TXID
2
Display original content of message
3
User presses back button and Message prompt appears.
User presses save button?
No
Yes
Delete message from file
change message in file to saved
Decrypt saved message using Private Key and TXID
4
Display original content of message
User presses back button

Figure 19: Inbox flow

1.3.1 Notification

The server sends a push notification to the device that a new Secure Message has been sent for it. This is received by device even if it was in sleep condition. It sends:

- TAG: SecureMessage
- Hash of TXID
- Receiver Public Address

The device can then download the message/attachments using TAG and hash of TXID.

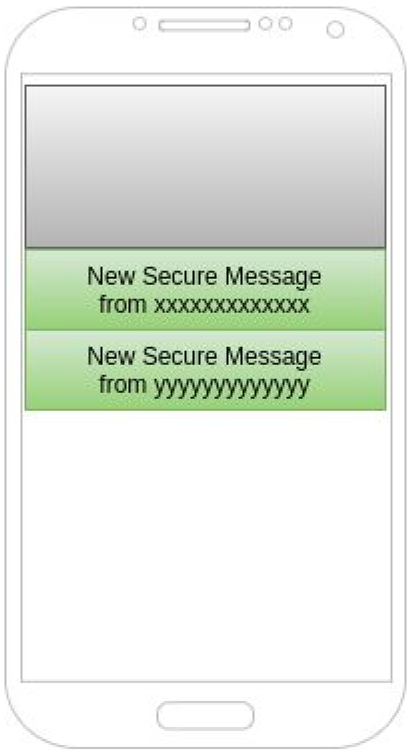

New Secure Message from xxxxxxxxxxxxx
New Secure Message from yyyyyyyyyyyyy

Figure 20: New Secure Message Notification

1.3.2 Download Message

New messages are downloaded when notifications are received from the notification server. All new transactions are found using *GetTransactionHistory (Wallet Address)* that returns a list of all new Transaction IDs from the Bitcoin Blockchain. Since the hash is received from the notification, the corresponding TXID can be compared and found. The *CheckDataOnBlockChain (TAG, Transaction ID)* will check if any messages are available in the blockchain with the TAG of type Secure Messaging and hash of the TXID. If the block is present, it downloads it using *DownloadData ()*.

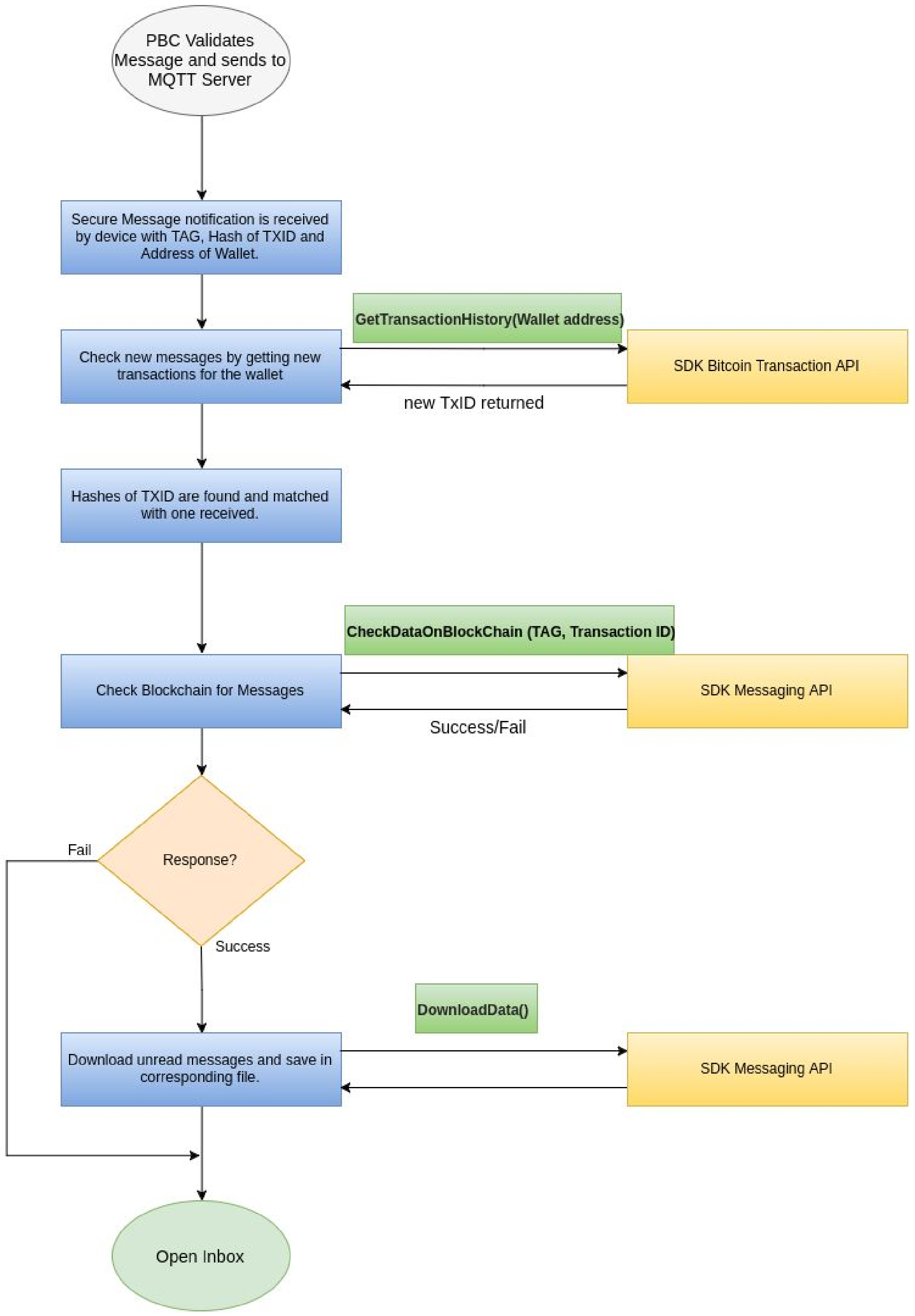

PBC Validates Message and sends to MQTT Server
Secure Message notification is received by device with TAG, Hash of TXID and Address of Wallet.
GetTransactionHistory(Wallet address)
Check new messages by getting new transactions for the wallet
SDK Bitcoin Transaction API
new TxID returned
Hashes of TXID are found and matched with one received.
CheckDataOnBlockChain (TAG, Transaction ID)
Check Blockchain for Messages
SDK Messaging API
Success/Fail
Fail
Response?
Success
DownloadData()
Download unread messages and save in corresponding file.
SDK Messaging API
Open Inbox

Figure 21: Download from Private Blockchain

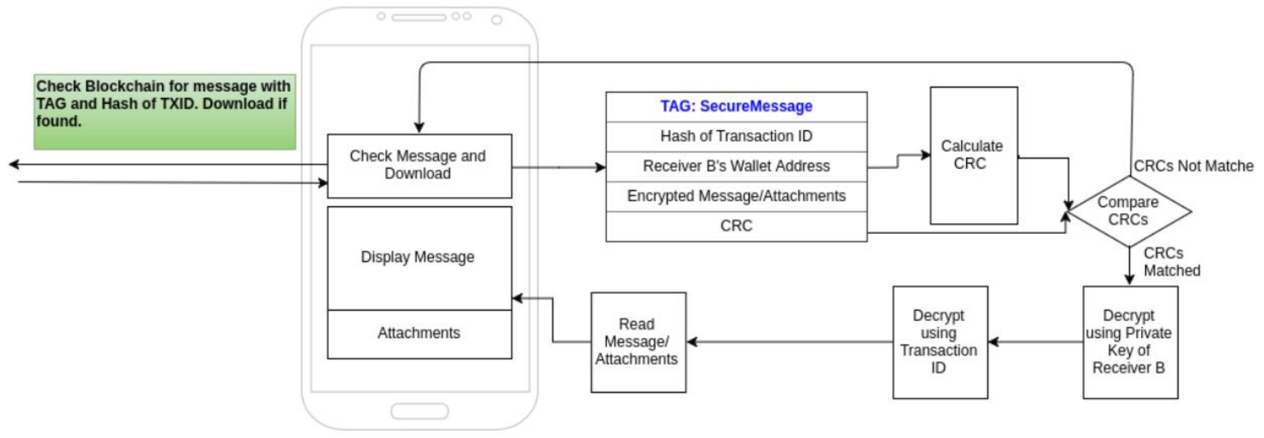

Check Blockchain for message with TAG and Hash of TXID. Download if found.
Check Message and Download
Display Message
Attachments
TAG: SecureMessage
Hash of Transaction ID
Receiver B's Wallet Address
Encrypted Message/Attachments
CRC
Calculate CRC
Compare CRCs
CRCs Not Matche
CRCs Matched
Decrypt using Private Key of Receiver B
Decrypt using Transaction ID
Read Message/ Attachments

Figure 22: Validate and decrypt message/attachment

Description	Checks the blockchain for new message using TAG and hash of TXID
Method	CheckDataOnBlockChain (TAG, Transaction ID)
Input Parameters	TXID Type: String
Output Parameters	True/False Type: Boolean

Description	Download Data
Method	*DownloadData ()*
Input Parameters	None
Output Parameters	True/False Type: Boolean

On downloading the encrypted message with CRC, the CRC of the encrypted message is generated and matched with the received CRC.

```
if(CRC matches)
{
        // Download the message
        //Show in inbox
        //send acknowledgement to private blockchain
}
else
{
        //Show Error: unable to download or poll again
}
```

When there is a new message for a user, SDK will validate the CRC of the encrypted message in order to check if the message received is the original message or the altered/corrupted message.

Description	Validate the CRC of the message
Method	ValidateCRC (MessageModuleBody, CRCfromMessage)
Input Parameters	MessageModuleBody Type: String CRCfromMessage Type: Long
Output Parameters	Success/Fail Type: boolean

1.3.3 Unread and Saved Messages

On selecting a message from unread or saved, the application retrieves the encrypted message from the storage and displays them after decrypting with corresponding TxID and Private Key. Kernel layer will first decrypt the encrypted message using the private key as only kernel layer has the access of the private keys and then the SDK layer will do the second level of the decryption. Once the two layer of decryption is done, the message will be completely decrypted.

If an unread message is displayed, the user can choose to save the message after reading it.

Pseudo code:

```
//Kernel to decrypt using private key

DecryptedAsym = AsymmetricDecryption (EncryMessage, PrivateKeyReceiver,
AsymDecryAlgorithm)

//SDK to decrypt using transaction ID

OriginalMessage = SymmetricDecryption (DecryptedAsym, TXID, SymDecryAlgorithm)
```

Kernel layer decrypts the first layer of encrypted message by using the below method:

Description	Decrypts the message using asymmetric decryption
Method	AsymmetricDecryption (EncryMessage, PrivateKeyReceiver, AsymDecryAlgorithm)
Input Parameters	Encrypted Message Type: String PrivateKeyReceiver Type: String AsymDecryAlgorithm Type: Algo Type (e.g. RSA)
Output Parameters	Decrypted Byte array Type: encrypted byte []

SDK layer will perform the second layer of decryption using the transaction ID

Description	Decrypts the message using symmetric decryption
Method	SymmetricDecryption (EncryMessage, TXID, SymDecryAlgorithm)
Input Parameters	Encrypted Message Type: String TXID Type: String SymDecryAlgorithm Type: Algo Type (e.g. AES)
Output Parameters	Byte array Type: byte []

Application will receive the decrypted text message by the SDK layer. The decrypted message will be shown in the application.

1.4 Sent Messages

Sent Messages can be seen on this UI. Once the user sends the message to another user, the encrypted message is saved in the sent items. When user clicks on any of the sent messages, the public key, encrypted message/attachments and the corresponding transaction ID can be seen on the screen.

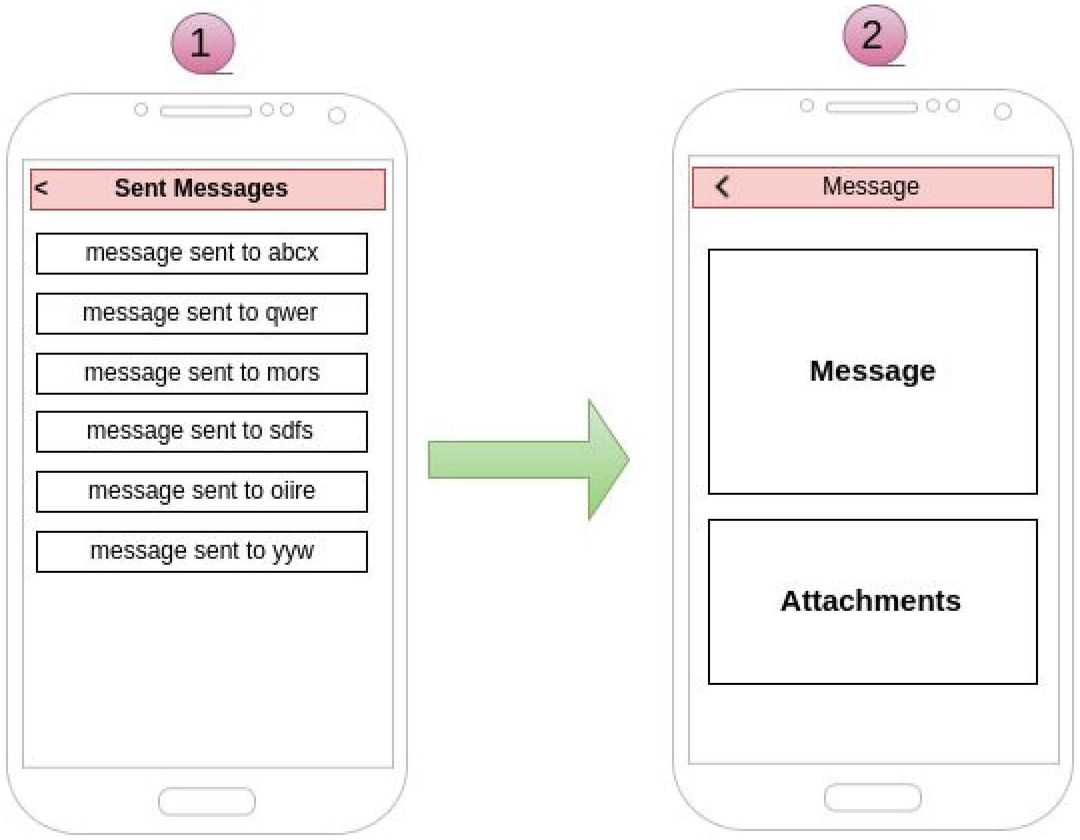


Figure 23: Sent Messages

1.5 BitVault Attachment

BitVault Attachment application on the desktop will be used for sending and receiving files over Private Blockchain (PBC). Files can either be sent from desktop to another desktop or from the BitVault phone to the user's desktop. The Secure Messaging application will initiate the bitcoin transactions, send the session key (desktop to desktop) and send attachments (desktop to BitVault phone).

For desktop to desktop, the Secure Messaging application will initiate a bitcoin transaction and send an encrypted Session Key to the receiver BitVault(phone) via PBC. This key, which is used for encrypting the files on desktop, is encrypted with the public key of the receiver's wallet. Receiver can decrypt this key using its wallet private key in BitVault. The TXID is scanned on the Sender desktop application. The files are encrypted with TXID and Session Key and sent to PBC. On getting a notification, the receiver BitVault downloads the Session Key and TXID (from Bitcoin Blockchain) which can then be entered or scanned to the desktop application. The files can finally be decrypted and viewed by receiver.

In the second case, the user will generate a session key on the desktop for receiving files. This will be scanned by the BitVault using QR code. A bitcoin transaction will be initiated. The returned TXID and session key will be used for encrypting and sending the files to PBC. On getting the notification on desktop, user can download the files after entering or scanning the QR code of the TXID. The files can then be decrypted and viewed by user.

1.5.1 Desktop Registration

The user has to register his desktop with the notification server for receiving notifications. When the application on desktop is first installed, it will ask the user to scan a QR code from the BitVault phone. The QR code will have the phone token/id with which it is registered with the server for notifications. The desktop will scan this code and register itself with the notification server. This will enable receiving notifications on desktop for BitVault Attachments.

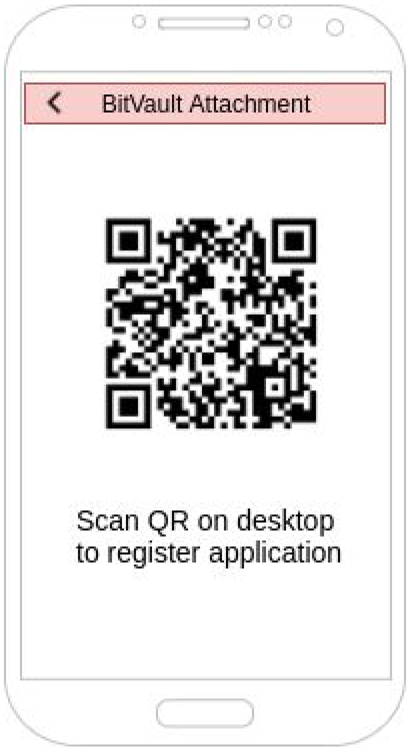
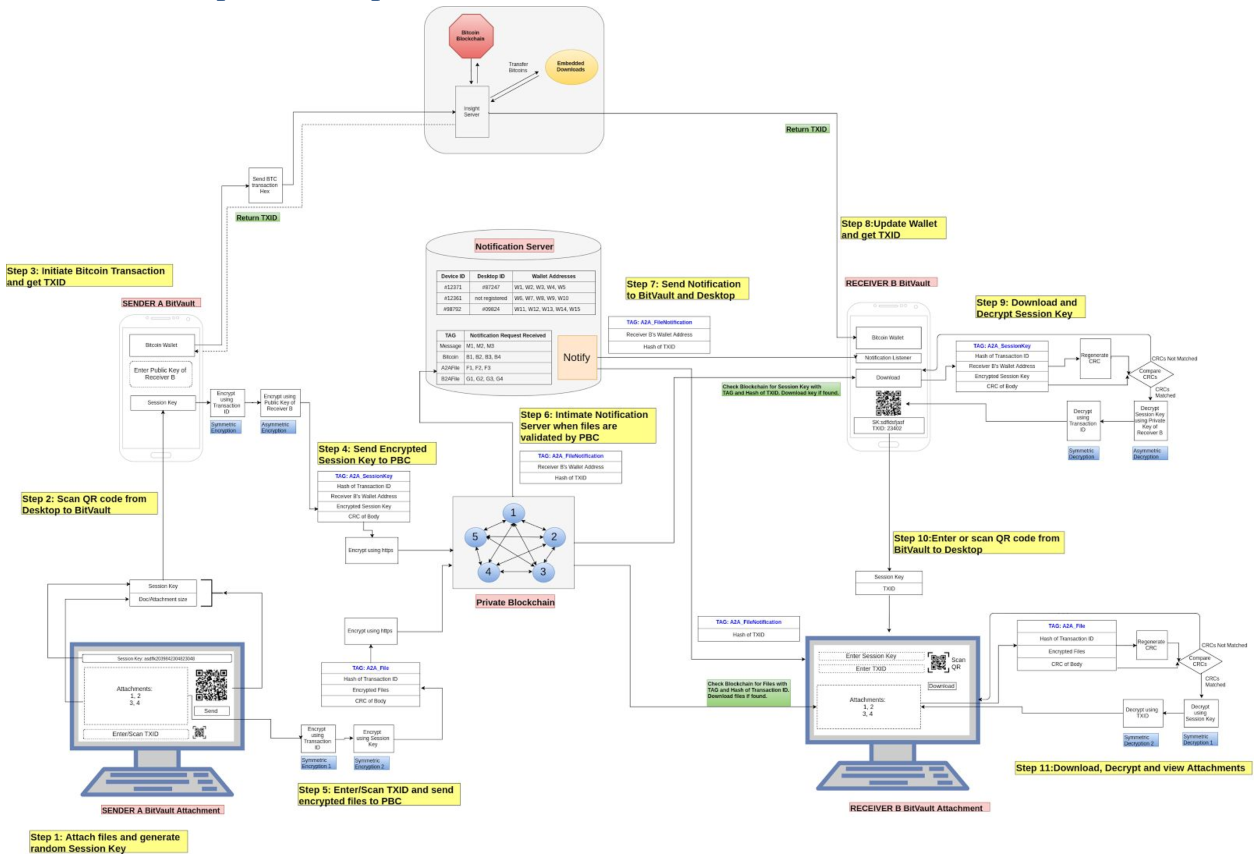


Figure 24: Desktop Application Registration

1.5.2 Desktop to Desktop



Bitcoin Blockchain
Transfer Bitcoins
Embedded Downloads
Insight Server
Return TXID
Send BTC transaction Hex
Return TXID
Step 3: Initiate Bitcoin Transaction and get TXID
SENDER A BitVault
Bitcoin Wallet
Enter Public Key of Receiver B
Session Key
Encrypt using Transaction ID
Encrypt using Public Key of Receiver B
Symmetric Encryption
Asymmetric Encryption
Notification Server
Notify
Step 7: Send Notification to BitVault and Desktop
TAG: A2A_FileNotification
Receiver B's Wallet Address
Hash of TXID
Step 8:Update Wallet and get TXID
RECEIVER B BitVault
Step 9: Download and Decrypt Session Key
Notification Listener
Download
TAG: A2A_SessionKey
Hash of Transaction ID
Receiver B's Wallet Address
Encrypted Session Key
CRC of Body
Regenerate CRC
Compare CRCs
CRCs Not Matched
CRCs Matched
Decrypt using Transaction ID
Decrypt Session Key using Private Key of Receiver B
Symmetric Decryption
Asymmetric Decryption
Check Blockchain for Session Key with TAG and Hash of TXID. Download key if found.
Step 4: Send Encrypted Session Key to PBC
Step 6: Intimate Notification Server when files are validated by PBC
Step 2: Scan QR code from Desktop to BitVault
Encrypt using https
Private Blockchain
Step 10:Enter or scan QR code from BitVault to Desktop
Session Key
TXID
Doc/Attachment size
Attachments: 1, 2 3, 4
Send
Enter/Scan TXID
TAG: A2A_File
Hash of Transaction ID
Encrypted Files
CRC of Body
Encrypt using Transaction ID
Encrypt using Session Key
Symmetric Encryption 1
Symmetric Encryption 2
Step 5: Enter/Scan TXID and send encrypted files to PBC
SENDER A BitVault Attachment
Step 1: Attach files and generate random Session Key
Check Blockchain for Files with TAG and Hash of Transaction ID. Download files if found.
Enter Session Key
Enter TXID
Scan QR
Decrypt using TXID
Decrypt using Session Key
Symmetric Decryption 2
Symmetric Decryption 1
Step 11:Download, Decrypt and view Attachments
RECEIVER B BitVault Attachment

Figure 25: Sending from desktop to desktop

4.1.3.1 Sending

Step 1: Sender A opens the Application on the Desktop. The user can attach documents, images, videos and other files. A random Session Key is generated for encrypting the files. QR code of Session Key and file size is generated and displayed on screen.

Step 2: In Secure Messaging App, user selects sending from BitVault Attachment. The Public Key of receiver can be added from address book, QR scanned or just typed. The BitVault scans the Session Key and File size from desktop app.

Step 3: A Bitcoin transaction is initiated by BitVault. Small amount of bitcoin is sent to Receiver B, fees sent to Embedded Downloads based on file size and a fixed miner fee. TXID is returned to Sender A.

Step 4: The Session Key is encrypted using TXID (symmetrically) and Public Key of Receiver B (asymmetrically) and sent to Private Blockchain with an identifier TAG: A2A_SessionKey along with Hash of TXID, Receiver's Wallet Address, Encrypted Session Key and CRC of body.

Step 5: The User scans or types the TXID in the desktop from the BitVault and presses send. The files are encrypted first with the TXID (symmetrically) and second with the Session Key (symmetrically). The files are sent to the PBC with the TAG: A2A_File, Hash of TXID, Encrypted Files and CRC of body.

Step 6: When PBC has validated the files i.e. compared CRCs and added to blockchain, the notification server is intimated about the file. PBC sends the TAG: A2A_FileNotification, Receiver's wallet address and hash of TXID to the notification server.

Step 7: The notification server has a list of device Ids, corresponding wallet addresses on the device and linked desktop application ID. This linkage helps find the mapped device and desktop. Notification is sent to the device with TAG:A2A_FileNotification, hash of TXID and Wallet Address. Notification is sent to desktop with TAG:A2A_FileNotification and hash of TXID.

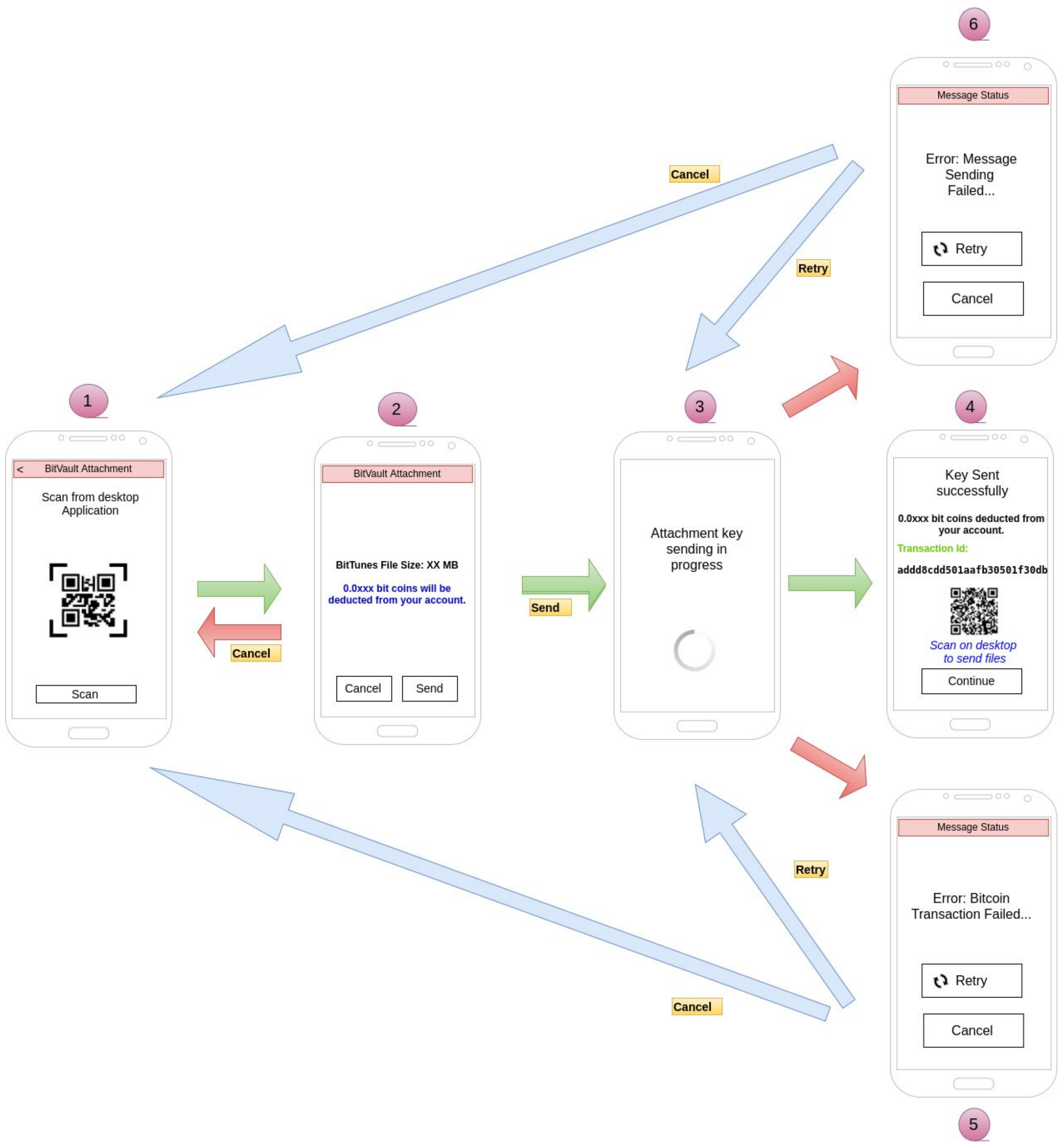

6
Message Status
Error: Message Sending Failed...
Retry
Cancel
Cancel
Retry
1
< BitVault Attachment
Scan from desktop Application
Scan
Cancel
2
BitVault Attachment
BitTunes File Size: XX MB
0.0xxx bit coins will be deducted from your account.
Cancel
Send
Send
3
Attachment key sending in progress
4
Key Sent successfully
0.0xxx bit coins deducted from your account.
Transaction Id:
addd8cdd501aafb30501f30db
Scan on desktop to send files
Continue
Retry
Cancel
Message Status
Error: Bitcoin Transaction Failed...
Retry
Cancel
5

Figure 26: Sending Session Key UI flow

4.1.3.2 Receiving

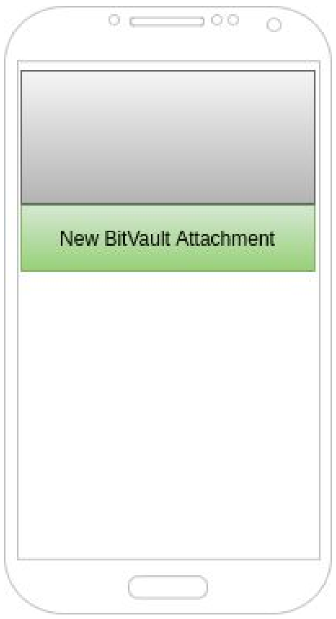


Figure 27. BitVault Attachment Notification

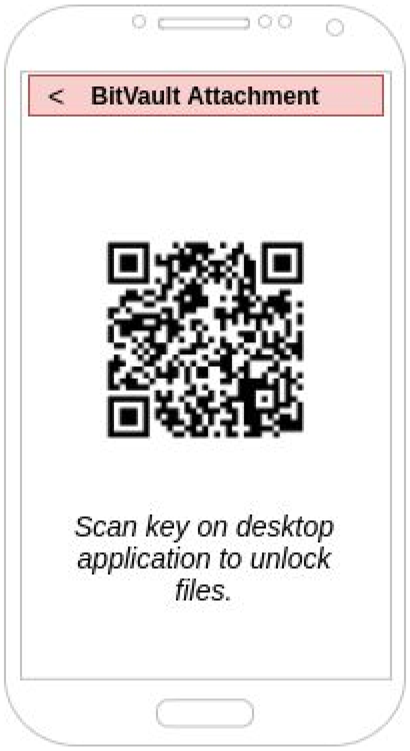


Figure 28: Sharing received session key

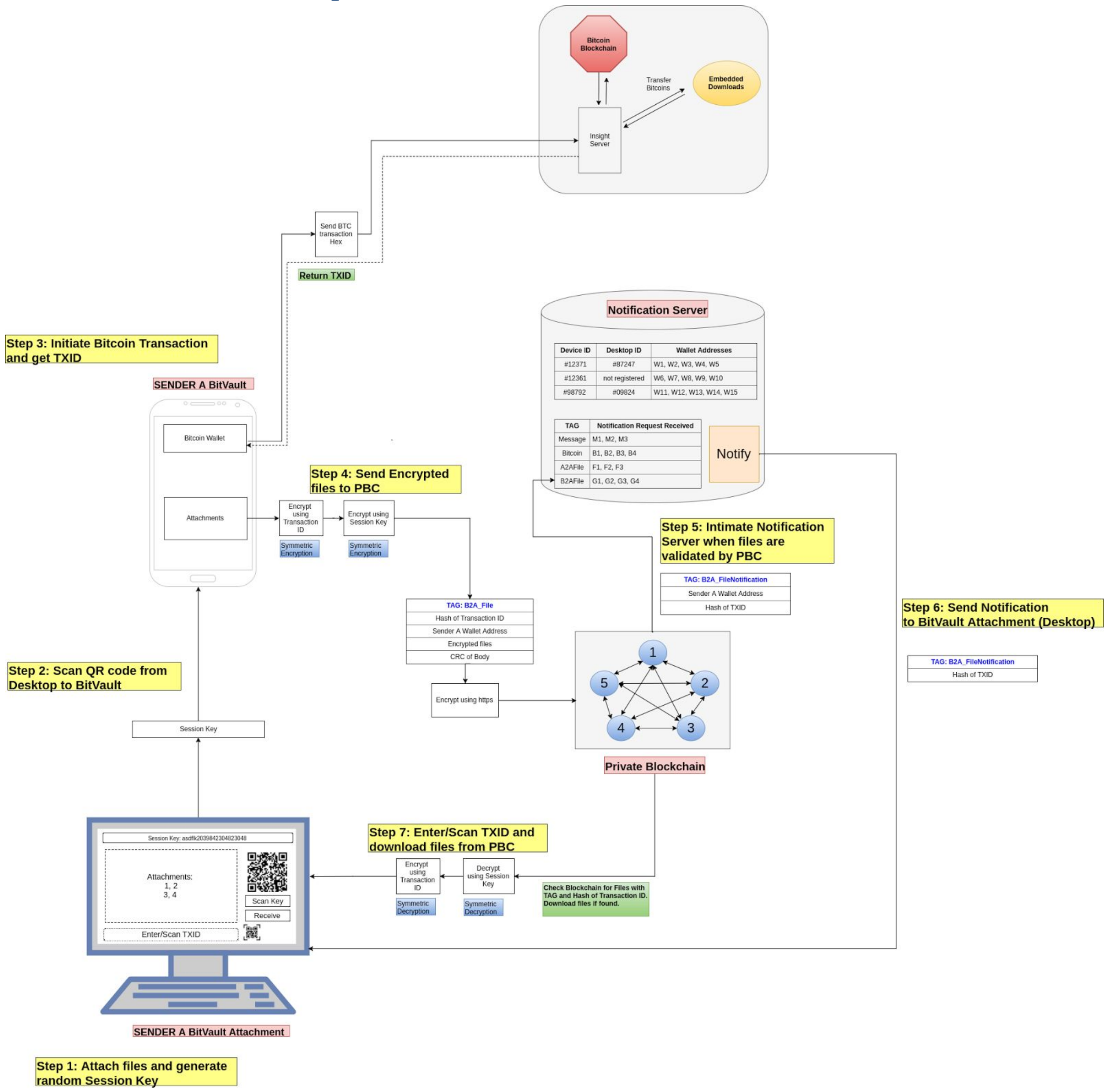
Step 8: Receiver B updates the wallet (from Bitcoin blockchain) whose address was received in the notification. The latest TXIDs are downloaded and their hash values are found.

Step 9: On comparing the hash values of notification vs. wallet update, the corresponding TXID is found . The TAG and TXID hash value are used to download the encrypted Session Key from the PBC. The Private Key of Receiver B, available only in the BitVault wallet, decrypts the Session Key. BitVault of Receiver B now has the TXID and the Session Key.

Step 10: User scans QR or types the Session Key and TXID in desktop.

Step 11: The hash of TXID and TAG is used to download the files from PBC. The Session Key and TXID are used to decrypt the files. The files can then be opened and viewed by Receiver B.

1.5.3 BitVault to Desktop


Bitcoin Blockchain
Transfer Bitcoins
Embedded Downloads
Insight Server
Send BTC transaction Hex
Return TXID
Step 3: Initiate Bitcoin Transaction and get TXID
SENDER A BitVault
Bitcoin Wallet
Attachments
Step 4: Send Encrypted files to PBC
Encrypt using Transaction ID
Symmetric Encryption
Encrypt using Session Key
Symmetric Encryption
TAG: B2A_File
Hash of Transaction ID
Sender A Wallet Address
Encrypted files
CRC of Body
Encrypt using https
Notification Server
Notify
Step 5: Intimate Notification Server when files are validated by PBC
TAG: B2A_FileNotification
Sender A Wallet Address
Hash of TXID
Step 6: Send Notification to BitVault Attachment (Desktop)
TAG: B2A_FileNotification
Hash of TXID
1
2
3
4
5
Private Blockchain
Step 2: Scan QR code from Desktop to BitVault
Session Key
Session Key: asdflk2039842304823048
Attachments: 1, 2 3, 4
Scan Key
Receive
Enter/Scan TXID
Step 7: Enter/Scan TXID and download files from PBC
Encrypt using Transaction ID
Symmetric Decryption
Decrypt using Session Key
Symmetric Decryption
Check Blockchain for Files with TAG and Hash of Transaction ID. Download files if found.
SENDER A BitVault Attachment
Step 1: Attach files and generate random Session Key

Device ID	Desktop ID	Wallet Addresses
#12371	#87247	W1, W2, W3, W4, W5
#12361	not registered	W6, W7, W8, W9, W10
#98792	#09824	W11, W12, W13, W14, W15

TAG	Notification Request Received
Message	M1, M2, M3
Bitcoin	B1, B2, B3, B4
A2AFile	F1, F2, F3
B2AFile	G1, G2, G3, G4

Figure 29: BitVault (phone) to user Desktop

1.5.4 *Sending*

Step 1: Sender A opens the application on the Desktop and selects “Receive files”. A random Session Key is generated on desktop.

Step 2: BitVault scans the QR code of Session Key.

Step 3: A Bitcoin transaction is initiated by BitVault. Bitcoin is sent as fees to Embedded

Downloads based on file size and a fixed miner fee. TXID is returned to Sender A.

Step 4: The files are encrypted using TXID first and then with the Session Key. This is sent to Private Blockchain with an identifier TAG: B2A_File, Hash of TXID, Sender Wallet Address, Encrypted Files and CRC of body.

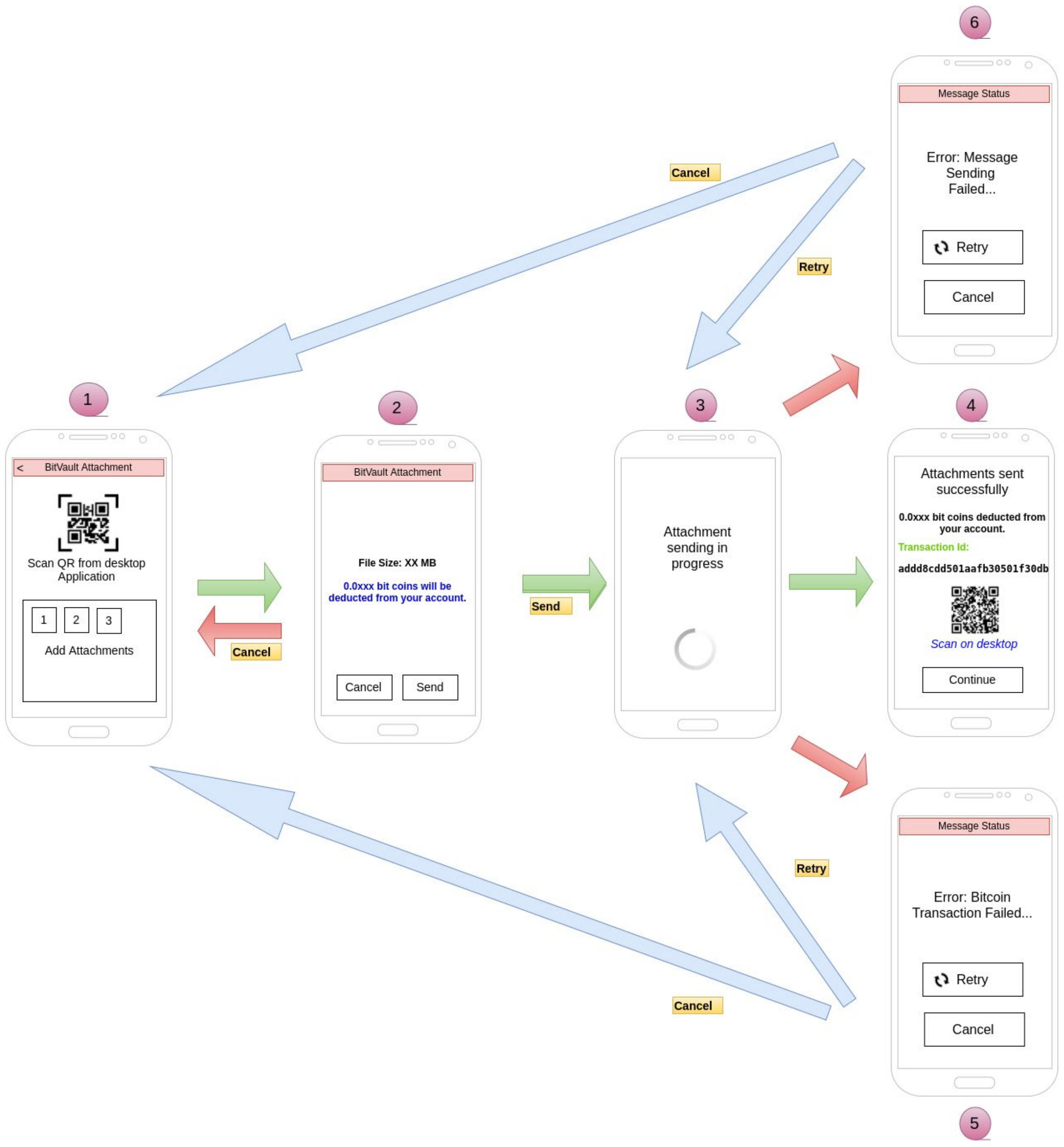

6
Message Status
Error: Message Sending Failed...
Retry
Cancel
Cancel
Retry
1
< BitVault Attachment
Scan QR from desktop Application
1 2 3
Add Attachments
Cancel
2
BitVault Attachment
File Size: XX MB
0.0xxx bit coins will be deducted from your account.
Cancel
Send
Send
3
Attachment sending in progress
4
Attachments sent successfully
0.0xxx bit coins deducted from your account.
Transaction Id:
addd8cdd501aafb30501f30db
Scan on desktop
Continue
Retry
Cancel
Message Status
Error: Bitcoin Transaction Failed...
Retry
Cancel
5

Figure 30: BitVault to desktop UI flow

1.5.5 *Receiving*

Step 5: When PBC has validated the files i.e. compared CRCs and added to blockchain, the notification server is intimated. PBC sends the TAG:B2A_FileNotification, Sender Wallet Address and Hash of TXID to the notification server.

Step 6: The notification server has a list of device IDs, corresponding wallet addresses on device and linked desktop application IDs. A notification is sent to the desktop with a TAG: B2A_FileNotification and Hash of TXID.

Step 7: The user scans or types the TXID on the desktop and downloads the files from the PBC using TAG and Hash of TXID. It is decrypted with Session Key generated earlier and then with the TXID. Files can then be viewed.

Note:

- Wireframes attached just describe the flow, final UI screen will be updated by UI designer Team.