How it works:

The three major components involved in the transaction are the user, merchant and bank servers. The user uses an android application (*Phoenix*) to communicate with merchant and bank servers.

Client, merchant, bank and certificate authority (CA) each have their RSA 512-bit private and public keys. Each entity's (client, merchant and bank) X509v1 certificates were generated using *Bouncy Castle Library*. These certificates were then signed with CA's private key. *Keystore* was used to safely store these certificates and key pairs. We have assumed that each entity has access to other's certificate to retrieve the public key.

Merchant and bank maintain a shared database (*PhoenixDatabase*). This database comprises of four tables:

Table Name	Fields
USERS	ID, NAME, EMAIL, PASSWORD, SALT, PHONE
SESSION	ID, EMAIL, SESSION_CTOM, SESSION_MTOB
PRODUCTS	ID, NAME, PRICE
OTP	ID, SESSION_MTOB, PHONE, OTP

Database functions:

Function	Usage
insert_users	Function inserts a new user in the *Users* table
getSaltString	Function retrieves salt from *Users* table
check_user	Function checks if username, email or phone number is already present in *Users* table
login_activity	Function used during login to verify the details
get_phone_users	Function to retrieve phone number from *Users* table
insert_session	Function to insert *session_ctom* when user logs in
update_session_pay	Function to insert *session_mtob*

get_phone	Function to get phone number from *Users* table using *Session* table
get_session_mtob	Function to get *session_mtob* from *OTP* table
get_mail	Function to get email from *Session* table
insert_OTP	Function to insert details in *OTP* table
get_OTP	Function to get OTP from *OTP* table

The actual process is preceded by four steps:

- Key and certificate generation.
- Storage and distribution of keys and certificates.
- Tables' generation.
- Populating the Products Table.

When the user opens *Phoenix* application on his Android phone, he is directed to the *LoginActivity* page. The login page has the following fields:

- Username
- Email
- Password
- Link to Registration Page

If the user has an account, he fills in the details and hits the login button; else he hits the link to registration screen button and gets directed to the registration page. If any of the fields is empty, a toast appears asking him to furnish all the details. If all the fields are given, a session key (*session_ctom*) is generated along with a timestamp. This session key is unique and will be used for all message exchanges between merchant and client in this transaction. A message is created E(K_m^+ ,[Identifier, Username, Password, Email, TimeStamp, *session_ctom*]) and sent to merchant server.

Merchant on receiving this message:

- Decrypts the information and checks the timestamp (a 30 second window is allowed)
- If timestamp is valid, it calls the *login_activity* function. (function checks if the details supplied are correct in the *Users* table)
- The function raises a flag, and accordingly a verification message is created. (successful/unsuccessful)
- If the login is successful, an entry in *Sessions* table is created.
- A message is then created: E(K_c^+ , [Verification Message, TimeStamp, *session_ctom*, Phone Number]) and then sent to client.

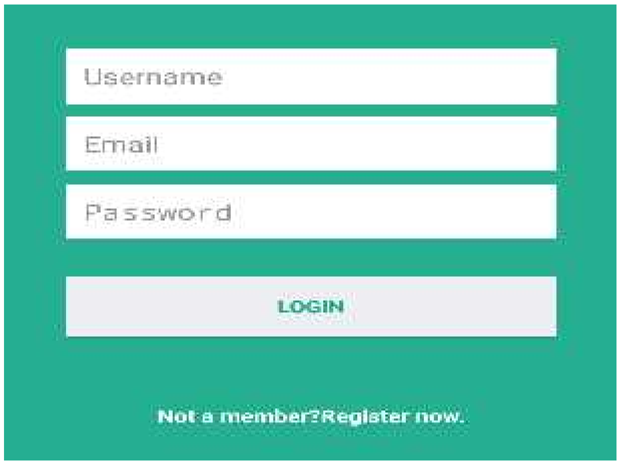


Fig E.2 User Login Page

Client on receiving this message:

- Verifies timestamp.
- Launches *ProductsActivity* for successful login and stores the phone number received in its own *Users* table.
- For an unsuccessful login, it generates a login failure toast message.

If the user doesn't have an account, he can register himself from *Phoenix's* Registration Page. The registration page has the following fields:

- Username
- Email
- Phone number
- Password

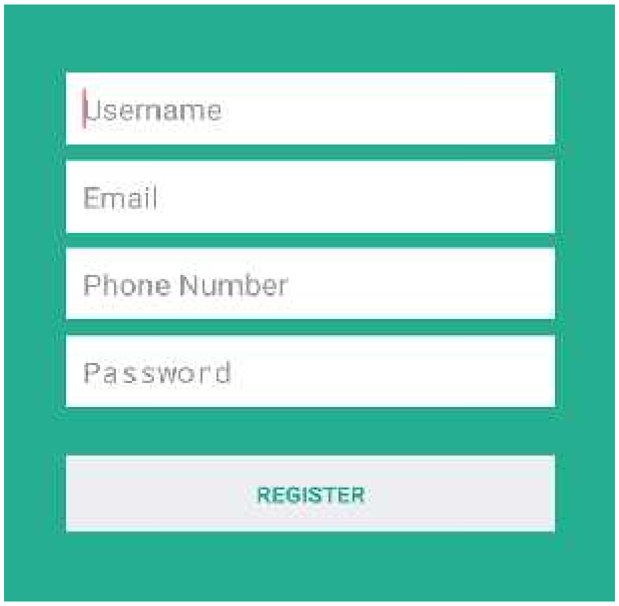


Fig E.3 User Registration Page

If any of the fields is empty, a toast appears asking him to furnish all the details. A message is created: E(K_m^+ ,[Identifier, Username, Password, Email, Phone Number] and sent to merchant server.

The server on receiving it:

- Decrypts the message and passes the fields to *insert_users* functions.
- The function in turn uses a *check_user* function to check if a user exists with the same parameters.
- If the user is new, a salt is created. New password is created concatenating *secret*, username, password and the salt generated, hashed using *HmacSHA1* and stored in the *Users* table.
- Accordingly a flag is generated; an encrypted message is drafted and then sent with the fields: verification message and timestamp.

Client on receiving the message verifies the timestamp and directs user to the Login page for successful login or displays a toast for unsuccessful login.

After a successful login, the user is directed to *ProductsActivity* page in the application. The application creates a message: E(K_m^+ , [Identifier, TimeStamp, *session_ctom*] and sends it to merchant server. The server verifies the time stamp and sends back the list of products along with a concatenated hash of products and *session_ctom* (MD-5).

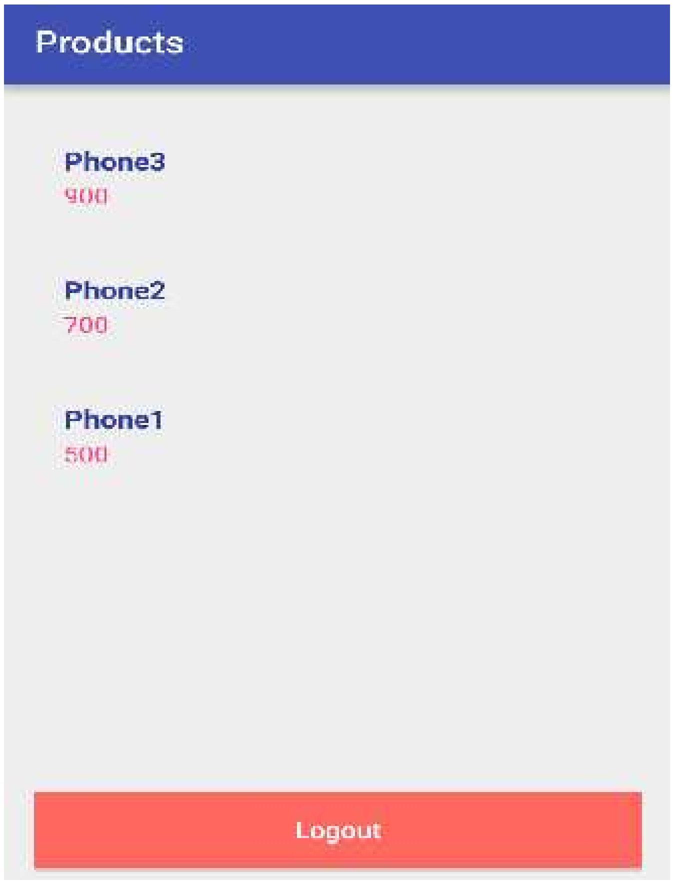


Fig E.4 Products List

Client on receiving this verifies the hash and displays the product list. The user then chooses a product and is directed to *PaymentActivity* page.

This page has the following fields:

- Card type
- Card number
- Expiration date
- CVV

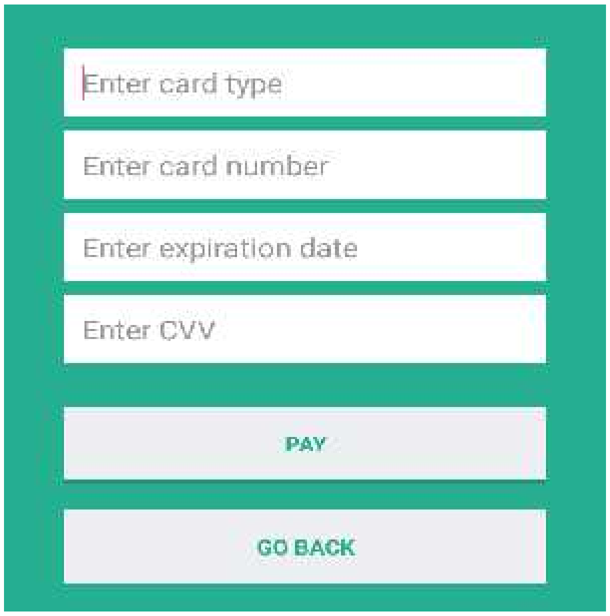


Fig E.5 Payment Details

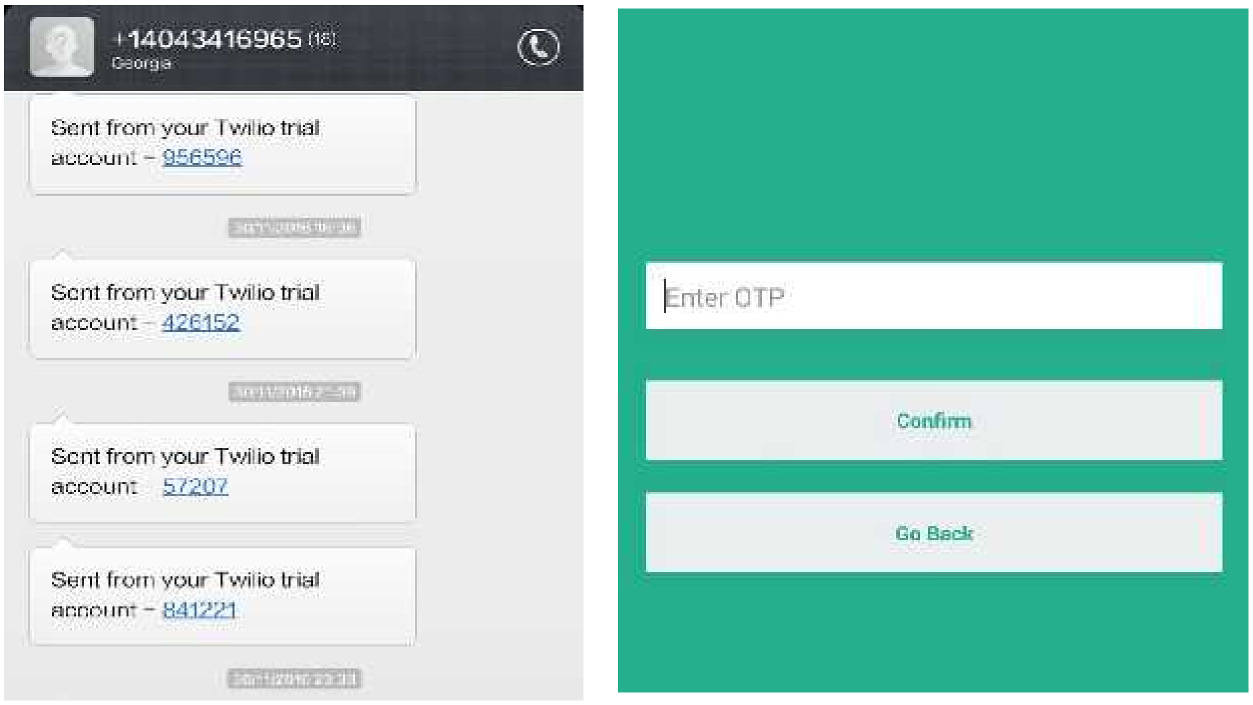
The user fills all the details and hits the pay button. Upon hitting this button a message is created: E(K_b^+ , [Card Number]), E(K_m^+ , [Product], Hash[Card Number],Hash[Product], Dual Signature), E(K_m^+ , [TimeStamp and *session_ctom*]) and sent to merchant server. The server:

- Decrypts TimeStamp, *session_ctom* and Product.
- Verifies TimeStamp, hash of Product and dual signature.
- Creates a session key *session_mtob* and a new TimeStamp.
- Stores *session_mtob* in Sessions table.
- Replaces the session key and TimeStamp in the original message, and forwards the new message to the Bank server.
- Generates a message: E(K_c^+ , [Verification Message, TimeStamp and *session_ctom*] and sends it to the client.

The bank server then:

- Decrypts TimeStamp, *session.n_mtob* and Card Number
- Verifies TimeStamp, hash of Card Number and dual signature.
- Accordingly generates a One Time Password (OTP) or a failure message.
- Stores the OTP in the OTP table.
- Uses Twilio's API to send the message to client.

The user after receiving the verification from merchant server is directed to the *OTPActivity* page. When he receives the OTP in message, he enters it and hits the send button. A message is created: E(K_b^+ , [Identifier, TimeStamp, OTP and Phone Number]) and sent to bank server.



+14043416965 (16)
Georgia

Sent from your Twilio trial account – 956596

Sent from your Twilio trial account – 426152

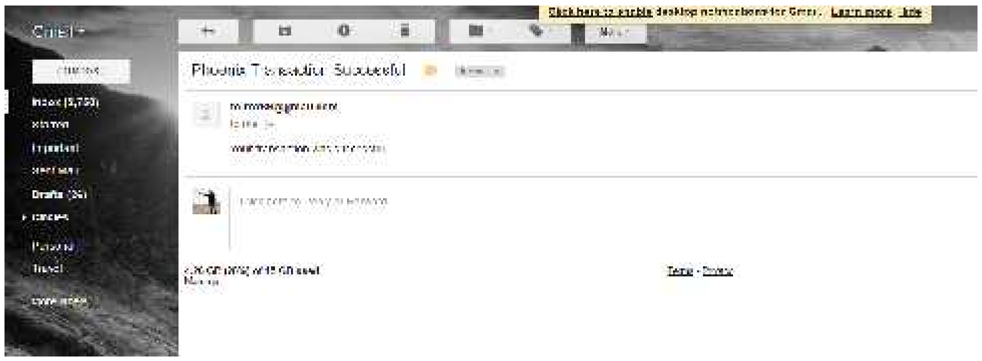
Sent from your Twilio trial account 57207

Sent from your Twilio trial account – 841221

Fig E.6 OTP Page

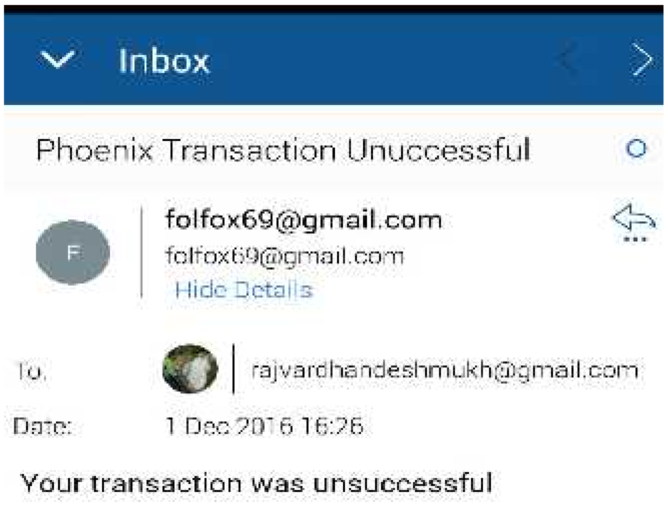
Bank on receiving this message verifies the time stamp and matches the OTP with the one it sent. It according creates verification message (successful/unsuccessful) and then generates a message: E(K_b^+ , [Identifier, TimeStamp, Session_MtoB and the message]) and sends it to merchant server.

Merchant server verifies the timestamp and sends client a successful/unsuccessful mail using the JavaMail API.



Phoenix Transaction Successful

Fig E.7 Successful Email Notification



Inbox

Phoenix Transaction Unuccessful

folfox69@gmail.com
folfox69@gmail.com
Hide Details

To: rajvardhandeshmukh@gmail.com

Date: 1 Dec 2016 16:26

Your transaction was unsuccessful

Fig E.8 Unsuccessful Email Notification

Possible improvements and extensions:

- Use of a better API than Twilio so that messages can be sent to everyone.
- Better threading in merchant and bank server to reduce code complexity.
- Use of larger keys.
- Improved storage of certificate and keys in android application.
- Creation of a credit card database, and matching the details when user enters it.
- Use of regular expression in application's input fields. (email and username length)