City University of Hong Kong
Department of Computer Science

BSCCS Final Year Project 2012-2013

12CS094

Data Transfer via Audio Channel on iPhones – an eWallet Application

(Volume 1 of 1)

Student Name	:	**Iris, LEUNG Wai Yin**
Student No.	:	**51604089**
Programme Code	:	**CS 4514**
Supervisor	:	**Dr. WONG, Shek Duncan**
Date	:	**15th April, 2013**
1st Reader	:	**Ms. MONG, Yu**
2nd Reader	:	**Prof. LI, Qing**

For Official Use Only

This page is left blank intentionally.

STUDENT FINAL YEAR PROJECT DECORATION

I have read the project guidelines and I understand the meaning of academic dishonesty, in particular plagiarism and collusion. I hereby declare that the work I submitted for my final year project, entitled:

Data Transfer via Audio Channel on iPhones – an eWallet Application

does not involve academic dishonesty. I give permission for my final year project work to be electronically scanned and if found to involve academic dishonesty, I am aware of the consequences as stated in the Project Guidelines.

Student Name: _____ Signature: _____

Student ID: _____ Date: _____

Abstract

The first concept of using credit cards for purchasing was described in 1887. The credit card system allows cardholders to pay for their goods and services without bringing cashes. Nowadays, we can see people using it every day in shops, supermarkets, and restaurant and so on. By swiping the cards through credit card machines, all the account details needed for payment can be read and sent to the credit card company servers for processing. It takes only a little time and it saves the time for money change back.

It seems to be convenient for using the cards. However, the swiping of credit cards runs a risk of leaking sensitive data of cardholder's accounts because of the advance of technology. It is not rare to hear from news nowadays that criminals install fake credit card machines in public to skim the card's data of careless users. Besides it, some of the sensitive data related to the card's accounts have already marked on the surface of the cards like the card account numbers, the card holder's names and the security codes. One is able to process online payment just by those data marked on the cards without any permission of the cardholders. As a result, the exposing and swiping of credit cards is no longer secure.

As to find the alternatives for the traditional credit card payment method, other technologies like the Radio Frequency Identification (RFID), Near Field Communication and "Square" card reader have been review. However, there are stills some security concerns on handling sensitive data. As a result, this project aims at develop a data transfer method through audio channels on iPhones. As for the data transmission method, an application will be developed in this project that allows iPhone users to process credit card payment securely. The Dual Tone Multi-Frequency technology (DMTF) would be the reference on the design of data transfer in this project. Also, some security algorithms such as hash, encryption and error detection codes would be applied in this project.

In the project, Xocde will be used as the main development tools for the IOS application. Then, the application will be deployed to different IOS devices for testing. This project is expected to cover the areas of information security, mobile computing, operating system and Cryptographic Algorithms and Protocols.

Table of Contents

1.Introduction

Nowadays, customers are common to use credit cards for payments in retail shops, restaurant, grocery shop, and so on. Customers hand in their cards to the cashiers for swiping when paying the bills. The account details stored in the magnetic stripes of the cards will then be read by the card readers used and sent to the credit card servers (HowStuffWorks.com, 2008). This is the most ordinary way of using credit cards that we see every day.

However, this kind of payment method comes with risks. For example, every time a customer hands in his/her card for a transaction, all the information marked on the card's surface like the credit card account number, the card expiry date and the security code may be seen by the cashier. Many may be unaware of the concern of exposing their cards. But for a person who has little experiences of online shopping, he or she may know that this information is just enough for proceeding payments on the Internet. Recently, there is certain news about the misuse of these details. For examples, a young model in UK was arrested of stealing credit card details from a man she met in a nightclub for purchasing online (Reilly, 2012). As a result, the swiping or exposing credit cards should be avoided as to protect the cardholders.

This project reviewed the existing credit card payment methods currently used. The aim of this project is to develop a new way for credit card payments that is suitable for the use of both merchants and the clients. With this new application developed, the cardholders do not need to show or swipe their credit card for transactions anymore. Both clients and merchants are able to process credit card payments at everywhere only by putting their mobile phones together. And then, the transaction details can then be transfer between each other's phones through voice signals securely. All the data stored in the application will be encrypted by AES256 to protect them from being accessed by unauthorized people. SHA256 and CRC16 will also be applied in the app to ensure its data confidentiality and integrity.

2.Literature Review

The aim of this section is to review the existing credit cards transaction methods that have been used for long in our society. The concerns of using these existing methods would be discussed. Then, we would introduce and discuss the new technologies that have been invented as the alternatives of the traditional machines used currently. After that, the use of different existing data transmission methods would be viewed. At Last, we would have a look on the security algorithms that are commonly used for maintaining data confidentiality and integrity.

2.1 Current transaction methods or data transfer technology

In this part, different current methods for banking transactions will be reviewed. Firstly, the using of credit cards machines will be introduced. Then, the new techniques such as radio frequency identification (RFID), near field communication and “Square” will be discussed.

2.1.1 Card Processing Machines

Using credit card processing machine for card transactions is the most common and traditional method for credit cards money transferrin our daily life. Pirraglia (2012) explained that the magnetic strip of the credit cards stored the information of the card account. And the data like cardholder's name, credit card number and security and verification codes would be read and sent to the processor’s server by swiping the card. There are three main types of card-processing machines that can be seen in the shops (Gotmerchant.com, 2012):

1. Standard Dial Up Terminal - The most commonly and traditional used terminal that works over the phone line.

2. IP Based Terminal - a newer type of terminal that is similar to standard dial up terminal, but it has a module inside that processes transactions over Internet connections instead of a phone

3. Wireless Credit Card Terminal - a wireless terminal that is very similar to a standard unit, but it works over a wireless network, such as CDMA or GPRS.

These machines are commonly used for our daily transactions. However, they run a risk for the leakage of cardholder's data. It is because the cardholders have to hand in their credit cards to the cashier or staff for card swiping when purchasing. This could be dangerous as the cards may sometimes be taken out of the cardholder's sights. Any illegal person could be able to take the cards away or swipe the cards with one's prepared machines for skimming and re-coping the card details for other purposes. So, the exposing and swiping of credit card should be avoided.

2.1.2 Radio frequency identification technology

The RFID credit cards provide one of the ways for credit card transactions that no swiping of credit cards is required. Dubinsky (2010) mentioned that these "Contactless" cards were embedded with a chip called radio frequency identification (RFID). With the use of radio wave, the information stored in the chip of the card could be read by just putting it on a payment terminal. As a result, the swiping of the card could be avoided.

However, this new technology raised another security risk. The data transferred through RFID connections are not encrypted and an RFID credit card reader can be found easily on the Internet like eBay. One may be able to scan or manipulate the data stored in theses cards by using a modified RFID card reader. Last year, there was news about the misuse of this technology. Lo (2012, March 08) reported that a family of five in Hong Kong were arrested of adding value to 61 octopus cards, totally HK$430,000, by using a home-made machine. As mentioned, the Hong Kong octopus card is one of the prime examples of using RFID technology for money transfer [RFID Journal, 2003]. As result, the security of RFID technology may not be reliable enough to be included in the implementation of this project.

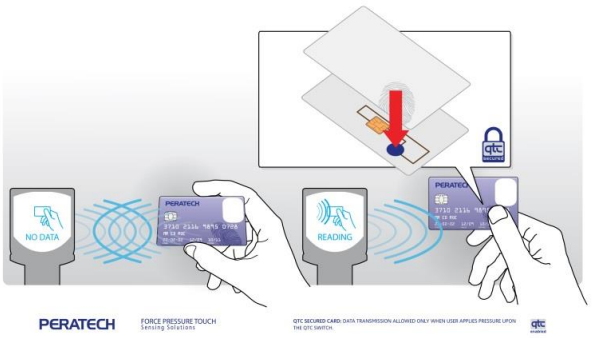


Figure 2.1.2 The reading of RFID credit cards

2.1.3 Near field communication technology

Near field communication (NFC) is another new technology that transfer data and files between smartphones by using radio frequencies. This technology is very similar to the use of RFID chip. A NFC chip is usually embedded in the mobile devices. By putting two mobile devices close together, data can be transferred from one to the other. The valid distance for this kind of communication is usually no more than a few centimeters. However, Strickland (2012) said that this distance limitation was possible to be by some specific applications like the antenna and receiver. It is because the NFC technology is using radio signals for communications that can be listened, transmitted or amplified by antenna. One can also disrupted the connections by broadcasting other radio signals. Moreover, the reader of NFC, same as those of RFID, can also be found easily as many smart phones are equipped with a NFC chips. As a result, the NFC technology would also not be considered.

Picture 2.1.3 NFC communication

2.1.4 Transaction via "Square" reader

"Square" is a new technology that has been raised recently. It allows the merchants to process credit cards transactions everywhere by using their smartphones. The "Square" card reader (Picture 2.1.4) is developed by Square Inc. By installing the square card reader, merchants are able to swipe credit cards for transactions through their own cell phones that are equipped with Wi-Fi or 3G accesses. However, CARR (2011) indicated that Electronic payment firm VeriFone found a big security problem of the square's hardware. It discovered that the square card reader did not encrypt the card data itself when the credit card was swiped. Instead, the

data encryption was done by the square's app. It means that the data can be copied easily by using a fake app with using the square card reader.

Picture 2.1.4 Square card reader

The square card reader requires the swiping of credit cards and it seems that the reader is critical to leakage of card data. But it is a good idea to have an application in smartphones for credit card transactions. So we would take it as reference materials for developing an iPhone app of card payments in this project.

2.2 Data Transmission methods

During the credit card payments, the transmission of data is the key function and an essential part that should be considered in the first place. An unreliable data transfer would lead to the missing, misrepresenting or even worse the leaking of sensitive data that could be valuable for unauthorized people. As a result, different transmission methods would be reviewed and discussed in this part. Firstly, the representation of text contents like the Binary codes and the ASCII code is reviewed. Then, the traditional methods used for the telephone systems for signals transfer would be reviewed.

2.2.1 Binary code

Binary code is the most common machine code for representing texts and computer processor instructions in computers. Computers communicate with each other by using binary digits 0 and 1 that can be presented as the flow of electricity. When the electricity is on, it is regarded as 1. And it is 0 when the electricity is off. For example, digit “9” is translated to “1001” in base 2 when it is transmitted through network (see table 2.5.1).

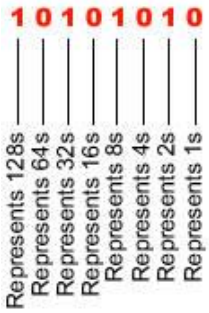


Figure 2.2.1 Binary representation

1	2	3	4
0001	0010	0011	0100
5	6	7	8
0101	0110	0111	1000
9	10	11	12
1001	1010	1011	1100

Table 2.2.1 Binary representations for the digits

However, this kind of text representation requires a lot of bits for each digit or characters. For examples, if we want to transfer a large number like “123” in decimal base, i.e. $(123)_{10}$, the binary representation of it would be $(1111011)_{2}$. It means that 7 bits have to be used for showing the number “123”. As a result, few bits would be used for large number that would slow down the process of data transmission. And it increases the complexity of sending data besides the digits as different character and digits are represented by different numbers of bit.

2.2.2 American Standard Code for Information Interchange (ASCII code)

As the binary code is only a mathematic method for the transmission of digit, it requires few bits to represent large numbers and digital information that contains symbols and alphabetic characters. So, the American Standard Code for Information Interchange (ASCII Code) is introduced. In the ASCII table (see table 2.2.2), two hexadecimal digits are used to represent each digit, character or symbol. The two hexadecimal digits will them be translated to a set of 8 – bits binary code for the transmission of data. The number of bits for each ASCII character is fixed.

ASCII Code Chart

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

Table 2.2.2 ASCII Table

This representation seems to be more feasible for the use of this project. It is because the card transactions require the transfer of texts and other symbols besides the digits.

2.2.3 Dual tone multi frequency (DTMF)

Rouse (2005) introduced that the DMTF is a technology that the phone companies used to generate signals for dialing numbers. Each key on the phones is assigned with two frequencies. One is from a lower frequency group and the other is form a higher frequency group. The signal delivered is a combination of these two frequencies as to form a unique sound that cannot be imitated easily. See table 2.5.3, these are the frequencies used by phones signals. Sound frequencies in 697 Hz and 1209Hz are used to represent digit “1”.

	1209 Hz	1336 Hz	1477 Hz
697 Hz	1	2	3
770 Hz	4	5	6
852 Hz	7	8	9
941 Hz	*	0	#

Table 2.2.3 Frequency table of DMTF

This way of transferring signals is based on the generations and recognitions of sound frequencies. It provides an idea for transmitting signals in this project.

2.3 Security Algorithms for Data Protection methods

As for an iPhone app for credit card payments, the management of sensitive data like passwords, card account numbers and card's security code ought to be a concern. The leakage of these data may cause a financial loss to the users. As to protect the data from being exposed, some famous security methods are reviews for protecting data confidentiality integrity.

2.3.1 Secure Hash Algorithm (SHA)

The Secure Hash Algorithm is a hashing algorithm that is published by the National Institute of Standards and Technology (NIST) as a U.S. Federal Information Processing Standard (FIPS). Inman (2013) explained that it was a one-way function. One was not feasible to recover the original data with its corresponding hashed values. Secure hash algorithms are often used in combination with other algorithms to authenticate messages, including digital signatures. SHA is an iterated function that adds in padding, initiator to the data and compresses them literately. (See figure 2.3.1)

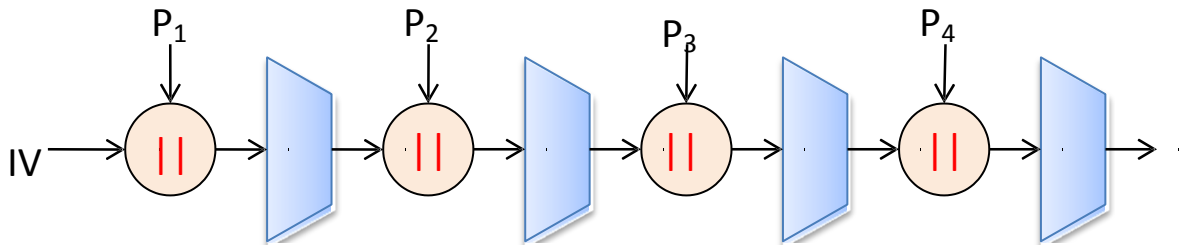


Figure 2.3.1

There are different versions of SHA:

SHA-0, the original version of the 160-bit hash function, was quickly replaced by version SHA-1 after a significant weakness was found.

SHA-1, a 160-bit hash function that is commonly used in Internet protocols and security tools. The National Security Agency (NSA) designed it as a component of Digital Signature Algorithm. But it is no longer used after 2010.

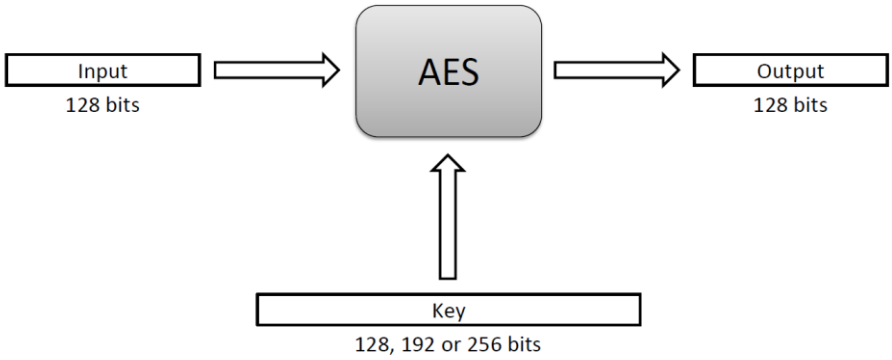
SHA-2, a group of similar hash functions, SHA-256 and SHA-512 are known functions in this group. Each member in the group has different block size. SHA-256 has a block size of 256 bits (32 bytes) and SHA-512 has a size of 512 bits (64 bytes.)

SHA-3, the latest hash function that was chosen in 2012 after a public competition among non-NSA designers. It is different significantly from the rest of the SHA family as it does not follow the bas algorithm of the previous versions.

2.3.2 Advanced Encryption Standard (AES)

The Advanced Encryption Standard (AES), which is published by the U.S. National Institute of Standards and Technology (NIST) in 2001, is a specification for the electronic data encryption (The National Institute of Standards and Technology (NIST), 2001). AES is a symmetric-key algorithm that uses the same key for both encrypting and decrypting data.

AES is also a block cipher that operates on 128-bit blocks. It is allowed the encryptions with different key sizes that are in 128, 192, or 256 bits long. They are known as AES-128, AES-192, and AES-256.



The numbers of repetitions of transformation rounds in AES depends on the key sized used:

- 10 cycles for 128-bit keys. (See figure 2.3.2)
- 12 cycles for 192-bit keys.
- 14 cycles for 256-bit keys.

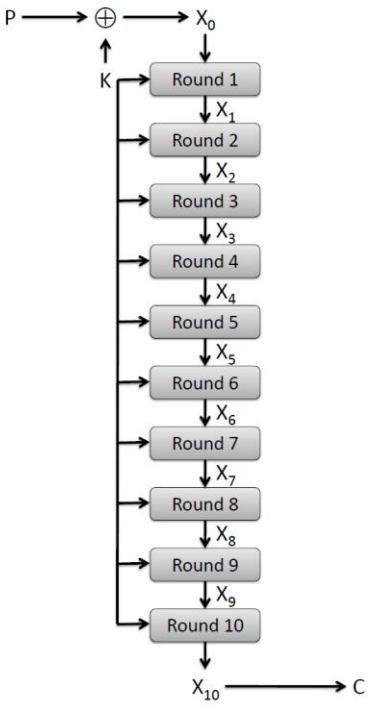


Figure 2.3.2 Round structure AES-128

Each round of AES composes four basic steps as follows:

1. SubBytes step: an S-box substitution step

2. ShiftRows step: a permutation step

3. MixColumns step: a matrix multiplication step

4. AddRoundKey step: an XOR step with a round key derived from the 128-bit encryption key

The longer the keys used, the more cycles of transformation rounds performed during the data encryption and decryption.

2.3.3 Cyclic Redundancy Check (CRC)

The Cyclic Redundancy Check is an error-detecting code that is commonly used when transmitting data through networks. It can also be applied for assuring the integrity of data delivered.

The sender can generate a short check value by inputting the data to the CRC algorithm before sending the data. The check value will then be attached to the data sent. After the receiver gets the data, she/he can calculate the check value by entering the data received into CRC algorithm again. By comparing self-calculated check value with the received one, the receiver is able to see of the data are changed.

There are 4 most common CRC polynomial lengths:

- 9 bits (CRC-8)
- 17 bits (CRC-16)
- 33 bits (CRC-32)
- 65 bits (CRC-64)

Williams (1993) explained that the choosing of CRC polynomial depends on the length of the data, the features of error protection, and the resources for calculating the CRC.

This research aims to develop an alternative method and technique for credit card payments. As reviewed, the transactions of using traditional credit card machines requires cardholders to hand out their cards for swiping. It drives security risk when the cards are taken away from the cardholder's sight. Then, we introduced the new RFID technology that was designed for avoiding card swiping, but it is found that the RFID credit cards data can also be easily skimmed as the RFID readers can be easily accessed. The implementation of NFC is very similar to that of RFID. As a result, the use of NFC technology meets the same problems of skimming sensitive data. In the study, we also mentioned a new technology called square up for mobile users. However, the square card reader also obtained a risk, as the reader itself does not encrypt the card data.

As a result, a new method for using voice as the medium for data transmission is needed to be developed with the use of the different data transmission methods like the Dual Tone Multi Frequency (DTMF). As to maintain data confidentiality and integrity, we have also reviewed some security algorithms like the Secure Hash Algorithm (SHA), Advanced Encryption Standard (AES) and Cyclic Redundancy Check (CRC).

3.AIMS AND OBJECTIVES

This section presents the aims, objectives, functional and non-functional requirements for the mobile application developed in this project. The application allows iPhone users to process credit card payments by transferring sensitive data via voice signals securely between each other.

3.1 Aims and Objectives

From the review of the above researches, the aim of this project is to provide an IOS application for transferring card data for credit card payments securely without the swiping of credit cards. The application infrastructure will be designed as a combination model for both merchant and client sides. The merchant side is able to request for payments through sending signals to the client's phones. On the other hand, the client side is able to collect the payment detail by listening to the signals sent from the merchant side.

This project is aimed at developing an IOS application called eWallet. Its basic function is to allow both merchant and client to make credit card payments that no swiping of cards is required. Additionally, the merchants and clients are able to conduct the payment processes at everywhere. To achieve the results studied in the previous section and to enhance the application, this section lists out the proposed application requirements (both functional and non-functional) and user requirements.

3.2 Functional Requirements

Here specify the functional requirements definition for the main user interface, the production of signals, the collection of voice signals, and the user authentication for the eWallet payment.

3.2.1 Main user menu functional requirements

- The main menu should be retrieved automatically once the application is triggered.
- The main user menu should provide clear buttons for leading users to tasks.

- The application should allow the users to return to the menu after finishing the tasks

3.2.2 The production of voices signals functional requirements

- The application should provide an easy interface for inputting data before producing signals.
- The application should be able to convert the user's input into voice signals.
- The production and delivery of voice signals should be completed in short time to avoid the long waiting time for users.
- The signals produced should be out of human hearing range.
- User should be able to produce the same signal again as to ensure that the entire signals are sent correctly.
- User should not be allowed to continue the next payment processed if the delivery of signals is interrupted.
- The signals produced should be encrypted to avoid the eavesdropping of data.

3.2.3 The collection of voice signal functional requirements

- The application should be able to distinguish the voice signals from background sounds.
- The application should be able to recognize and decrypt the signals with different frequencies accurately.
- The application should display the data collected to the interface.
- User should be able to re-collect the signals until the data received are error –free.
- The application should be able to handle the duplicated signals.
- The data received should be checked as error-free automatically
- The users should be able to check the data manually

3.2.4 The user authentication functional requirements

- The application should provide user authentication for protecting user's accounts.
- User should input password before entering the merchant page and client page.

3.2.5 The secure data storage functional requirements

- The password stored should be hashed or encrypted as to avoid it from being seen directly
- All the data of the card accounts should be encrypted
- The file for storage should be hidden

3.3 Non- functional Requirements

This part specifies the non-functional requirements, which are IOS application performance and the interface design for the eWallet Application.

3.3.1 Interface requirements

- The application should provide a user-friendly interface.
  - The texts on the buttons should be clearly defined and shown.
  - The buttons should be significant.
  - The button's size should be large enough.
- There should be return buttons for user to return to the previous page.
- The background of the interface can be customized.

3.3.2 The account management requirements

- The users should be able to register more than one account.
- The users should be able to delete the unused accounts
- The users should be able to choose any account they registered for transactions.

3.3.3 The password management requirements

- User should re-enter the password when initializing or resetting the password as to avoid the wrong setting of password due to the typing mistakes
- The users should be able to reset their passwords
- The users should enter the old password for authentications before resetting the passwords.

3.3.4 Transaction Records requirements

- The transaction date like card account numbers, the card expiry date and the amounts to pay should be stored as transaction records for tracing.
- The users should be able to retrieve the records after login
- The users should be able to delete the records

3.3.5 Session Invalidation requirement

- The Session should be stored after users logging in successfully.
- The session should be logged out after 15 minutes of user inactivity.
- The users should be able to logout sessions manually

3.4 Users Requirements of application

All the user requirements of the eWallet application are listed as follows.

3.4.1 Merchant Page

1. Request payment	
Description	Merchant users can generate the payment signals to clients.
Scenario	Merchant users enter the amount for payment and press the button "Request Payment".
Expected Result	The signals of the payment request are generated and delivered.
2. Collect client credit card details	
Description	Merchant users collect client's credit card details for transactions.
Scenario	Merchant users collect the signals generated from clients.
Expected Result	The necessary client's accounts details will be shown on the merchant's application interface.
3. Confirm payment transaction	
Description	Merchant users confirm the details and then send the payment requests to the credit card servers.
Scenario	Merchant users check and confirm all the information involved and press "confirm" button.
Expected Result	The details will be sent to the credit card server.
4. Send receipts to client users	
Description	Merchant users acknowledge the clients about the completion of transaction.
Scenario	The confirmation page sends the confirmation signals automatically.
Expected Result	The signals for the receipts are generated and delivered.

3.4.2 Merchant Account Page

1. Retrieve merchant accounts	
Description	Merchant users retrieve the accounts added.
Scenario	Merchant users enter the page of "Merchant Accounts".
Expected Result	The added accounts are listed in the page.

2. View account details	
Description	Merchant users view the details of their accounts.
Scenario	Merchant users points one of the rows in the list under “Merchant Accounts”.
Expected Result	The account details will be shown on the user interface of the application.
3. Add new merchant accounts	
Description	Merchant users add new merchant accounts.
Scenario	Merchant users click “Add” button and enter the necessary data to the text fields in the interface, then press the “Save” button to store the account.
Expected Result	The account will be saved and it can be seen on the list of “Merchant Accounts”.
4. Delete merchant accounts	
Description	Merchant users are able to delete the unused accounts.
Scenario	Merchant users press “Delete” button after swiping a selected row in the list under “Merchant Accounts”.
Expected Result	The row selected is deleted and removed from the list. Its data stored in the files is also deleted.

3.4.3 Merchant History Page

1. Retrieve merchant transaction records	
Description	Merchant users retrieve the transaction records.
Scenario	Merchant users enter the page of “Transaction History”.
Expected Result	The transaction records are listed in the page.
2. View transaction details	
Description	Merchant users view the details of their transactions.
Scenario	Merchant users choose one of the rows in the list under “Transaction History”.
Expected Result	The transaction details will be shown on the user interface.
3. Delete transaction records	
Description	Merchant users are able to delete the transaction records.
Scenario	Merchant users press “Delete” button after swiping a selected row in the list under “Transaction History”.

Expected Result	The row selected is deleted and removed from the list. Its data stored in the files is also deleted.

3.4.4 Client Page

1. Collect details payment	
Description	Client users collect details of payment from merchant's phones.
Scenario	Client users press "Pay" in the main menu to enter the page of "Pay". Then, the page will collect the signals automatically.
Expected Result	The signals for payment are received and the payment details are shown on the interface.
2. Send credit card details to merchant user	
Description	Client users send credit card details to merchant users for payments.
Scenario	Client users check the detail, choose the card accounts and press the button to continue the payments.
Expected Result	The signals for the client user's card account are generated.
3. Receive receipts from merchant user	
Description	Client users receive acknowledges of successful transaction.
Scenario	Client users collect the signals of successful payment.
Expected Result	The signals for the receipts are received and shown on the interface.

3.4.5 Client Card Directory Page

1. Retrieve card accounts	
Description	Client users retrieve the card accounts added.
Scenario	Client users enter the page of "My Wallet".
Expected Result	The added accounts are listed in the page "My Wallet".
2. View account details	
Description	Client users view the details of their card accounts.
Scenario	Client users choose one of the rows in the list under "My wallet".
Expected Result	The card account details will be shown on the user interface of the application.
3. Add new card accounts	
Description	Merchant users add new card accounts.

Scenario	Client users click “Add” button and enter the necessary data to the form provided, then press the “Save” button to store the accounts.
Expected Result	The account it will be saved and it can be seen on the list of “My Wallet”.
4. Delete card accounts	
Description	Client users are able to delete the unused card accounts.
Scenario	Client users press “Delete” button after swiping a selected row in the list under “My Wallet”.
Expected Result	The row selected is deleted and removed from the list. The data related in the file will also be deleted.

3.4.6 Client History Page

1. Retrieve wallet transaction records	
Description	Client users retrieve their card transaction records.
Scenario	Client users enter the page “History” under “My Wallet”.
Expected Result	The transaction records are listed in the page.
2. View transaction details	
Description	Client users view the details of their transaction records.
Scenario	Client users choose one of the rows in the list under “History”.
Expected Result	The transaction details will be shown on the user interface.
3. Delete transaction records	
Description	Client users are able to delete the transaction records.
Scenario	Client users press “Delete” button after swiping a selected row in the list under “History”.
Expected Result	The row selected is deleted and removed from the list. The related data stored in the files will also be deleted.

4.System overview and design

In this project, the application is designed for users to make credit card payments through iPhones. The speakers and microphones of the iPhones can be utilized for developing our IOS app for card payments. So, we decided to use voice signals as the media for data transfer. As reviewed in 2.2.3, the dual tone multi frequency (DTMF) technology has been used long for sending signals between phones. The long history of using DTMF technology proofs the feasibility of using voices for data transfer. As a result, the use of DTMF will be considered as a reference for generating and transferring signals between the iPhones. The security for data storage should also be a big concern for this app. So, the following parts will show the design on how the application hash and encrypt user's data as to protect data confidentiality. The use of Cyclic Redundancy Code for detecting error and the design on limiting password setting will also be mentioned in the later parts.

4.1 System Overview

The application is designed as a peer-to-peer architecture. Merchants and clients are regards as peers. Merchants and clients exchange sufficient data with each other through audio channels (See figure 4.1). Merchants send transaction requests to card server only after collecting all information required through Wi- Fi or 3G. This project mainly focuses on the data communication between the iPhones rather than sending request to credit card servers. So, the part of sending data to the card servers will not be included in this project.

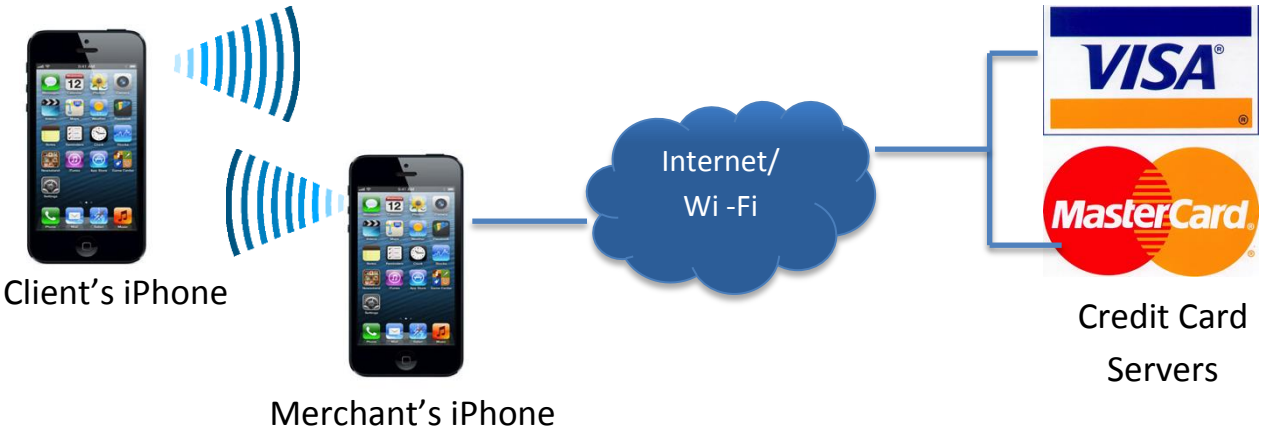


Figure 4.1. System overview

4.2 The sound frequencies used for voice signals

As the speakers and microphones equipped in iPhones play an important role in generating and collecting the voice signals, it is necessary to investigate into the range of frequencies that are feasibly supported by them. We will take the frequency ranges as references for designing the frequencies to be assigned to each character. And the results for the findings on the frequency ranges of iPhone speakers and microphones are shown below.

The frequency responses for iPhones are limited (Ben, 2010). The 2 figures below show the frequency ranges that can be used by the iPhones. (See Figure 4.1a and Figure 4.2b)

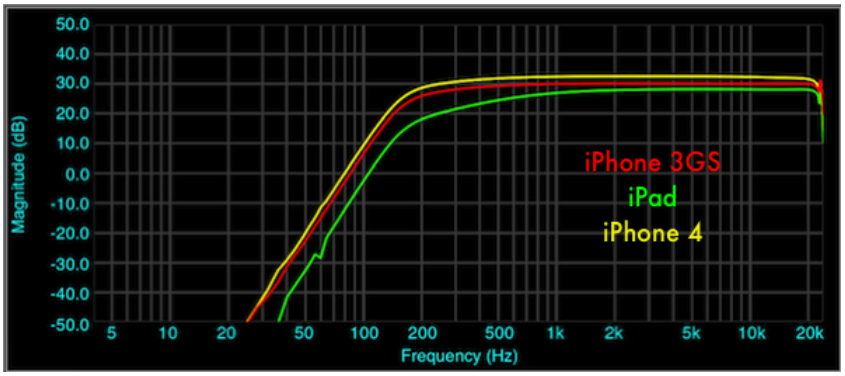


Figure 4.2a The iPhone 4 Headset Input Frequency Response

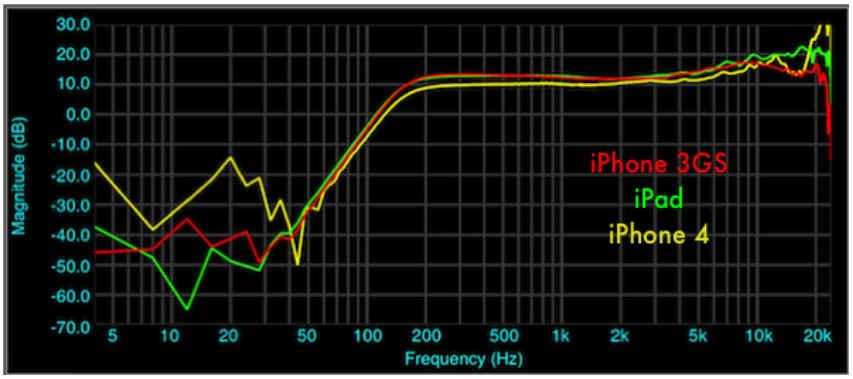


Figure 4.2b iPhone 4 Built-in Microphone Frequency Response

From the figures above, it indicates that the frequencies which both the handsets and microphones response most are from 150 Hz to 20k Hz. It means that this project is only feasible to use voice signal in this range. The human hearing frequency range is between 20Hz and 20 kHz. (See Diagram 4.2a)

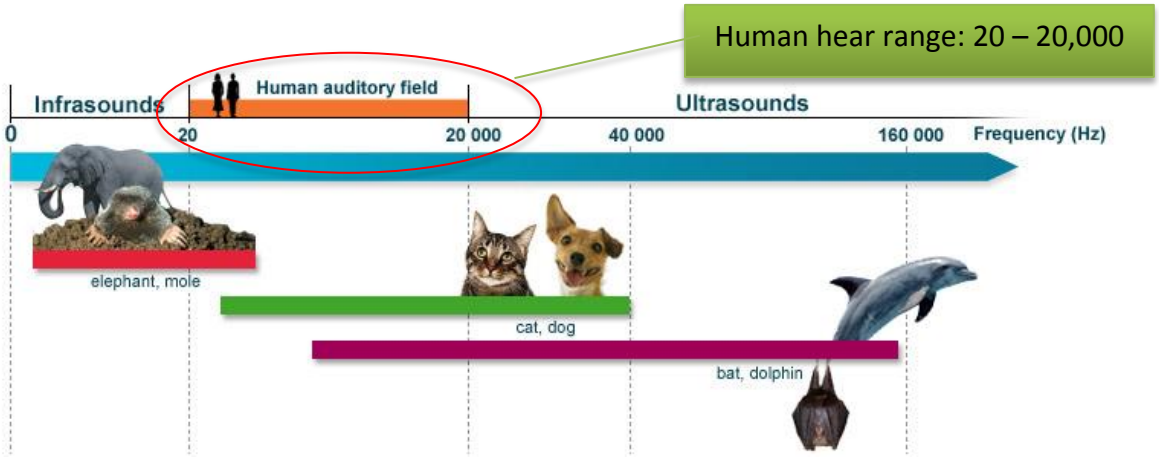


Diagram 4.2a Human hear frequency range

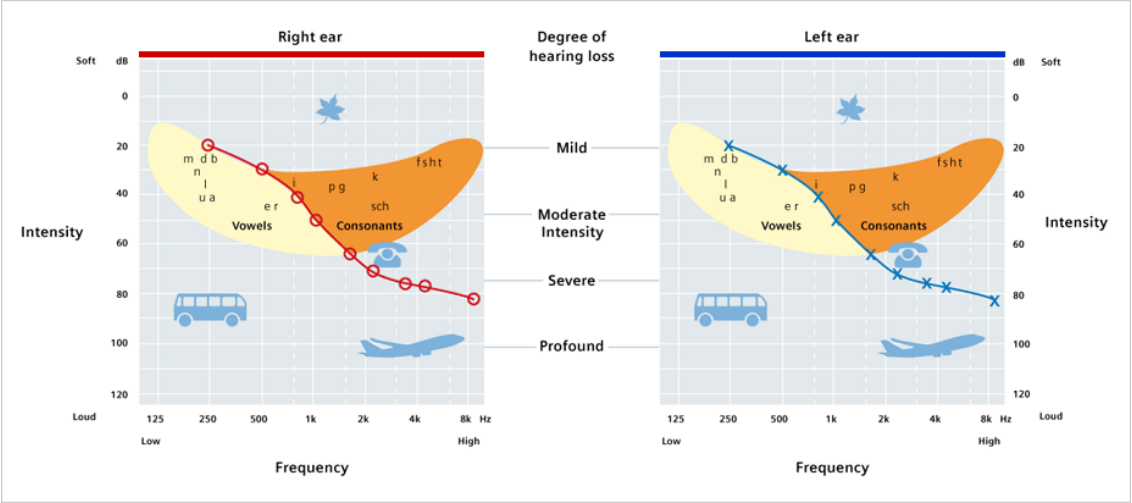


Diagram 4.2b Human speech frequency range

The frequencies of ordinary human speech communication are between 200 and 8,000 Hz (Diagram 4.2b) and the human ears are most sensitive to frequencies around 1,000-3,500 Hz. Due to the limitation of the hardware and as to avoid the disturbance of signals from human speech, we should prefer frequencies that are higher than 8000 Hz and less than 20k Hz.

4.3 Preliminary assignment for the DTMF table

As for the representation of the texts, we chose the ASCII encoding. It is because the ASCII code encoding method requires fewer bits for the complicated contents. Additionally, the DTMF technology is chosen as the method for signal transfer. It is because the DTMF has been used for the telecommunication system in centuries, its reliability has been fully proofed. As a result, it is proposed to have a combination use of the ASCII encoding and DTMF methods for the assignment of signals to each characters. The following shows the modified DTMF table with the combination of ASCII table.

	1209 Hz	1336 Hz	1477 Hz	1633 Hz
697 Hz	1	2	3	A
770 Hz	4	5	6	B
852 Hz	7	8	9	C
941 Hz	*	0	#	D

ASCII Code Chart

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

Hz	19000	19100	19200	19300	19400	19500	19600	19700	19800	19900
17000	0	1	2	3	4	5	6	7	8	9
17100	A	B	C	D	E	F	G	H	I	J
17200	K	L	M	N	O	P	Q	R	S	T
17300	U	V	W	X	Y	Z	a	b	c	d
17400	e	f	g	h	i	j	k	l	m	n
17500	o	p	q	r	s	t	u	v	w	x
17600	y	z	#	$	%	‘	*	+	-	=
17700	,	.	/	_	(	)	“	!	{	}

Table 4.3 The new DTMF table of the new app

The table above (see Table 4.3) shows that each text character is assigned with two frequencies. For examples, “1” is assigned with frequencies of 17000Hz and 19000Hz. However, it is only a preliminary design to show how to assign the frequencies for different digits, symbols and characters. We will further modify it after that.

4.4 Secure storage of sensitive data

As to prevent the data from being accessed by unauthorized people, the following methods will be applied in this project. They are the hash algorithm, data encryption method and the concerns on password setting. However, this part contains the concepts on how to use these methods only, the details on the implementation and practical use of them will be seen in section 5.4 (Programming Techniques Related to Security Issues) later.

4.4.1 How to generate hash value for the passwords

As to protect the passwords stored from being seen by others directly, this application is designed to store the password hash value instead. As reviewed, the secure hash algorithm (SHA) is a one-way function. The original data are supposed to be infeasible for recovering even one gets the hash values. So, It is suitable to convert the password into a pile of hexadecimal that cannot be reused by others.

In this protect, SHA-256 is chosen for generating the hash values for passwords. It is because the older version, SHA-1, is no longer used after 2010. SHA-1 was found to be insecure on the collision problems (Schneier, 2005). As a result, SHA-256 is applied and the hash values in 256 bits (32 bytes) will be generated. We can see the flow of the passwords using SHA -256 for password setting in figure 4.4.1a.

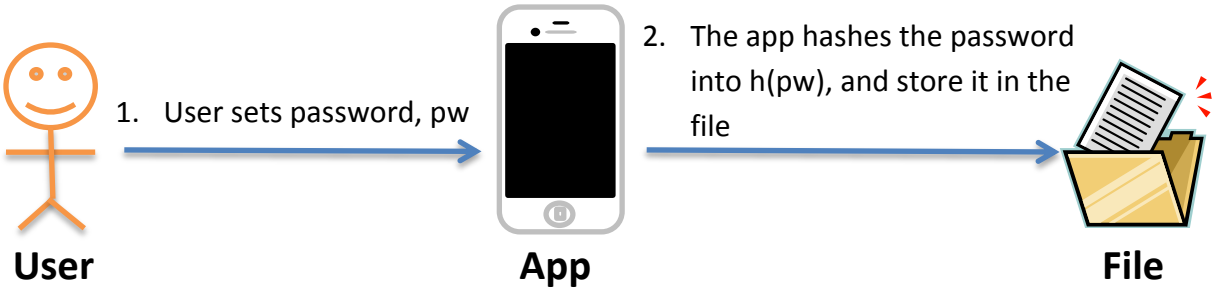


Figure 4.4.1a User sets password for initialization

From the figure above, the users enter a password into the app at the first time he/she use it. Then, the app will generate the hash value of the password and store it into a hidden file in the system. (See table 4.4.1b)

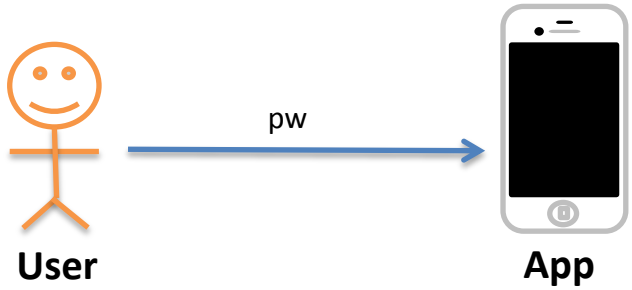
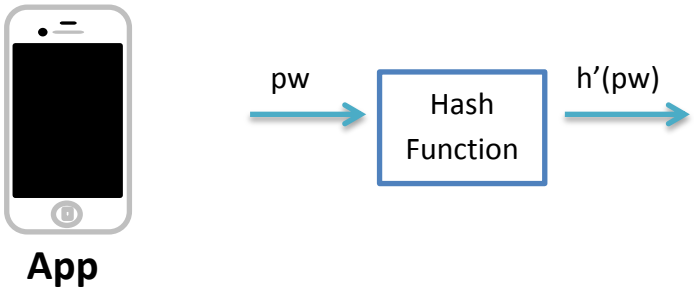
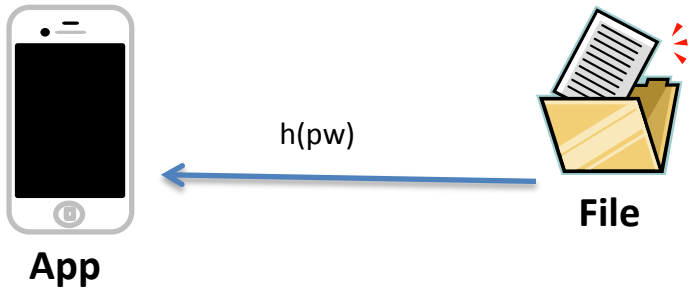
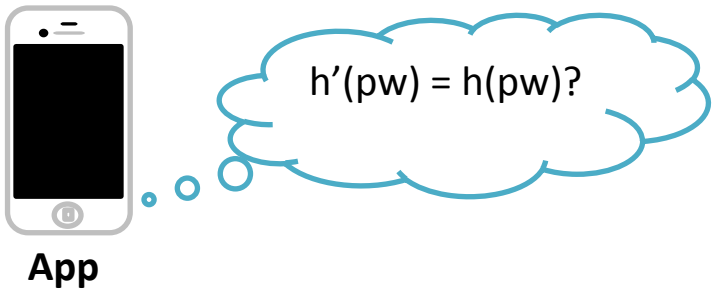
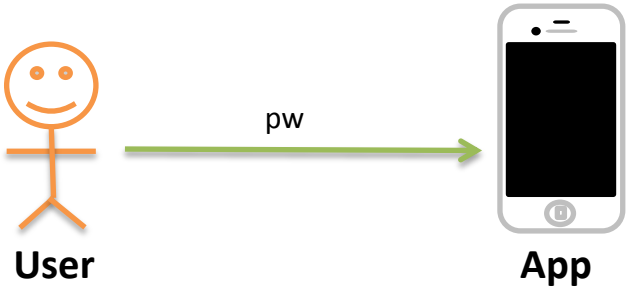
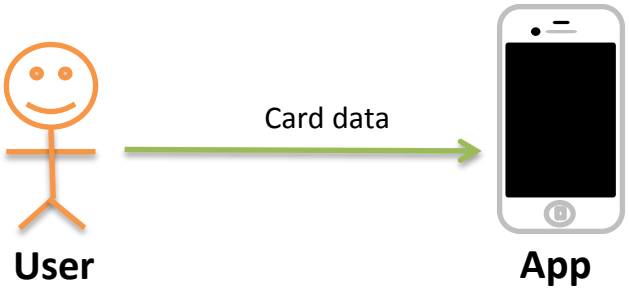
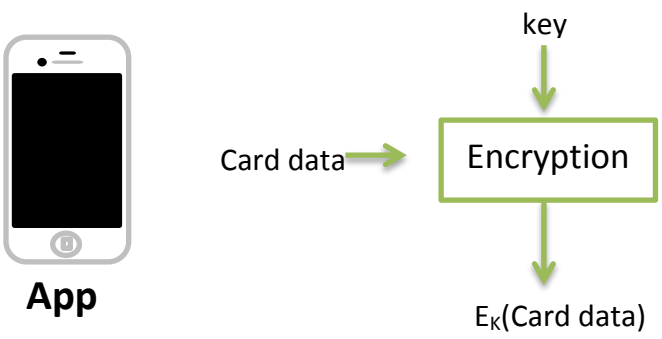
Steps:	Description
1. **User logs in with the password set before, pw**	
2. **The app hashes the password into h' (pw)**	
3. **The app loads the stored h(pw)**	
4. **The app compares h'(pw) with h(pw)**	

Table 4.4.1b User logs in with password

4.4.2 How to encrypt and decrypt the data used

As to protect the sensitive account data, all the data used by the app should be encrypted including the transaction histories. As reviewed in section 4, AES-256 is selected as the algorithm for encrypting and decrypting the data used. It is because AES is a symmetric key encryption method that encrypts data more efficiently than RSA, which is an asymmetric key encryption method.

In the application, a random key in 128 bits is generated once the first account data is added. Then, the account data will be encrypted with the random key. The encrypted data will then be stored in a hidden file in the system. Finally, the key used for encrypting data will be encrypted by the user's password and stored in another hidden file in the system. (See table 4.4.2a)

Steps:	Description
1. User logs in with password, pw	
2. User enters the card account data into the app and presses "Save" button	
3. The app generates a random key (K) and encrypts the data using it.	

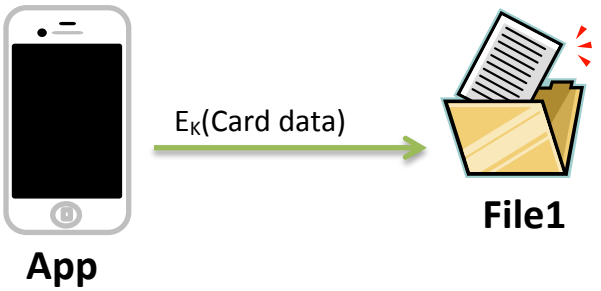
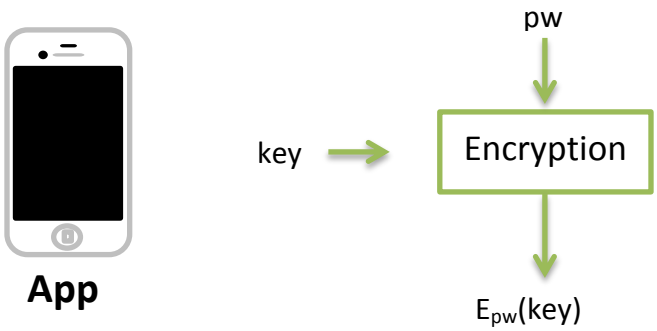
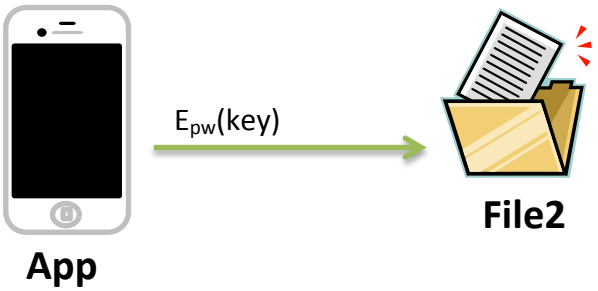
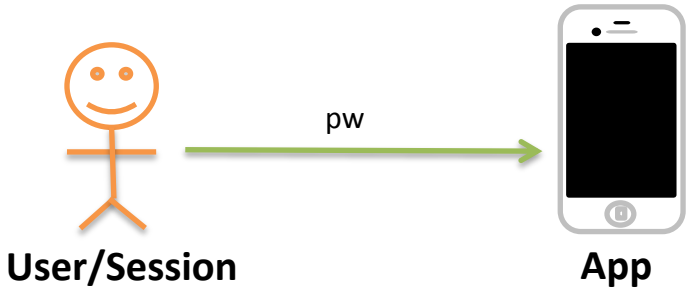
4. The app stores the encrypted data, EK(Card data) into file	
5. The app encrypts the random key (key) with user's login password, pw	
6. The app stores the encrypted key, Epw(key) into a hidden file	

Table 4.4.2a Encrypting data in the app

After encrypting the data, it is important that we can restore and reload them for later use. As a result, the app should be able to use the same key for decryptions. The app will use the user's password to decrypt the key used for decryption first. Then it uses the key for decrypting the account data. (See table 4.4.2b)

Steps:	Description
1. The app gets the user's login password, pw	

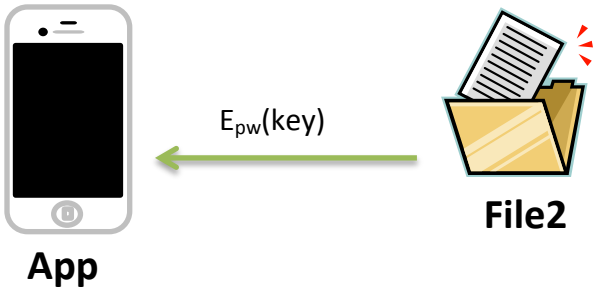
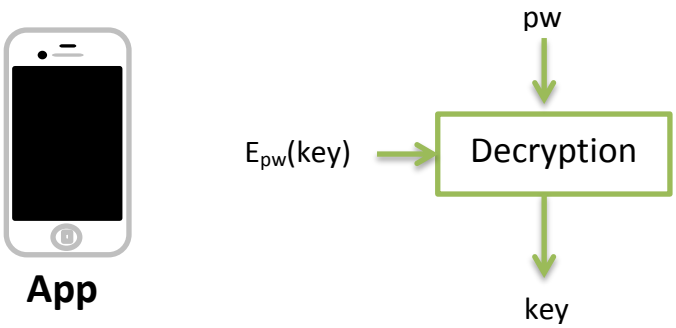
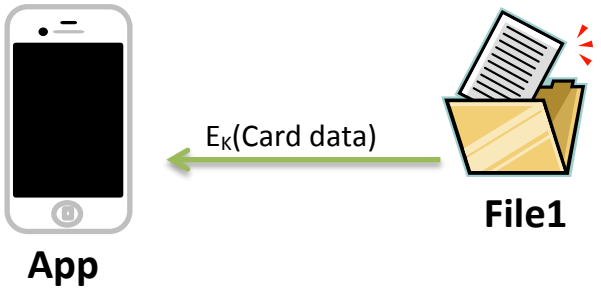
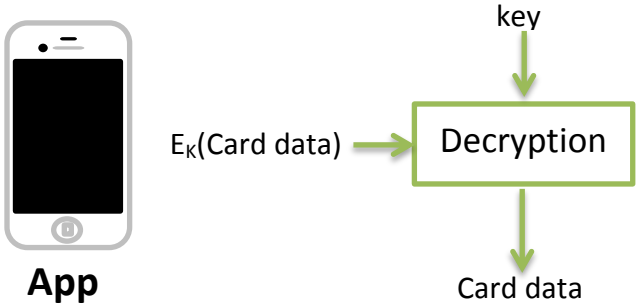
2. User requests to show the card data	
3. The app loads the encrypted key used, Epw(key)	
4. The app decrypt the encrypted key, Epw(key) that is use for encrypting card data	
5. The app loads the encrypted card data, E_K(Card data) from files	
6. The app decrypts the encrypted data, E_K(Card data) with the random key, key.	

Table 4.4.2a Decrypting data in the app

4.4.3 How to use Cyclic Redundancy Code (CRC) for data integrity

It is common that some signals maybe lost when delivering due to the interference of background noises. As to ensure that the data transferred are error-free and has not been changed, the Cyclic Redundancy Code (CRC) is used. In this project CRC-16 is selected. Before sending the data signals, the app will calculate the CRC value of the data. (See Figure 4.4.3a)

Figure 4.4.3a Generating CRC value

Then, the app of the sender embeds the CRC value at the end of the data to be sent. (See Figure 4.4.3b)

Figure 4.4.3b Embedding CRC value to data

After the receiver's app gets the data, it divides the data from the CRC value. (See Figure 4.4.3c)

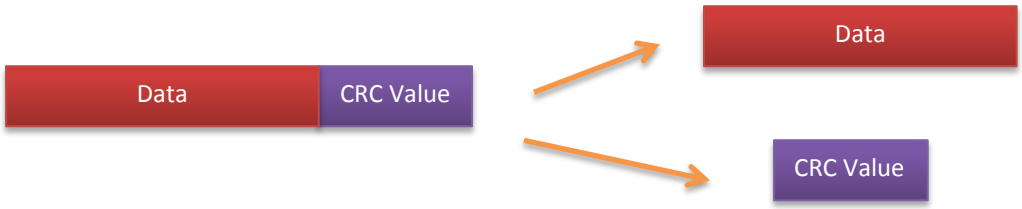


Figure 4.4.3b Dividing data and CRC value

The receiver's app calculates the CRC value itself with the data obtained in figure 4.4.3c

Figure 4.4.3c Receiver generates new CRC value with the data

Finally, the receiver's app compares the new CRC value with the received CRC value (See figure 4.4.3d).

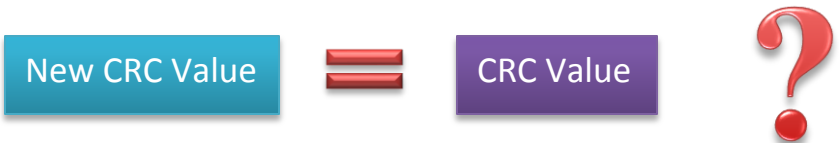


Figure 4.4.3d Receiver compares new CRC value with the received CRC values

If the CRC values match, it means that the signals received are correct. Then, the receiver's app will show the data received on its user interface.

On the contrary, if the CRC values do not match, the user interface of the receiver's app will show an error message. So, the receiver can press the "Retry" button and listen the signals again.

4.4.4 Designs on password settings

Password setting and resetting are important parts for password management. The security of the authentication system depends on the length of the password. Also, the users should change their passwords periodically as to prevent the leakage of them.

4.4.4.1 The limitation to the password length

The security level of the system depends on the setting of password. If the password is too short, it can be guessed or discovered quickly by brute-force attack. As to avoid null password, the length of password set should be longer than 1 character.

4.4.4.2 The reset of password

As to ensure that the password will not be exposed through shoulder check or other methods, the users should reset the password regularly. As a result, our app should provide an user-friendly interface for resetting passwords.

But it is known that users often input wrong password mistakenly, the passwords they set may not be correctly so that they may not be able to login again next time. So, the app should require the users to re- enter the passwords when resetting them. It is for the users to confirm the passwords they input.

4.5 User Interface designs in the application

As to be user-friendly, the designs and flows of the payment procedures should not be missed. The procedures of card payment in the new app can see step by step as bellows:

4.5.1 Account Initializing

As to protect our user's data, both merchant and client users are required to set up a password at the first time they use the app. Once a user has entered the page of "Merchant" or "My Wallet", a small window will pop up for setting passwords (See diagram 4.5.1a)

Diagram 4.5.1a Pop up windows for setting password

After users entered the password, the app will pop up another window for users to re-enter the password again. It is to ensure that the users have not set up wrong passwords due to typing mistakes. (See diagram 4.5.1b)

Diagram 4.5.1b Pop up windows for re-entering password

4.5.2 User logging in

After setting passwords, the users are required to login before entering the pages that loads user's account data. If the password entered is not correct, the app will pop up another window again for re-entering. (See diagram 4.5.2)

Diagram 4.5.2 Pop up windows for logging in

4.5.3 Sending payment Requests

After checking out all the items purchased by the clients, the merchant can input the amount to pay into the merchant's app. The merchant's name and merchant's ID selected will be shown automatically on the interface. After inputting the amount to pay and pressing the button "Request payment", the app will send out the voice signals that contain the merchant's ID and the amount of purchase through the phone speakers. During the sending of data, the merchant can cancel the request if needed (See diagram 4.5.1).

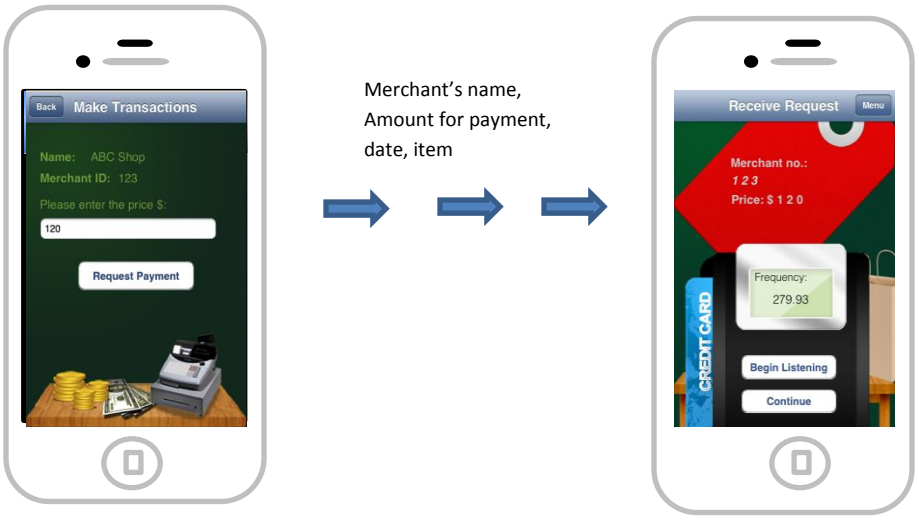


Diagram 4.5.1 Merchant's app sends payment requests to client's app

4.5.4 Client processing card payment

Once the payment request from the merchants is received, the client's app will show the merchant's ID and the amount to pay for the clients to approve. The client can click "Continue" to continue the payment process or click "Begin Listening" to receive the signals again. After choosing the card account for payment and clicking "Proceed Payment" button, the client's app will send the cardholder's account details to the merchant (See diagram 4.5.2a).

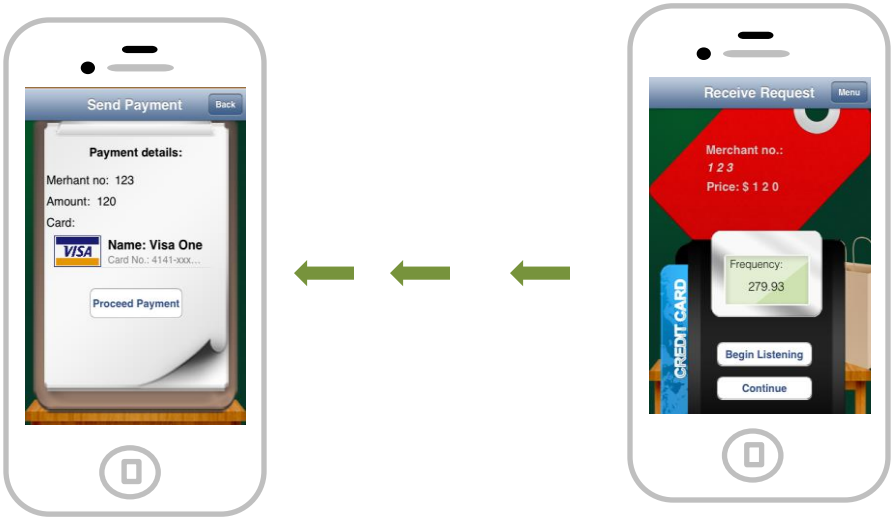


Diagram 4.5.2 Client's app chooses card detail to merchant

4.5.5 Sending confirmations and receipts for the payment

After the card details are received from the client, the merchant's app will show the client's card details received. The merchant can press the "Continue" button to see the confirmation page. Then, the merchant presses the "Confirm" button to send the transaction details to credit card company servers or press "Cancel" Button to exit (See diagram 4.5.3a). If the "Confirm" button is pressed, the merchant app will send the data to the card server through Wi-Fi or 3G for transaction.

Diagram 4.5.3a Merchant's view for confirming card detail to

After the confirmation from the server is received, the merchant's app will send the receipt to the client's app again through the speaker of the phone. (See diagram 4.5.3)

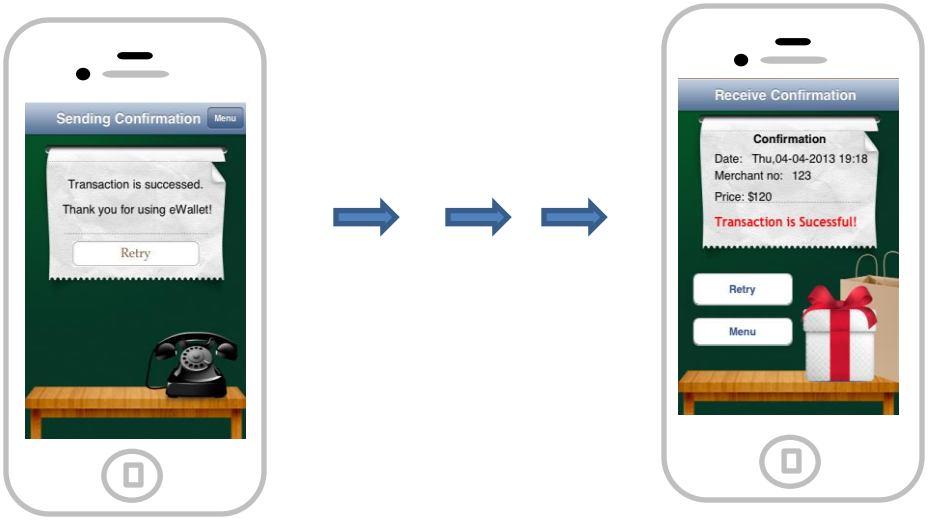


Diagram 4.5.3 Merchant's app sends receipt for transaction to client's app

4.5.6 Managing user's card/merchant accounts

In this project, the app users are allowed to store multiple accounts for transactions. So, they can choose either one account for processing payments. They can manage their accounts by the options below:

a. <u>Viewing stored accounts</u>

Both merchant and client users are allowed to view the list of accounts stored. If they selected any row in the list, the related account's details will be shown on the interface. (See diagram 4.5.6a and diagram 4.5.6b)

Diagram 4.5.6a Merchant's view of account list and account details

Diagram 4.5.6b Client's view of account list and account details

b. Adding new accounts

Users are able to add new accounts by clicking the “+” button at the upper right corner of the app (See diagram 4.5.6c and diagram 4.5.6d). After entering all the data required on the page, users can press the “Save” button to store the account details.

Diagram 4.5.6c Merchant’s view of adding new accounts

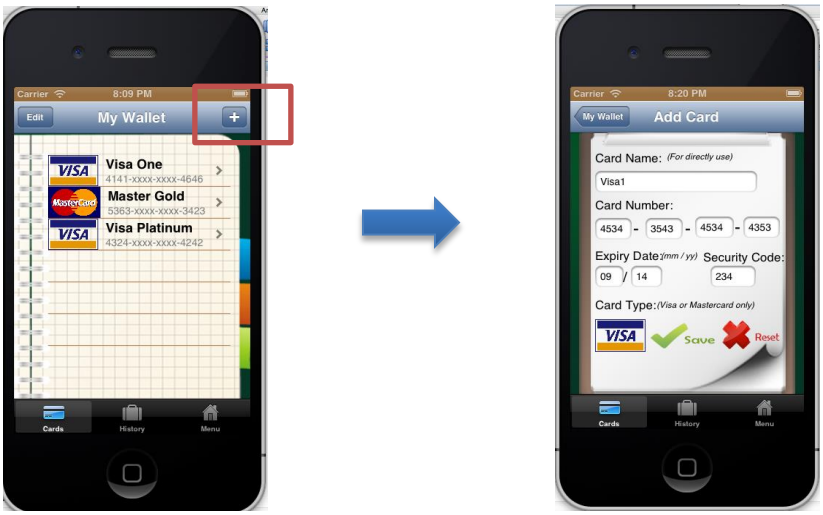


Diagram 4.5.6d Client’s view of adding new accounts

c. Deleting stored accounts

Users are able to delete the stored accounts by swiping the selected row in the list. A “Delete” button will appear at the right of the row swiped. The account of the selected row will then be deleted after user pressed the “Delete Button”. (See diagram 4.5.6e and diagram 4.5.6f)

Diagram 4.5.6e Merchant’s view of deleting accounts

Diagram 4.5.6f Client’s view of deleting accounts

d. Setting the default merchant account

Merchant users can set the default merchant account so that the account data selected will be shown and loaded automatically on the transaction page. (See diagram 4.5.6g)

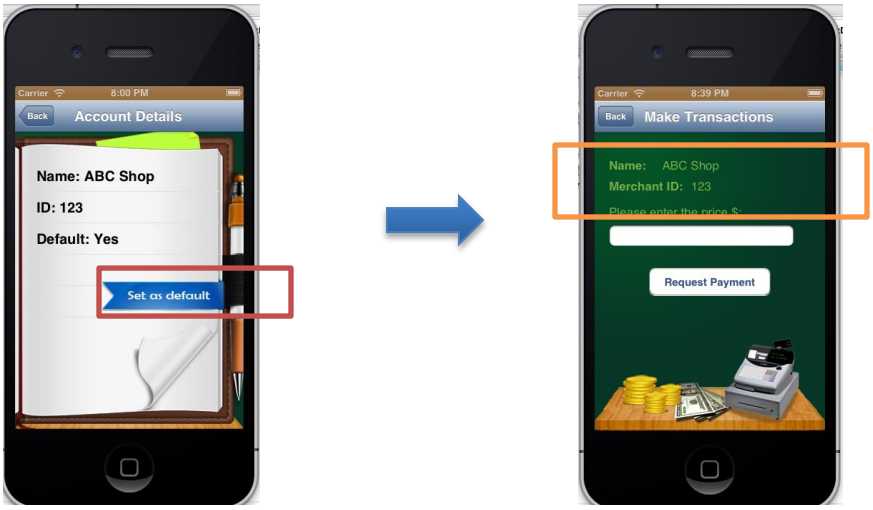


Diagram 4.5.6g Merchant's view of setting default account

4.5.7 Viewing transaction histories

The details for each successful transaction will be stored automatically by the app. Both the merchant and client users are able to view their transaction records. (See diagram 4.5.7)

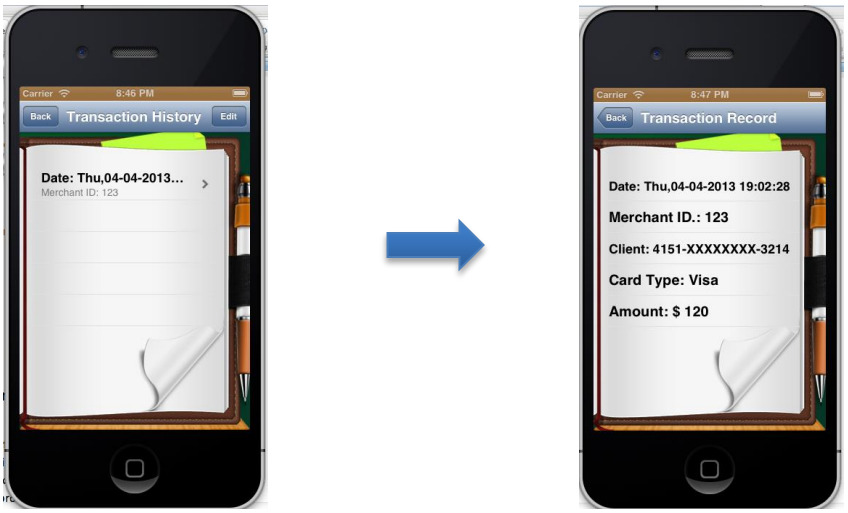


Diagram 4.5.7 Merchant and client's view of transaction histories

4.5.8 Resetting Password

The users can reset their passwords by clicking the “Setting” button on the main menu. A pop up menu will be shown for options. (See diagram 4.5.8a)

Diagram 4.5.8a Pop p windows for setting password

After choosing the button for changing password, the app will pop up a window for entering the old password for authentication first. Then, the user will be asked for entering the new passwords for 2 times to avoid any typing mistakes. (See diagram 4.5.8b)

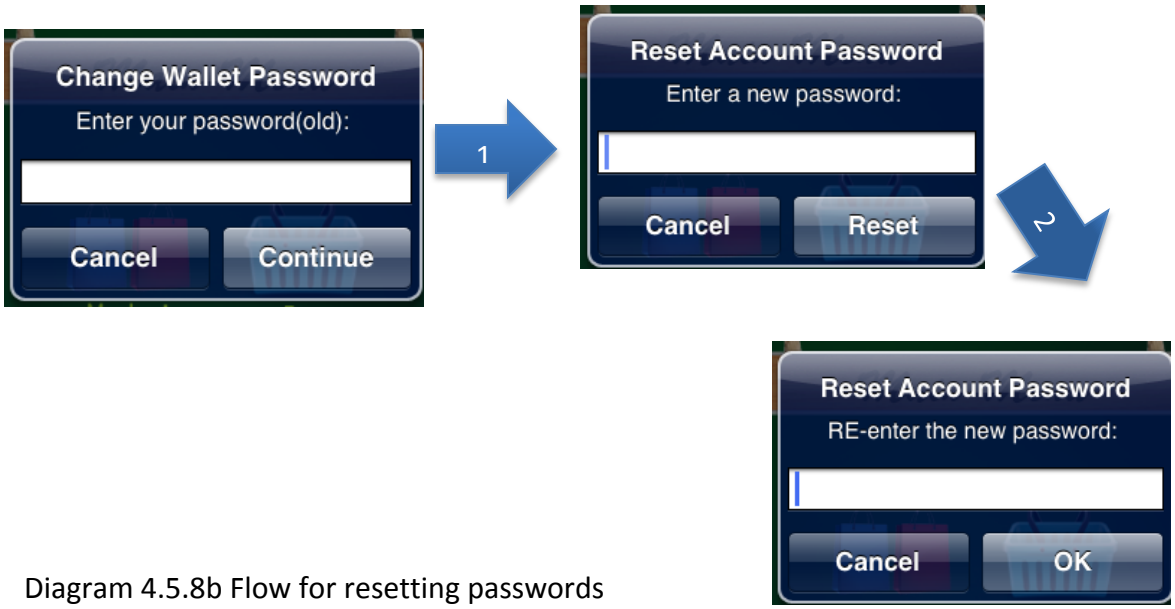


Diagram 4.5.8b Flow for resetting passwords

4.6 Use case diagram of the payment processing app system

The use case diagram in Figure 3.5 illustrates the functions of the iPhone credit card payment app. Only the functional requirements will be shown in the following diagram. There are two actors interacting with the system, the client and the merchant. The merchant is able to use the app for sending request to clients, exchanging data with the clients, sending payment request to the credit card servers for approval and sending receipts to clients if the transactions are succeed. While the client is processing the payment by receiving the payment requests sent from the merchants, exchanging card account data for payments and receiving receipts after the transactions are completed.

The function “Processing payment” includes the exchange of data between merchants and clients (e.g. merchant’s names, amounts to pay, client’s card account numbers…etc.) and the approval of the payment from the client. Moreover, merchants send receipts of the transactions after the credit card servers have approved the payments. Otherwise, the merchant’s apps notify the clients about the failures of payment.

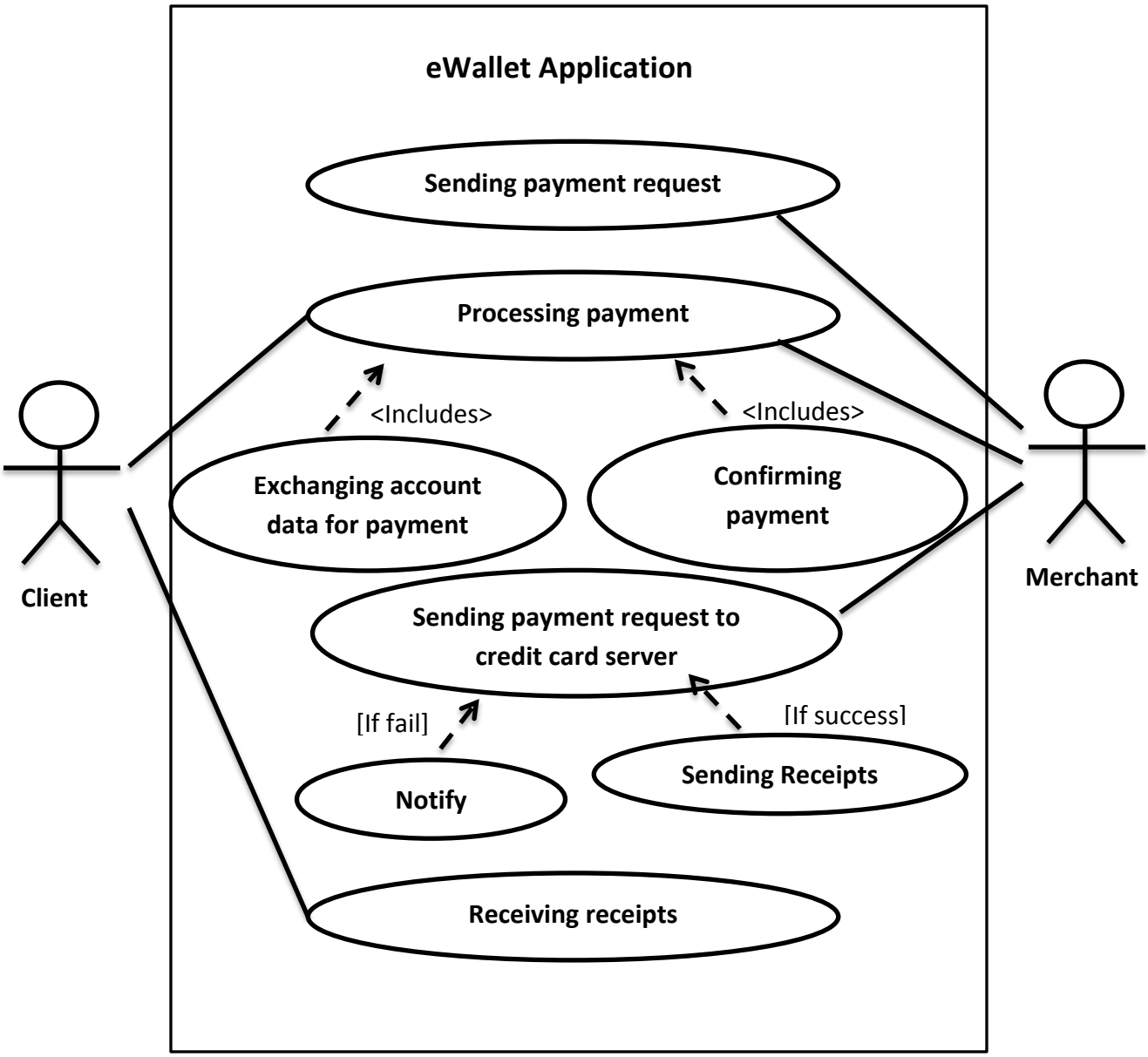


Figure 4.6 Use case diagram of the eWallet Application

4.7 Flow Diagram for the payment processing app system

Figure 4.7 shows the flow on how the transaction process. In the beginning, a merchant uses the app to send payment request to a client. After the data are transferred to the client's phone by the sounds of different frequencies, the client can choose whether to accept or reject the payment. If the client rejects it, the transaction is terminated. Otherwise, the client's app sends the credit card details back to the merchant's phone after the client pass the security for inputting the PIN of the credit card. Finally, the merchant's app sends the card details received from to the credit card server for the approval of transaction. If the transaction is successful, the merchant's app sends a receipt to the client's app. Otherwise, the merchant's app notifies the clients for the failure of transaction.

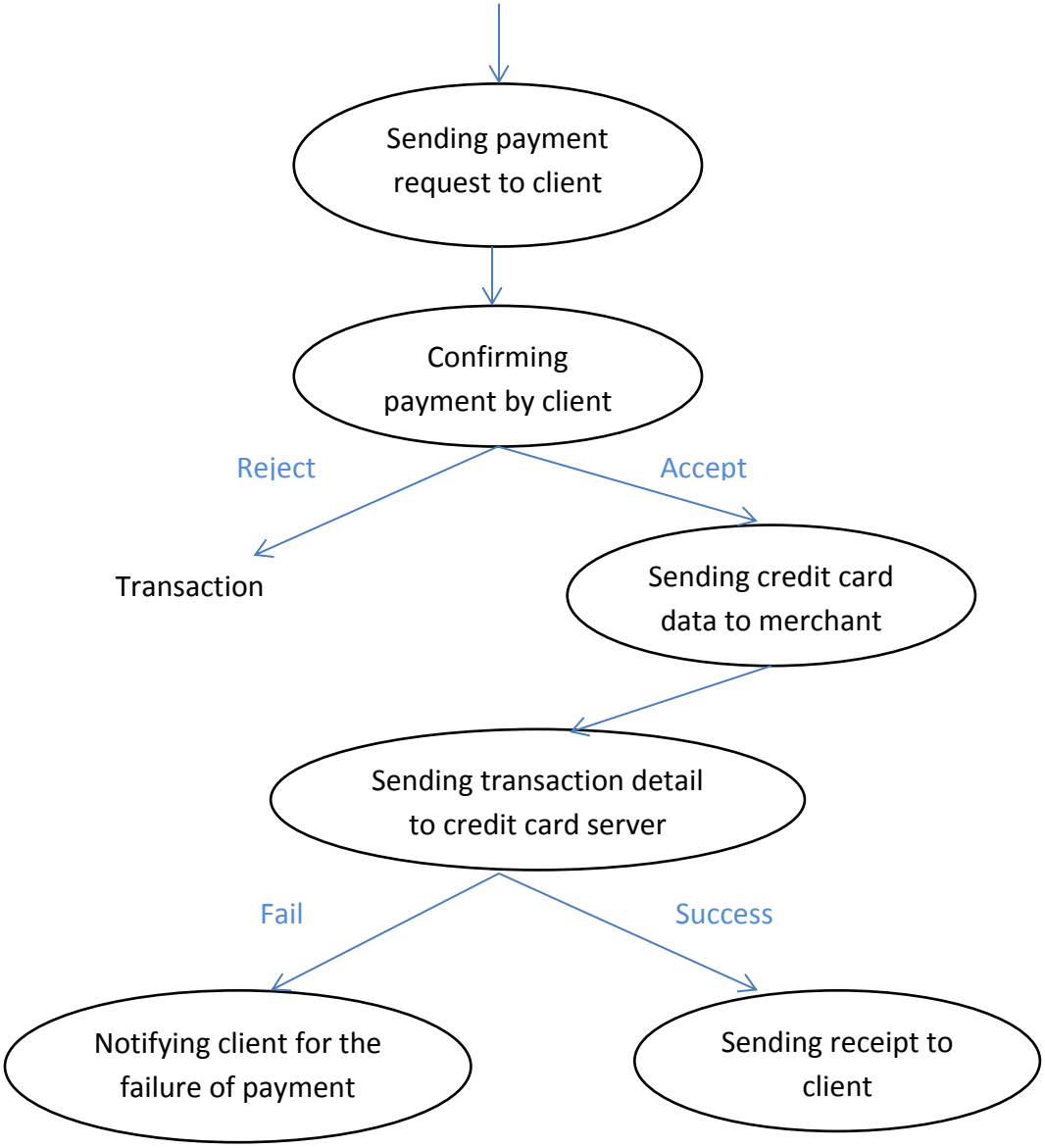


Figure 4.7 Use case diagram of the iPhone Credit Card Payment App

5. Implementation and testing

This section evaluates the programming tools used, describes the implementation techniques applied and illustrates the testing and experiments that was taken to the application.

5.1 Programming tools evaluation

5.1.1 Xcode - Version 4.5.2 (4G2008a)

Xcode is an Integrated Development Environment (IDE) that contains a suite of software development tools developed by Apple for developing software for OS X and iOS. It supports C, C++, Objective-C, Objective-C++, Java, AppleScript, Python and Ruby source code with a variety of programming models, including but not limited to Cocoa, Carbon, and Java.

In the development of the iPhone application in this project, Xcode is used as the main development tool for implementations. Xcode is chosen because it is free for mac book users. Besides, Xcode supports objective - C and is equipped with useful UI tools and models for interface design in this iPhone application.

Figure 5.1.1a Start Page of Xcode 4.6

An IOS simulator is equipped in Xcode 4.6 that it provides conveniences for developers to debug and check the performances of the applications. It allows developers to preview the actuals interactions that will take place when the application is running on an IOS device such as iPhone or iPad.

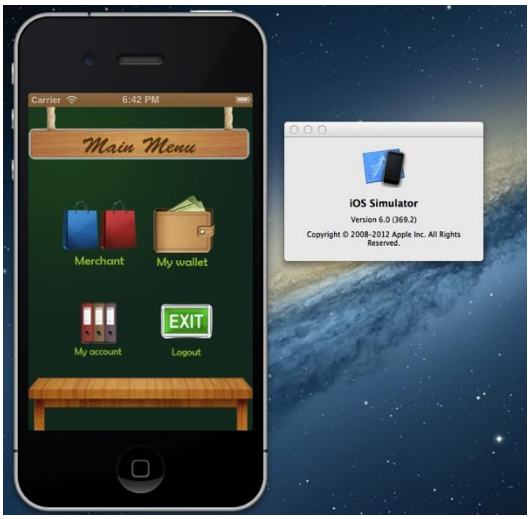


Figure 5.1.1b IOS simulator in Xcode 4.6

5.1.2 Development Provisioning Assistant

For deploying the eWallet application into IOS devices, a certification authority (CA) certificate is signed by the Apple Inc. With the use of development provisioning assistant in the IOS developer website, a certificate request for developer can be sent to the Apple signing authority. By following the steps of key assistant with the registered developer's account, a certification authority (CA) certificate will be signed that allows the deployment of the codes to the IOS devices.

5.2 Process Model

In this project, the development process is mainly divided into 4 Stages that can be revised as needed. The five stages are: requirements, design, develop, testing and evaluation(see Figure 5.2.).

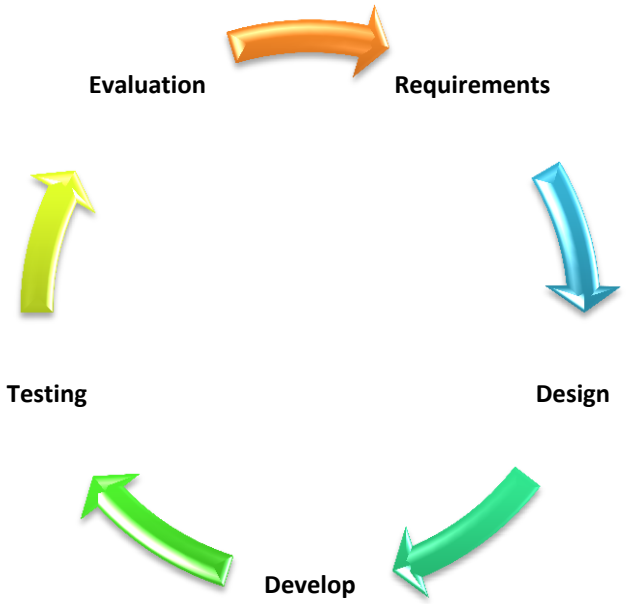


Figure 5.2 Process Model

In the beginning of the project, we start with the Design stage first. It includes the literature reviews for studying if the methods chosen are feasible for the project, and then it comes to the system design, user interface design and object orientated design that define the components needed to be developed. After the basic researches and designs, the keys functions for the application like the generation and detection of voices is developed.

After the developing of the key functions in the application, it comes to the testing stage to test if the development works. The generation and detection of sounds in different frequencies are the most impartment parts for the whole project. They are the keys for voice signal transmissions. So, it should be developed in earlier stage and be tested sufficiently in order to enable enough time for debugging. After the debugging stage, revising the testing stage to test again if the key functions work appropriately. Then, refactor the codes for better code structure before adding functions.

5.3 Programming technique

5.3.1 IOS tone generator using Audio Units

AudioUnits are defined as the lowest level of sound generation on iOS and it is the lowest hardware abstracted layer commonly used on the Mac devices. The audioUnits are able to produce raw audio samples and place them into output buffers. (Gallagher, 2013)

The audio samples can be generated by calculating the values of audio waveform. The tone to be created is actually a basic sine wave. So the value of it can be calculated by the equation below:

$$f(n) = a \sin (\theta(n))$$

In the above equation, n is the current sample that is pre-defined as 44100, a is the sound's amplitude and θ(n) is the current angle of the waveform, and a sine curve that show the result of $\sin (\theta(n))$ can been seen in graphs (see Figure 5.3.1).

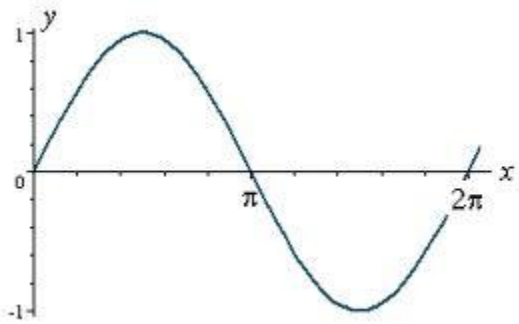


Figure 5.3.1 Sine Curve

θ(n) can be calculated by the equation as follows:

$$\theta(n) = 2\pi f \, n / r$$

ƒ is the frequency of the tone we want to generate for each input text and r is the sample rate of the audio we're generating.

With the equations above, we can create the audio unit simply by the function like this:

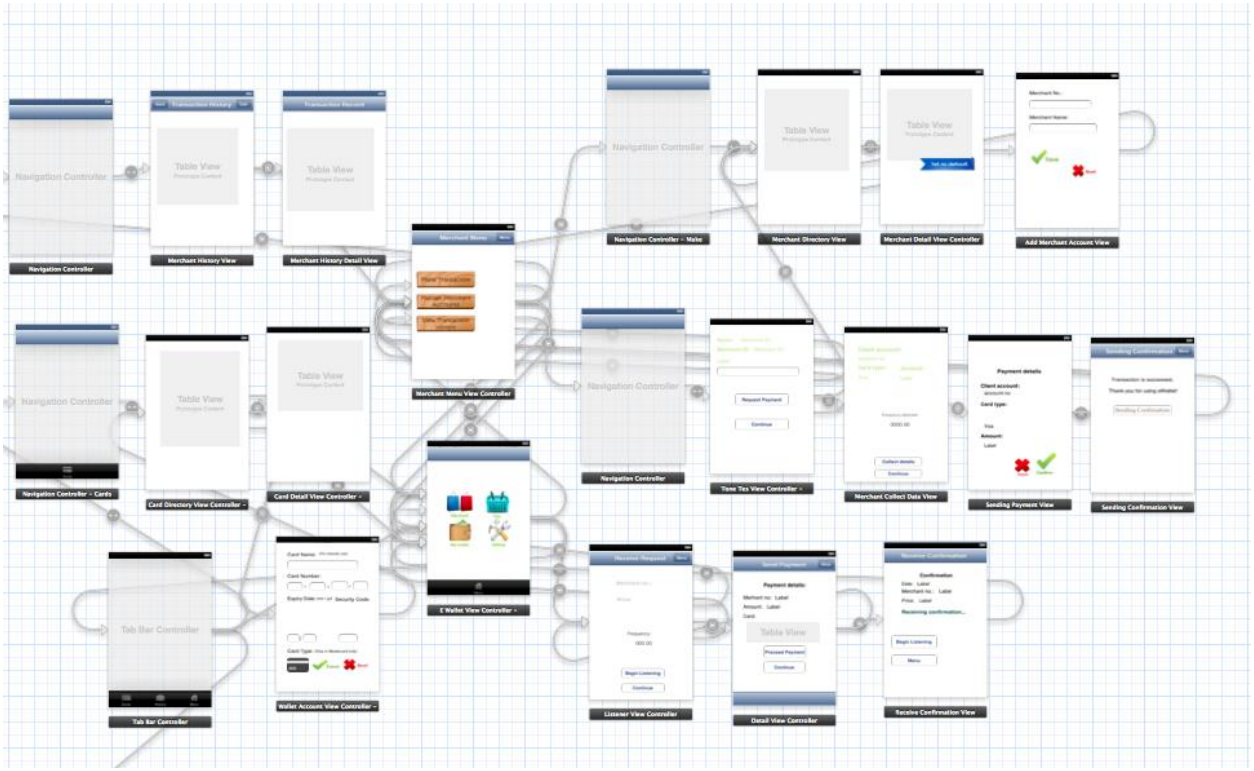
```
OSStatus RenderTone(
    void *inRefCon,
    AudioUnitRenderActionFlags *ioActionFlags,
    const AudioTimeStamp *inTimeStamp,
    UInt32 inBusNumber,
    UInt32 inNumberFrames,
    AudioBufferList *ioData)

{
    // Fixed amplitude is good enough for our purposes
    const double amplitude = 0.25;

    // Get the tone parameters out of the view controller
    ToneGeneratorViewController *viewController =
        (ToneGeneratorViewController *)inRefCon;
    double theta = viewController->theta;

    double theta_increment =
        2.0 * M_PI * viewController->frequency / viewController->sampleRate;

    // This is a mono tone generator so we only need the first buffer
    const int channel = 0;
    Float32 *buffer = (Float32 *)ioData->mBuffers[channel].mData;

    // Generate the samples
    for (UInt32 frame = 0; frame < inNumberFrames; frame++)
    {
        buffer[frame] = sin(theta) * amplitude;

        theta += theta_increment;
        if (theta > 2.0 * M_PI)
        {
            theta -= 2.0 * M_PI;
        }
    }

    // Store the updated theta back in the view controller
    viewController->theta = theta;

    return noErr;
}
```

θ(n) = 2π*f* n / r

f(n) = a sin (θ(n))

5.3.2 Use storyboard as user interface development

The storyboard feature in Xcode is very favorable for designing user interfaces. The View Controllers act as a sort of the glue between the programming codes and the user interfaces. By adding view controllers, the links between different screens are clearly shown and defined.



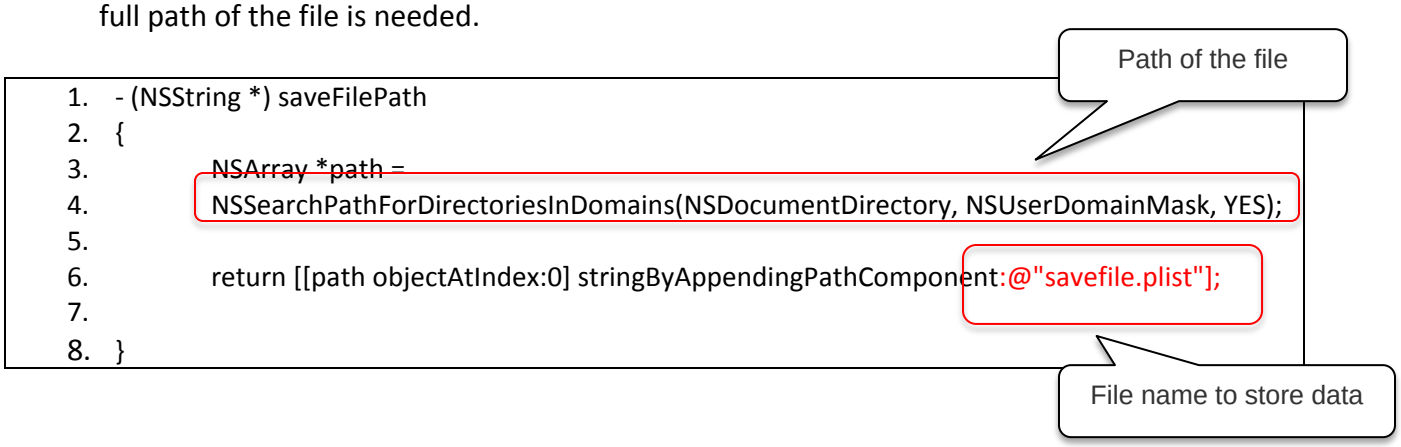
The storyboard provides a conceptual overview of all the screens that makes it easier for human understanding on the performance and the flows of the application interfaces.

5.3.3 How to save and load data in files

IPhone does have a file system for storing and saving data (OSCARVGG, 2010). The followings are the steps for storing and loading data:

1. Get the path for the file used for saving data

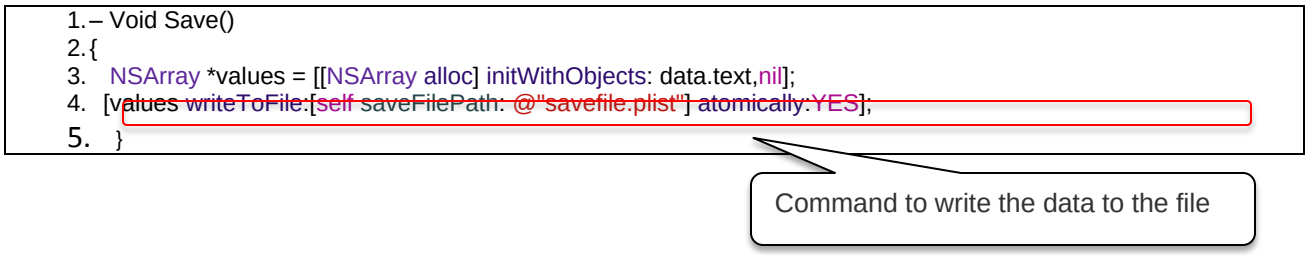
   In line3 of the coding below, we use NSArray "path" to point to the NSSearchPathForDirectoriesInDomains. It is an object that consists of 3 arguments: the directory, the user domain and a Boolean. The Boolean is set as "YES, it is to indicate a full path of the file is needed.



   In line 6 above, the path of the file is returned with an string to the directory. It is the name of the file. In this case, "savefile.plist" is the name of the file we wanted for saving data.

2. Saving data to file

   In the segment of coding below, an NSArray called "values"(line 3) points to the data that we want to store. Then the method "writeToFile" in line 4 save the "values" into the file indicated as an argument of it, "savefile.plist.



3. Loading data from file

   I. The followings are the coding for loading data from the files. Firstly, we have to get the path of a file for loading data (line 1).

II. Then, we check the existence of the file (line2-3). It is an important part that cannot be misses. It may cause run-time errors if we get data from the file that does not exist.

III. Finally, we read the data from the file in line 5.

```
1. NSString *myPath = [self saveFilePath];
2.   BOOL fileExists = [[NSFileManager defaultManager]
   fileExistsAtPath:myPath];
3. if (fileExists) {
4.   NSArray *values = [[NSArray alloc] initWithContentsOfFile:myPath];
5.   sha256.text = [values objectAtIndex:0];
6.
7.  }
```

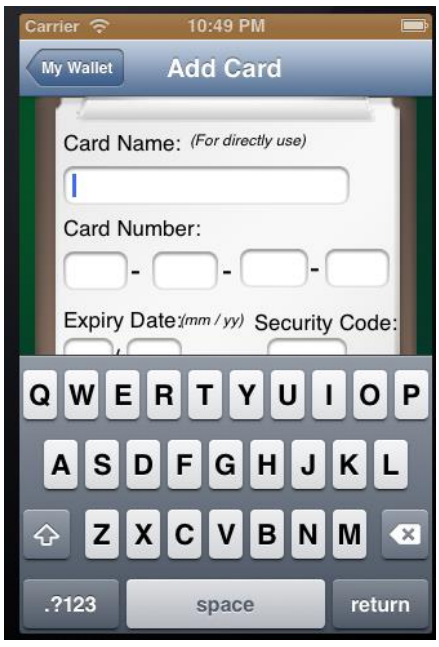
1. Get the path of the file

2. Check the existence of the file

3. Read data from file

5.3.4 How to slide UITextFields to avoid the keyboard

It is common and quite annoying that the keyboard popped up may cover the UITextFields when we are inputting data into the app's interface. (See figure 5.3.4)



Gallagher

Figure 5.3.4 Example for the keyboard that covers the UITextFields

(2008) indicated that the iPhone's keyboard occupies 216 pixels at the bottom of the screen. It covers nearly half of the screen that prevents us from editing the text fields at the bottom of the form. As a result, Gallagher proposed to slide the UITexfields as below:

Firstly, he divided the screen into three parts:

No sliding animation will be performed when the text fields in the top section are edited. The middle section and the bottom section will animate upwards in different proportions of the screen.

Below shows the fractions for animating upwards for the UITextFields and the height of keyboard in portrait view:

```
static const CGFloat KEYBOARD_ANIMATION_DURATION = 0.3;
static const CGFloat MINIMUM_SCROLL_FRACTION = 0.2;
static const CGFloat MAXIMUM_SCROLL_FRACTION = 0.8;
static const CGFloat PORTRAIT_KEYBOARD_HEIGHT = 216;
```

The coding for animating upward the screen is as follows:

```
- (void)textFieldDidBeginEditing:(UITextField *)textField
{
    CGRect textFieldRect =
    [self.view.window convertRect:textField.bounds fromView:textField];
    CGRect viewRect =
    [self.view.window convertRect:self.view.bounds fromView:self.view];

    CGFloat midline = textFieldRect.origin.y + 0.5 * textFieldRect.size.height;
    CGFloat numerator =
    midline - viewRect.origin.y
    - MINIMUM_SCROLL_FRACTION * viewRect.size.height;
    CGFloat denominator =
    (MAXIMUM_SCROLL_FRACTION - MINIMUM_SCROLL_FRACTION)
    * viewRect.size.height;
    CGFloat heightFraction = numerator / denominator;

    if (heightFraction < 0.0)
    {
        heightFraction = 0.0;
    }
    else if (heightFraction > 1.0)
    {
        heightFraction = 1.0;
    }

    UIInterfaceOrientation orientation =
    [[UIApplication sharedApplication] statusBarOrientation];
    if (orientation == UIInterfaceOrientationPortrait ||
        orientation == UIInterfaceOrientationPortraitUpsideDown)
    {
        animatedDistance = floor(PORTRAIT_KEYBOARD_HEIGHT * heightFraction);
    }
    else
    {
        animatedDistance = floor(LANDSCAPE_KEYBOARD_HEIGHT * heightFraction);
    }

    CGRect viewFrame = self.view.frame;
    viewFrame.origin.y -= animatedDistance;

    [UIView beginAnimations:nil context:NULL];
    [UIView setAnimationBeginsFromCurrentState:YES];
    [UIView setAnimationDuration:KEYBOARD_ANIMATION_DURATION];

    [self.view setFrame:viewFrame];

    [UIView commitAnimations];

}
```

1. Get the rects of the text

2. Calculate the fraction between the top and

3. Take the fraction and convert it into an amount to

4. Apply the animate

The result of the coding above will be shown as follows:

5.4 Programming techniques related to security issues

We have explained lots of design on the topic of “Secure Storage of Sensitive Data” in section 4.4. However, we have not yet explained the details on how to implement these algorithms. This part is going to show the techniques and practical uses of these security issues on the app’s development.

5.4.1 Programming for hash password

After explaining the concept on how to use hash algorithm for protecting user’s password, this part is about the programming codes and steps used to hash the actual password.

1. Storing the hash value of the password

It is very convenient that the hash algorithm, SHA256, is already included in the reference library of Objective- C. We can simply call the method “CC_SHA256” after importing the reference .h file, “CommonCrypto/CommonDigest.h” by the command below:

```
#import <CommonCrypto/CommonDigest.h>
```

After it, we can input the following codes for hashing the password and storing it into the file we decided:

```
NSData* inputData = [passwordStored dataUsingEncoding: NSUTF8StringEncoding]; // Could use UTF16 or other if you like


//hash the password before storing
unsigned char passwordDgstchar[CC_SHA256_DIGEST_LENGTH];
        CC_SHA256([inputData bytes],[inputData length],passwordDgstchar);



//Convert unsinged char to String
 NSMutableString* sha256 = [[NSMutableString alloc] init];
 for (int i = 0 ; i < CC_SHA256_DIGEST_LENGTH ; ++i)
 {
     [sha256 appendFormat: @"%02x", passwordDgstchar[i]];
 }


 //Store the based file in plist
 NSString *myPath = [self saveFilePath: @"walletAccount.plist"];
 NSArray *values = [[NSArray alloc] initWithObjects: sha256,nil];

[values writeToFile:[self saveFilePath: @"walletAccount.plist"] atomically:YES];
```

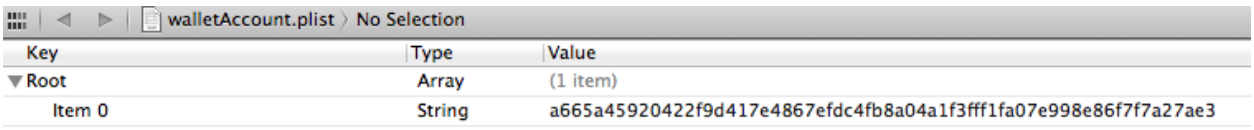
1. Convert the string to NSData as the argument of CC_SHA256 is in NSData

2. Hash the data using method "CC_SHA256"

3. The result type of CC_SHA256 is in unsigned char [], so we convert the result into NSString

4. Store the hash value into file

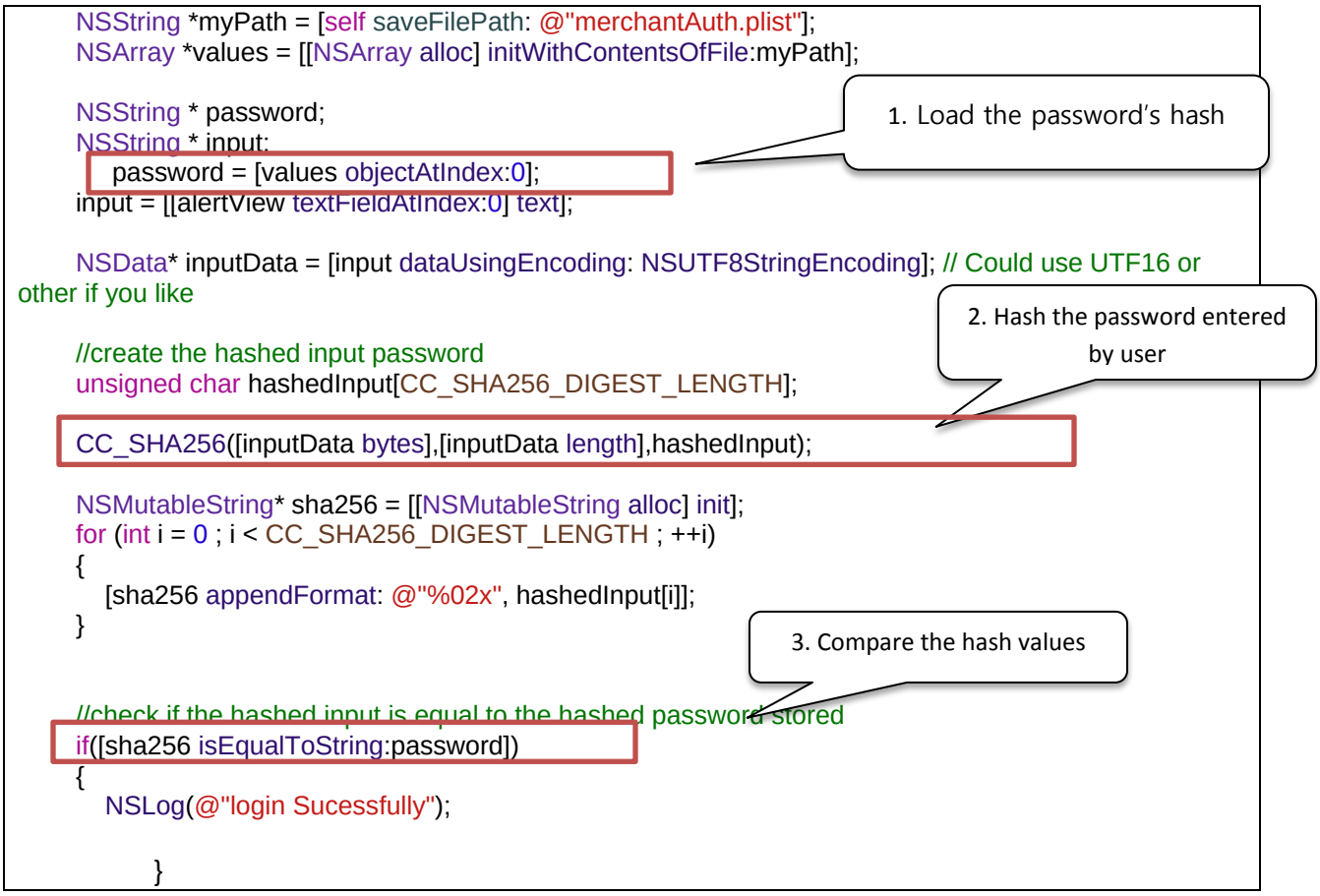
After typing the codes above, a hashed value of the password is stored in the file:



2. Hashing and comparing the password when user logging in

As explained in section 4.4.2, there are 3 main steps for authentication when the user is logging in. They are: i) Load the hash value of password stored, ii) Hash the password entered by user and iii) Compare the hash values. In this part, we are just reusing the CC_SHA256 in part 1 to hash the password user just input. And then convert it into NSString for comparison.

Let's have look at the coding below:



5.4.2 Programming for Encrypting and decrypting data

Encryption and Decryption of data is a kernel part of the security of our app. It differs from the use of hash algorithm, as we have to recover the data after they are encrypted. In this project, AES256 is selected as the cipher for encrypting and decrypting our data.

1. Encrypting the data for storage

   As mentioned in section 4.4.2, there are 4 main steps to encrypt the data: i) Generate a random key in 128 bits, ii) Encrypt the data using the random key, iii) Encrypt the random key using user's password and, iv) Store the encrypted key and encrypted data

   The actual codes can been seen as below:

```
NSString *keyEncryption;

//get the key for decrpt the key for encryption
NSString * key = [[NSUserDefaults standardUserDefaults] objectForKey:@"MerchantKey"];

keyEncryption = [self randomKey:16];



NSString *myPath = [self saveFilePath: @"merchantAccount.plist"];
BOOL fileExists = [[NSFileManager defaultManager] fileExistsAtPath:myPath];


NSString *merNo = merchantNo.text;
NSString *merName = merchantName.text;

//Convert data to NSdata
NSData* dataNo=[merNo dataUsingEncoding:NSUTF8StringEncoding];
NSData* dataName=[merName dataUsingEncoding:NSUTF8StringEncod

//Encrypt data
NSData *cipherNo = [self AES256EncryptWithKey:keyEncryption text:dataNo];
NSData *cipherName = [self AES256EncryptWithKey:keyEncryption text:dataName];

NSMutableDictionary *nameDictionary = [NSMutableDictionary dictionary];

//A
 [nameDictionary setValue: cipherNo forKey:@"1"];
 [nameDictionary setValue:cipherName forKey:@"2"];

if (!fileExists) {
 [nameDictionary setValue: @"Yes" forKey:@"3"];
}else
 [nameDictionary setValue: @"No" forKey:@"3"];


NSMutableArray *plist = [NSMutableArray arrayWithContentsOfFile:myPath];


if (plist == nil) plist = [NSMutableArray array];
 [plist addObject:nameDictionary];


[plist writeToFile:[self saveFilePath: @"merchantAccount.plist"] atomica


//Convert data to NSdata
NSData* dataKey=[keyEncryption dataUsingEncoding:NSUTF8StringEnco

//Encrypt data
NSData *cipherKey = [self AES256EncryptWithKey:key text:dataKey];
```

1. Generate the random

2. Encrypt the data using the random key

4. Save the encrypted data

3. Encrypt the random key

```
NSArray *values = [[NSArray alloc] initWithObjects: cipherKey,nil];
[values writeToFile:[self saveFilePath: @"merchantKey.plist"] atomically:YES];
```

4. Save the encrypted random key

The "AES256EncryptWithKey" methods is the key part for the whole encryption process, the coding of the method is:

```
- (NSData *)AES256EncryptWithKey:(NSString *)key  text: (NSData *) msg{
    // 'key' should be 32 bytes for AES256, will be null-padded otherwise
    char keyPtr[kCCKeySizeAES256+1]; // room for terminator (unused)
    bzero(keyPtr, sizeof(keyPtr)); // fill with zeroes (for padding)

    // fetch key data
    [key getCString:keyPtr maxLength:sizeof(keyPtr) encoding:NSUTF8StringEncoding];

    NSUInteger dataLength = [msg length];

  //See the doc: For block ciphers, the output size will always be less than or
    //equal to the input size plus the size of one block.
    //That's why we need to add the size of one block here
    size_t bufferSize = dataLength + kCCBlockSizeAES128;
    void *buffer = malloc(bufferSize);

    size_t numBytesEncrypted = 0;
    CCCryptorStatus cryptStatus = CCCrypt(kCCEncrypt, kCCAlgorithmAES128, kCCOptionPKCS7Padding,
                                          keyPtr, kCCKeySizeAES256,
                                          NULL /* initialization vector (optional) */,
                                          [msg bytes], dataLength, /* input */
                                          buffer, bufferSize, /* output */
                                          &numBytesEncrypted);
    if (cryptStatus == kCCSuccess) {
        //the returned NSData takes ownership of the buffer and will free it on deallocation
        return [NSData dataWithBytesNoCopy:buffer length:numBytesEncrypted];
    }

    free(buffer); //free the buffer;
    return nil;
}
```

The method "CCCrypt" used is a defined method in the reference library of Objective –C. The .h reference file is "CommonCrypto/CommonCryptor.h>".

2. Decrypting the data for storage

The decrypting of data is like a reverse of the coding in part 1, but the method "AES256DecryptWithKey" is used for decrypting data instead. It also uses "CCCrypt" which is defined method in the reference library of Objective –C. The .h reference file is "CommonCrypto/CommonCryptor.h>".

The coding of “AES256DecryptWithKey” is:

```
- (NSData *) AES256DecryptWithKey:(NSString *)key text:(NSData*)msg{
    // 'key' should be 32 bytes for AES256, will be null-padded otherwise
    char keyPtr[kCCKeySizeAES256+1]; // room for terminator (unused)
    bzero(keyPtr, sizeof(keyPtr)); // fill with zeroes (for padding)

    // fetch key data
    [key getCString:keyPtr maxLength:sizeof(keyPtr) encoding:NSUTF8StringEncoding];

    NSUInteger dataLength = [msg length];

    //See the doc: For block ciphers, the output size will always be less than or
    //equal to the input size plus the size of one block.
    //That's why we need to add the size of one block here
    size_t bufferSize = dataLength + kCCBlockSizeAES128;
    void *buffer = malloc(bufferSize);

    size_t numBytesDecrypted = 0;
    CCCryptorStatus cryptStatus = CCCrypt(kCCDecrypt, kCCAlgorithmAES128, kCCOptionPKCS7Padding,
                                          keyPtr, kCCKeySizeAES256,
                                          NULL /* initialization vector (optional) */,
                                          [msg bytes], dataLength, /* input */
                                          buffer, bufferSize, /* output */
                                          &numBytesDecrypted);

    if (cryptStatus == kCCSuccess) {
        //the returned NSData takes ownership of the buffer and will free it on deallocation
        return [NSData dataWithBytesNoCopy:buffer length:numBytesDecrypted];
    }

    free(buffer); //free the buffer;
    return nil;
}
```

5.4.3 Validation of user input

The validation of user input is also a part of our app’s security concerns that cannot be neglected. The security level of the app’s authentication system depends on the setting of the password. The overflow of data may lead to a serve data leakage because of user’s malicious inputs. Also, the invalid data will lead to unsuccessful transactions. As a result, the validation of user input is needed.

1. <u>The checking of expiry date for the card accounts</u>

   As to check whether the card is expired, the app get the updated current year and month from NSDate that is already defined in Ojective-C:

```
// Get current date time
NSDate *currentDateTime = [NSDate date];
// Instantiate a NSDateFormatter
NSDateFormatter *dateFormatter = [[NSDateFormatter alloc] init];
// Set the dateFormatter format
//[dateFormatter setDateFormat:@"yyyy-MM-dd HH:mm:ss"];
[dateFormatter setDateFormat:@"yy"];
// Get the date time in NSString ofr current year
NSString *dateInStringFormatedYear = [dateFormatter stringFromDate:currentDateTime];

// get current Month
[dateFormatter setDateFormat:@"MM"];
NSString *dateInStringFormatedmonth = [dateFormatter stringFromDate:currentDateTime];
```

Then, we use the current year and moth obtained to check the expiry date entered by user:

```
if ([self.exYear.text integerValue] < [dateInStringFormatedYear integerValue] || ([self.exYear.text
integerValue] == [dateInStringFormatedYear integerValue] && [self.exMonth.text integerValue] <
[dateInStringFormatedmonth integerValue]))
   {
      stateLabel.text = @"The card is not expired.";

   }
```

2. <u>The checking of card account numbers</u>

The preliminary checking of card account numbers is required as our app support Visa and Mastercard only. It is checked that the card account numbers of Visa and Mastercard start with '4' and '5' respectively. So, the aims of this validation are to check if the first digits of the account numbers are either '4' or '5' and to check if the users have input totally 16 digits. The coding is:

```
if([accountNoInput1.text length]>0)
   {
      cardNumber = accountNoInput1.text;
      cardTypeIndicator=[cardNumber substringWithRange:NSMakeRange(0,1)];
   }

if([self.accountNoInput1.text length]< 4 || [self.accountNoInput1.text length]> 4 ||
[self.accountNoInput2.text length]< 4  || [self.accountNoInput2.text length]> 4 ||
[self.accountNoInput3.text length]< 4 || [self.accountNoInput3.text length]> 4 ||
[self.accountNoInput4.text length]< 4 || [self.accountNoInput4.text length]> 4)
   {
```

```
        stateLabel.text = @"Card Number. is incorrect.";

    }else if (!([cardTypeIndicator isEqualToString: @"4"] || [cardTypeIndicator isEqualToString:
@"5"])) {

        stateLabel.text = @"Card Type is not correct."
```

But their may always be typing mistakes, so the app limit the number of digits that can be input in the text fields by using the following programming codes:

```
if ([strin length] > 0) {

  if([textField.text  length]>3)
  {

      if( textField == accountNoInput1)
      {
          [accountNoInput1 resignFirstResponder];
          [accountNoInput2 becomeFirstResponder];

          if ([textField.text  length]>=4) {
              return [textField.text  length] < 4;

          }
    }

}
```

Change the text field to be focused

Delete the last digit of more than 4 digits are typed

5.5 Difficulties, Solution and Limitations

There are some difficulties met during the development stages. The descriptions and suggested solutions of these problems are listed in this part.

5.5.1 The function of collecting signal frequencies doesn't work on iPad

After developing the function of signal collection, we tested it many times by using the iPhone simulator equipped in Xcode. It worked ordinary. However, when we install the app into an iPad, an error occurred. The iPad was unable to collect signals and the app was terminated immediately due to the buffer –related problems.

After reviewing the codes again, it is found that the variable "kBufferSize" which is to define the size of buffer used for collecting signals was set as 512. It seemed not to be compatible to the buffer size used in an iPad.

```
#define kInputBus 1
#define kOutputBus 0
#define kBufferSize 512
#define kBufferCount 1
#define N 10
```

After changing the "kBufferSize" into "1024", the function worked properly on both iPad and iPhone simulator again.

```
#define kInputBus 1
#define kOutputBus 0
#define kBufferSize 1024
#define kBufferCount 1
#define N 10
```

5.5.2 Different frequencies for audio detection between the machines

The frequencies for the voice detection and delivery are different among the machines due to the hardware limitation. It is observed that the microphones detect voices in approximately every 0.1 seconds. It means that there might be a missing of voice signals if each output sound lasts less than 0.1 seconds. As to avoid this situation, each output sounds should be released for more than 0.1 each times.

However, the longer each signal takes, the longer transmission time required for sending the whole set of data. There should be a balance between the missing of data and the slowdown of speed for transmission. Finally, the output sounds are set to be longer than 0.2 seconds.

5.5.3 The machine is only able to detect single tone

The microphones of the iPhones are able to detect one tone at each time only. It only detects the combination of frequencies for a dual tones signal. For examples, a signal with 100 Hz and 1000 Hz might be detected as 990 HZ only .If we really want to detect dual tones sounds, it might require the detection and analysis on the patterns of the sounds, e.g. A set of sound frequencies [702, 780, 859, 937, 1210, 1327, 1483] Hz will be detected as the combination of [607,1209] Hz (see Figure 5.4.2). It involves many complicated physical theories and algorithm (The MathWorks, 2004). The result frequencies used for DTMF detection can be seen in **appendix D**. The difficulty for analyzing the patterns is too high from the technical needs of our project. As a result, we would change the dual-tones sound signals into single-tone sound signal. As for the ease of testing, the frequencies in human hearable range will be used. The table for human hearable frequencies used for testing in this project can be seen in **appendix B**.

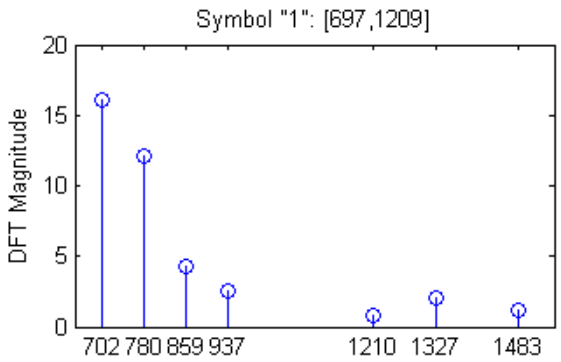


Figure 5.5.3 Result frequencies of a combination of 697 and 1209Hz

5.5.4 The duplication of voice signals

As mentioned in 5.4.2, the frequencies of voice detections are different in different machines. So, some of the voice signals may be detected twice or more. Take “123” as an example of the data delivered through voice signals, the detection side may detect the signals of “2”for two

times that is due to the different detection frequencies. And it gives the results as “1223”. In this situation, the duplicated signals should be disposed. However, the direct detention of duplicated signals is not favorable for the data with the same consecutive text such as “23334”. The 2nd and 3rd signals of “3” may be treated as the duplicated signed to be filtered.

As to solve this problem, two frequencies are assigned to each text. And the frequencies are used for signal generation alternatively when the same consecutive text is used for more than one time (see table 5.5.4a).

Text	Times	Frequency assigned for signal generation
1	1st	1000 Hz
	2nd	1500 HZ
2	1st	2000 Hz
	2nd	2500 Hz

Table 5.5.4a Example of frequencies used for representing each text

From the table above, if the text data is “1222”, then the application will use the 1st assigned frequency of “1”, the 1st assigned frequency of “2”, the 2nd assigned frequency of “2”, then reuse the 1st assigned frequency of “2”. So the output voices frequencies sequences are: “1000 Hz”, “2000 Hz”, “2500 Hz” and “2000 Hz”. (See table 5.5.4b and figure 5.5.4)

Sequences no.	Text input	Repeated times	Actual frequency output
1	“1”	1st of “1”	1000 Hz
2	“2”	1st of “2”	2000 Hz
3	“2”	2nd of “2”	2500 Hz
4	“2”	3rd => 1st of “2”	2000 Hz

Table 5.5.4b Example of frequencies for signal output

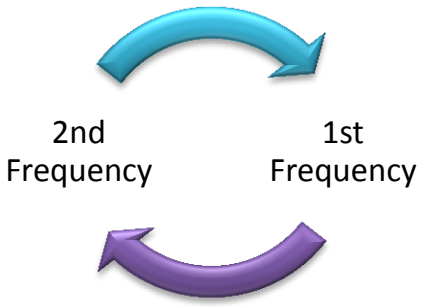


Figure 5.5.4 Cycle of frequency used for consecutive texts

5.5.5 The amplitude of high frequency voices

This project mainly uses a MacBook Air and an iPad for testing. After the testing son generating signals in different frequencies, it is found that the amplitudes for signals in frequencies higher than 9600 Hz generated is very low. The hardware we used cannot generate the curve we expected in section4.2. As a result, we decided to use the frequencies in hearing ranges.

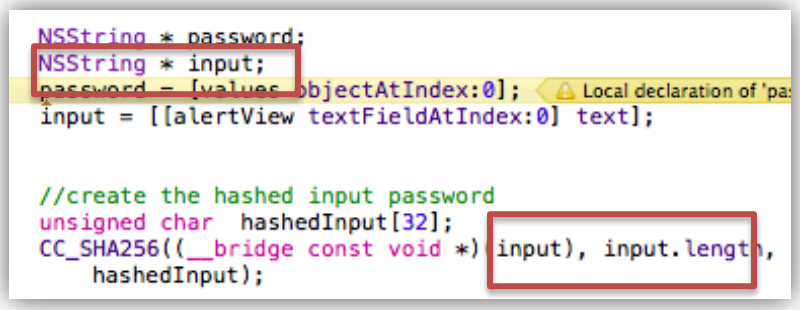
5.5.6 The background noise that are detected as data signals

As we are using the frequencies in hearing ranges, there may be much background noise that will be detected. Some of the background noises maybe recognized as data signals wrongly. As a result, a pair of start signals in higher frequencies is added to notify the app to begin collecting data signals. Similarly, a pair of end signals is also added to tell that app to stop collecting data signals. The start and end signals are attached at the head and tail of the data to be sent.

5.5.7 The hash value for password checking collided

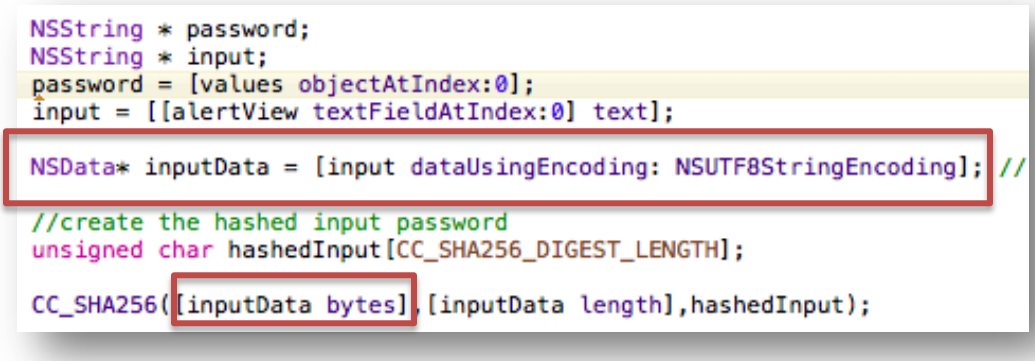
During the testing of the authentication feature, it was found that the hash value for logging in collided. For example, the hash value of “123” is the same as that of “134”. It was checked that the inputs with same number of digits would create the same hash values.

After reviewing the codes again, it is found the error occurred was due to the wrong data type of arguments used to pass to the SHA256 function. See picture 5.5.7a, an “NSString” named “input” was used as an argument in the CC_SHA256 function.



Picture 5.5.7a Wrong type of arguments used

However, the argument for CC_SHA256 function should be in “NSData” type but not “NSString”. It caused the generations of wrong hash values. As a result, the problem was solved by converting the “NSString” named “input” into a “NSData ” called “inputData”. Then, used “inputData” as an argument for CC_SHA256 function. (See Picture 5.5.7b)



Picture 5.5.7b Fixed problem on the wrong arguments used

5.5.8 Signals loss leads to wrong data received

When delivering data through different media, it is common that some signals may lose. The loss of the signals may lead to the receiving of wrong data especially for the long data like credit card account numbers. The wrong account numbers received may lead to the service errors and the loss of money in worst case. As a result, there should be a method to detect the error.

As to solve this problem, the CRC16 algorithm mentioned in section 4.4.3 will be applied to detect if there is an error during data transmission. A CRC table was added into the page that the use of CRC algorithm was need. (See Picture 5.5.8a)

```
static const unsigned short crc16table[] =
{
    0x0000, 0xC0C1, 0xC181, 0x0140, 0xC301, 0x03C0, 0x0280, 0xC241,
    0xC601, 0x06C0, 0x0780, 0xC741, 0x0500, 0xC5C1, 0xC481, 0x0440,
    0xCC01, 0x0CC0, 0x0D80, 0xCD41, 0x0F00, 0xCFC1, 0xCE81, 0x0E40,
    0x0A00, 0xCAC1, 0xCB81, 0x0B40, 0xC901, 0x09C0, 0x0880, 0xC841,
    0xD801, 0x18C0, 0x1980, 0xD941, 0x1B00, 0xDBC1, 0xDA81, 0x1A40,
    0x1E00, 0xDEC1, 0xDF81, 0x1F40, 0xDD01, 0x1DC0, 0x1C80, 0xDC41,
    0x1400, 0xD4C1, 0xD581, 0x1540, 0xD701, 0x17C0, 0x1680, 0xD641,
    0xD201, 0x12C0, 0x1380, 0xD341, 0x1100, 0xD1C1, 0xD081, 0x1040,
    0xF001, 0x30C0, 0x3180, 0xF141, 0x3300, 0xF3C1, 0xF281, 0x3240,
    0x3600, 0xF6C1, 0xF781, 0x3740, 0xF501, 0x35C0, 0x3480, 0xF441,
    0x3C00, 0xFCC1, 0xFD81, 0x3D40, 0xFF01, 0x3FC0, 0x3E80, 0xFE41,
    0xFA01, 0x3AC0, 0x3B80, 0xFB41, 0x3900, 0xF9C1, 0xF881, 0x3840,
    0x2800, 0xE8C1, 0xE981, 0x2940, 0xEB01, 0x2BC0, 0x2A80, 0xEA41,
    0xEE01, 0x2EC0, 0x2F80, 0xEF41, 0x2D00, 0xEDC1, 0xEC81, 0x2C40,
    0xE401, 0x24C0, 0x2580, 0xE541, 0x2700, 0xE7C1, 0xE681, 0x2640,
    0x2200, 0xE2C1, 0xE381, 0x2340, 0xE101, 0x21C0, 0x2080, 0xE041,
    0xA001, 0x60C0, 0x6180, 0xA141, 0x6300, 0xA3C1, 0xA281, 0x6240,
    0x6600, 0xA6C1, 0xA781, 0x6740, 0xA501, 0x65C0, 0x6480, 0xA441,
    0x6C00, 0xACC1, 0xAD81, 0x6D40, 0xAF01, 0x6FC0, 0x6E80, 0xAE41,
    0xAA01, 0x6AC0, 0x6B80, 0xAB41, 0x6900, 0xA9C1, 0xA881, 0x6840,
    0x7800, 0xB8C1, 0xB981, 0x7940, 0xBB01, 0x7BC0, 0x7A80, 0xBA41,
    0xBE01, 0x7EC0, 0x7F80, 0xBF41, 0x7D00, 0xBDC1, 0xBC81, 0x7C40,
    0xB401, 0x74C0, 0x7580, 0xB541, 0x7700, 0xB7C1, 0xB681, 0x7640,
    0x7200, 0xB2C1, 0xB381, 0x7340, 0xB101, 0x71C0, 0x7080, 0xB041,
    0x5000, 0x90C1, 0x9181, 0x5140, 0x9301, 0x53C0, 0x5280, 0x9241,
    0x9601, 0x56C0, 0x5780, 0x9741, 0x5500, 0x95C1, 0x9481, 0x5440,
    0x9C01, 0x5CC0, 0x5D80, 0x9D41, 0x5F00, 0x9FC1, 0x9E81, 0x5E40,
    0x5A00, 0x9AC1, 0x9B81, 0x5B40, 0x9901, 0x59C0, 0x5880, 0x9841,
    0x8801, 0x48C0, 0x4980, 0x8941, 0x4B00, 0x8BC1, 0x8A81, 0x4A40,
    0x4E00, 0x8EC1, 0x8F81, 0x4F40, 0x8D01, 0x4DC0, 0x4C80, 0x8C41,
    0x4400, 0x84C1, 0x8581, 0x4540, 0x8701, 0x47C0, 0x4680, 0x8641,
    0x8201, 0x42C0, 0x4380, 0x8341, 0x4100, 0x81C1, 0x8081, 0x4040,
};
```

Picture 5.5.8a CRC table

After that, A CRC function is defined in order to return the CRC values for signal transfer. (See Picture 5.5.8b)

```
- (unsigned short)crc16: (NSData*) data
{
    unsigned int    crc;

    crc = 0xFFFF;

    uint8_t byteArray[[data length]];
    [data getBytes:&byteArray];

    for (int i = 0; i<[data length]; i++) {
        Byte byte = byteArray[i];
        crc = (crc << 8) ^ crc16table[((crc >> 8 ^ byte) & 0xFF)];
    }

    return crc;
}
```

Picture 5.5.8b Function defined for generating the CRC values

5.6 Test plan and test case

Test plan for the iPhone application has been divided into unit test, integrated/system test, user acceptance test, performance test and installation test. At the development stage, unit testing and integrated/ system testing are implemented as to check the processing and output controls of the applications. After finishing all the functions required, user acceptance test, performance test and installation test will be conducted to check of it is portable for non-technical user.

5.6.1 Unit Test

Unit test in this project is to test and verify each function independently. It is to assure that each key function works properly and meets the requirement defined.

5.6.2 Integration Test/System test

After combining and adding different functions in to a single project together, the integration test will be taken. Bottom up testing approach will be chosen as to be sure that the key functions in lower layer have been tested sufficiently.

5.6.3 User Acceptance test

The user acceptance test will be employed after the integration test as to collect the feedbacks from the application before release. Some Phone users may be invited to test the application referring to the user acceptance test documentation.

5.6.4 Performance test

Performance test is used to verify the data transmission speed of the application using voice signals. This is to test if it takes long time for transmitting a long number of digits.

5.6.5 Installation test

Installation test is to install the eWallet application different models of IOS devices like iPhone, iPad and iPod as to test if the interface and functions perform normally.

5.6.6 Test case and Result on transmitting singals

Unit Testing 1. Generating voice signals			
Case no.	Scenarios:	Expected Results:	Actual Results:
1.1	1. Input one digit , i.e. “1” 2. Press “Request Payment” button	A voice in 5600 Hz is generated.	A voice in 5600 Hz is generated successfully.
1.2	1. Input one digit , i.e. “9” 2. Press “Request Payment” button	A voice in 4600 Hz is generated.	A voice in 4600 Hz is generated successfully.
1.3	1. Input 2 different digits , i.e. “12” 2. Press “Request Payment” button	Voices in 5600 Hz and 1800 Hz are generated.	Voices in 5600 Hz and 1800 Hz are generated successfully.
1.4	1. Input 2 same digits , i.e. “11” 2. Press “Request Payment” button	Voices in 5600 Hz and 5800 Hz are generated.	Voices in 5600 Hz and 5800 Hz are generated.
1.5	1. Input 2 same digits , i.e. “55” 2. Press “Request Payment” button	Voices in 3000 Hz and 3200 Hz are generated.	Voices in 3000 Hz and 3200 Hz are generated.
1.6	1. Input 3 different digits , i.e. “123” 2. Press “Request Payment” button	Voices in 5600 Hz, 1800 Hz and 2200 Hz are generated.	Voices in 5600 Hz, 1800 Hz and 2200 Hz are generated.
1.7	1. Input 3 digits (2 consecutive digits are the same) , i.e. “122” 2. Press “Request Payment” button	Voices in 5600 Hz, 1800 Hz and 2000 Hz are generated.	Voices in 5600 Hz, 1800 Hz and 2000 Hz are generated.
1.8	1. Input 3 digits (2 digits are the same) , i.e. “121” 2. Press “Request Payment” button	Voices in 5600 Hz, 1800 Hz and 5800 Hz are generated.	Voices in 5600 Hz, 1800 Hz and 5800 Hz are generated.
1.9	1. Input 3 same digits, i.e. “111” 2. Press “Request Payment” button	Voices in 5600 Hz, 5800 Hz and 5600 Hz are generated.	Voices in 5600 Hz, 5800 Hz and 5600 Hz are generated.

Unit Testing 2. Recognizing voice signals			
Case no.	Scenarios:	Expected Results:	Actual Results:
2.1	1. Press “Listen” button 2. Collect the signal of 1 single digit, i.e. “1”	1. Frequency of 5600 Hz is detected. 2. Digit “1” is shown in the label of input.	1. Frequency of 5600 Hz is detected. 2. Digit “1” is shown in the label of input.
2.2	1. Press “Listen” button 2. Collect the signal of 1 single digit, i.e. “9”	1. Frequency of 4600 Hz is detected. 2. Digit “9” is shown in the label of input.	1. Frequency of 4600 Hz is detected. 2. Digit “9” is shown in the label of input.
2.3	1. Press “Listen” button 2. Collect the signal of 2 different digits, i.e. “12”	1. Frequencies of 5600 Hz and 1800 Hz are detected. 2. Digits “12” are shown in the label of input.	1. Frequencies of 5600 Hz and 1800 Hz are detected. 2. Digits“12” are shown in the label of input.
2.4	1. Press “Listen” button 2. Collect the signal of 2 same digits, i.e. “11”	1. Frequencies of 5600 Hz and 5800 Hz are detected. 2. Digits “11” are shown in the label of input.	1. Frequencies of 5600 Hz and 5800 Hz are detected. 2. Digits “11” are shown in the label of input.

5.7 Experiment on testing frequencies that are difficult to detect

During the development of the app, it is noticed that there were specific frequencies that may not be received well when they were transmitted between the two devices. As a result, some tests were implemented to find out which frequencies were not transferred well.

<u>The environment setting for the experiment:</u>

In the experiment, each frequency signal was transferred under two different environments. The first one was a silent place. The signals were transferred inside a room where no other electric device like TV, fans or radio was turned on but still contains little domestics noise.

The second environment was also inside a room with domestic noises. But an electric fan was turned on that time. The electric fan was placed half a meter away from the iPad and Mac Book used for transferring signals. The source of the fan's noises includes aerodynamic noises. And a previous experiment for collecting the fan noises was conducted and found on Internet. In that experiment, the diameter of the electric fan used was 120mm. It contains 7 blades, 4 struts for motor mounting and operates at 13V. A microphone was used 1 m away from the upstream side of the fan. (Wikibooks, 2013). The result of it is as below: (See Figure 5.7)

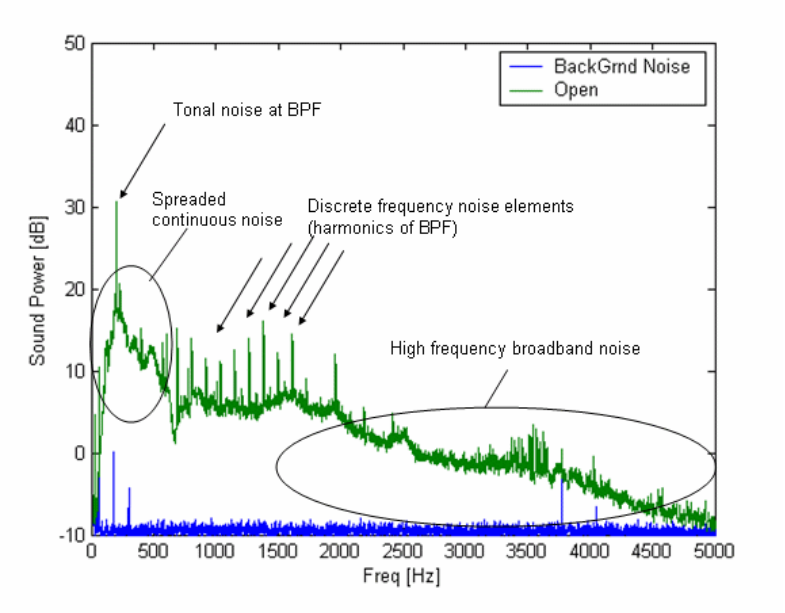
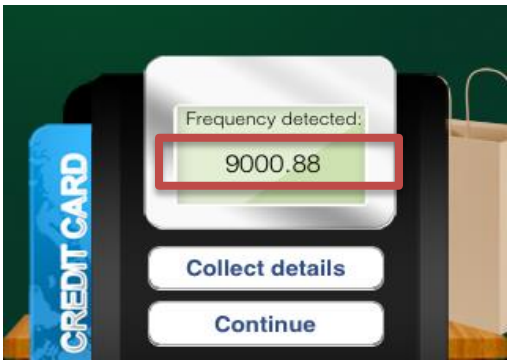


Figure 5.7 The loudness spectrum of electric fans

As the frequencies generated from the fans were quite similar to the frequencies this projected used for data transfer, an electric fan was also used as the background noise for our experiment.

The definitions of the performance column in the results of the experiment:

Good - The frequency signals detected were stable. It means that the frequency signal detected was very accurate under the two environments. The frequency label was able to show the discrete frequency of the signal consecutively. See the picture below, the label clearly showed "9000.88" as the frequency detected.



Moderate - The frequency signals detected were sometime unstable. It means that the frequency signal detected was not very accurate in the noisy environments. The frequency label was not able to show the discrete frequency of the signal consecutively. However, the frequency signals could still be detected correctly.

Fair - The frequency signals detected were unstable. It means that the frequency signal detected was not accurate in the noisy environments. The frequency label showed the wrong frequency of the signal detected. The frequency signals could not be detected correctly.

Below are the tables of the results for the experiment on transferring signal under different environment:

Text	Times	Frequency assigned	Frequency Detected in a silent place	Frequency Detected under the noise of an electric fan	Performance
0	1st	1000 Hz	990.52 – 990.53	990.52 – 990.53	Good
	2nd	1200 Hz	1205.86	1205.86	Good
1	1st	1400 Hz	1399.66	1378.12	Fair
	2nd	1600 Hz	1593.46	1614.99	Fair
2	1st	1800 Hz	1808.79	~1808.79	Moderate
	2nd	2000 Hz	2002.58 - 2002.59	2002.58 - 2002.59	Good
3	1st	2200Hz	2196.39	2196.39	Good
	2nd	2400 Hz	2390.19	~2390.19	Moderate
4	1st	2600 Hz	2605.52	~2605.53	Moderate
	2nd	2800 Hz	2799.32	2799.32	Good
5	1st	3000 Hz	2993.11-2993.12	2993.11-2993.12	Good
	2nd	3200 Hz	3208.45	3208.45	Good
6	1st	3400 Hz	3402.25	~3402.25	Moderate
	2nd	3600 Hz	3596.04	3596.04	Good
7	1st	3800 Hz	3789.84	3789.84	Good
	2nd	4000 Hz	4005.17- 4005.18	4005.17- 4005.18	Good

Text	Times	Frequency assigned	Frequency Detected in a silent place	Frequency Detected under the noise of an electric fan	Performance
8	1st	4200 Hz	4198.97	4198.97	Good
	2nd	4400 Hz	4392.77	4392.77	Good
9	1st	4600 Hz	4608.11	4608.11	Good
	2nd	4800 Hz	4801.90	~4801.90	Moderate
Start	1nd	9000 Hz	9000.88	9000.88	Good
	2nd	9200Hz	9194.68	9194.68	Good
End	1nd	9200 Hz	9194.68	9194.68	Good
	2nd	9000Hz	9000.88	9000.88	Good
Separator	1nd	9400Hz	9410.00 -9410.10	9410.00 -9410.10	Good

The frequency signals with “Moderate” means that the signals in particular frequency were sometimes not accurate. So, the use of the particular frequency is not preferred. The frequency signals with “Fair” means that the signals in particular frequency was not accurate under noise. So, the use of the particular frequency should be avoided.

5.8 Experiment on testing the loudness of frequency signals

In this experiment, a Mac book is used as the sender to generate signals in different frequencies. An iPad installed with an app names ”Max_db_Time” was used as the receiver of the signals sent. The iPad was placed 15 cm away from the Macbook as to imitate the actual distance between merchants and clients.

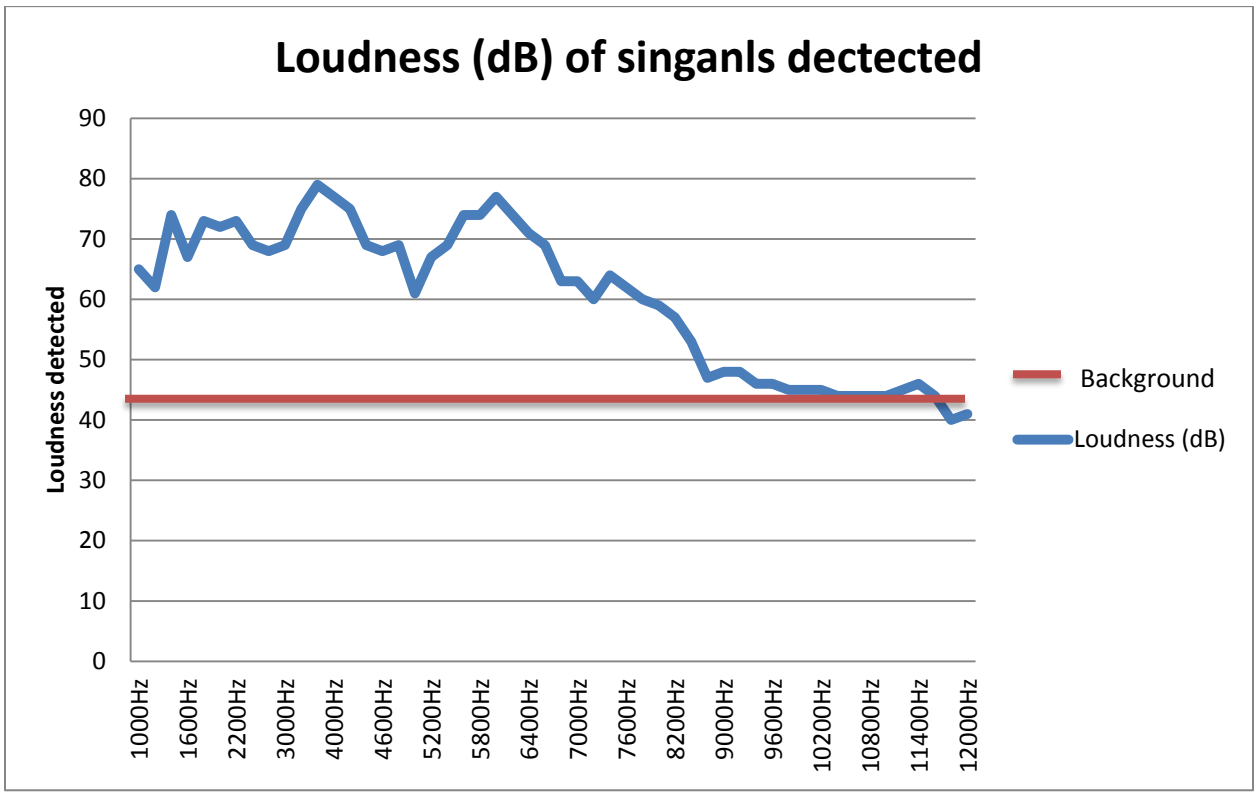
Before the experiment, it is detected that the loudness of background noises that come from vehicles that passed through the road and other domestic noises was around 42 to 50 dB.

And then, the window of the room was closed to keep the room silent. And the loudness of the frequency signals were detected as below:

Frequency used: (Hz)	The loudness detected (dB)	Frequency used:	The loudness detected (dB)
Background noise	42-50	**6600**	69
1000	65	**6800**	63
1200	62	**7000**	63
1400	74	**7200**	60
1600	67	**7400**	64
1800	73	**7600**	62
2000	72	**7800**	60
2200	73	**8000**	59
2400	69	**8200**	57
2600	68	**8400**	53
3000	69	**8600**	47
3200	75	**9000**	48
3400	78	**9200**	48
3600	79	**9400**	46
4000	77	**9600**	46
4200	75	**9800**	45
4400	69	**10000**	45

4600	68	**10200**	45
4800	69	**10400**	44
5000	61	**10600**	44
5200	67	**10800**	44
5400	69	**11000**	44
5600	74	**11200**	45
5800	74	**11400**	46
6000	77	**11600**	44
6200	74	**11800**	40
6400	71	**12000**	41

From the result above, a graph can be plotted like this:



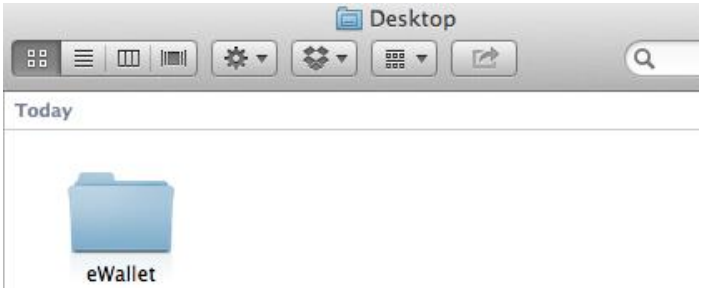
Graph 5.8 The Loudness (dB) of Signals Detected

From the graph above, it is clear that the loudness of signals with frequencies higher than 9600Hz were very close to that of the background noises. These signals maybe covered by the background noises that are not favorable and preferred to be used for transmitting signals.

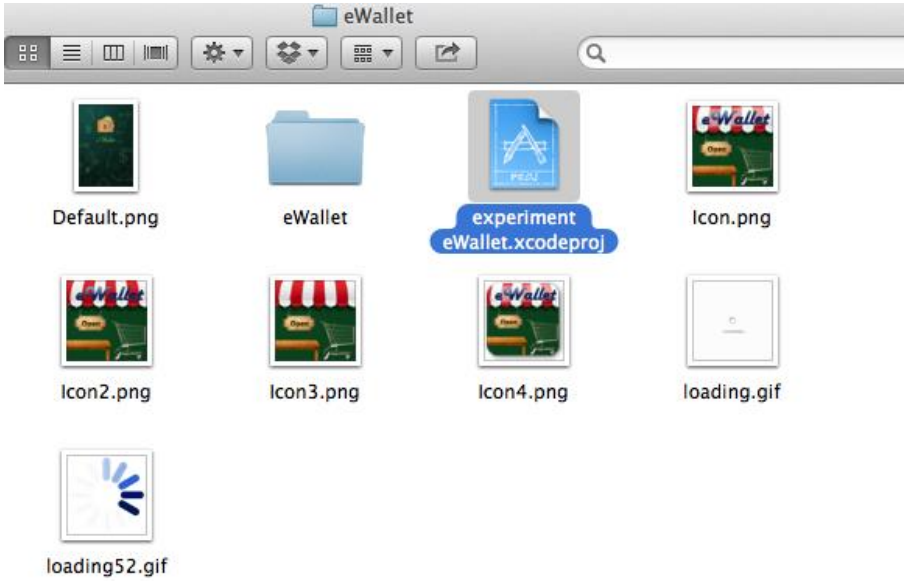
6. How to setup this app with your Xcode

If you want to set up the app to your own devices, you have to use Xcode as the development tool in your Mac Book. The followings are the steps on how to install the app into your own devices:

1. Download the file “ewallet.zip” and unzip it. Then, you will get a folder called “eWallet” as below:

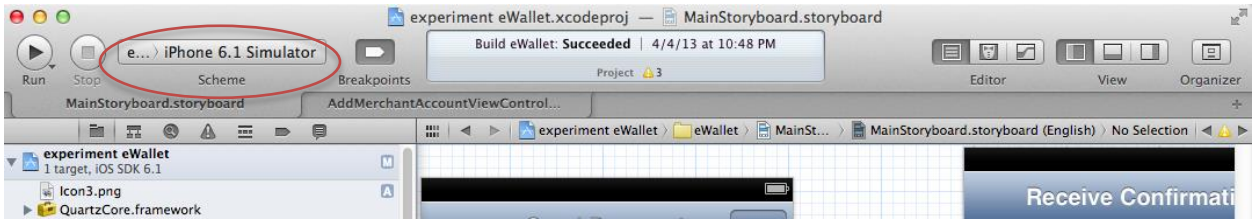


2. Double-click the “eWallet” folder. Then, you will see a file name “experiment eWallet.xcodeproj”.



3. Double-click the file “experiment eWallet.xcodeproj” and Xcode will be launched automatically.

4. Choose the target device you would like to install the app on the Toolbar (circled). If you cannot find the Toolbar like the picture below, you should choose View -> Show Toolbar in the menu bar of Xcode



If you cannot not find your own device, you can follow the steps below:

i. Connect your device to the Mac Book.

ii. A window will pop up to show the details and the name of the connected device. Then, click the “Set as development tool” button in the window to set the device as a development tool.

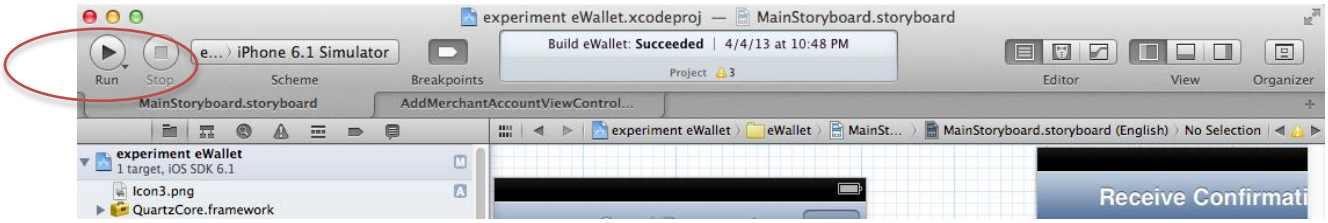
iii. Click the button again and the IOS developer website will be popped open.

iv. Login the IOS developer website and click open to launch the “Development Provisioning Assistant”.

v. Register the device by following the steps in the “Development Provisioning Assistant”.

vi. Then, the device is ready to be used in Xcode.

5. Choose the registered device and click the “Run” button to install the app into your selected devices. Then, you will see the app is running on your device.



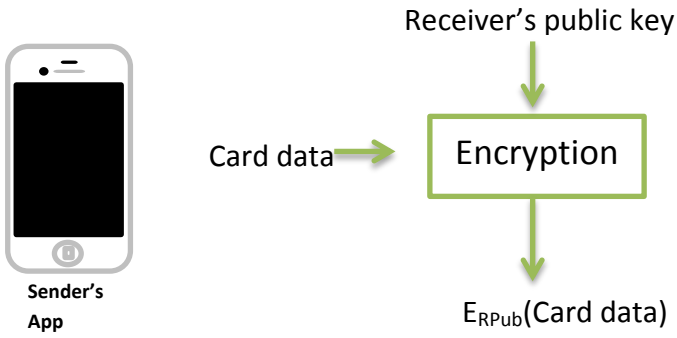
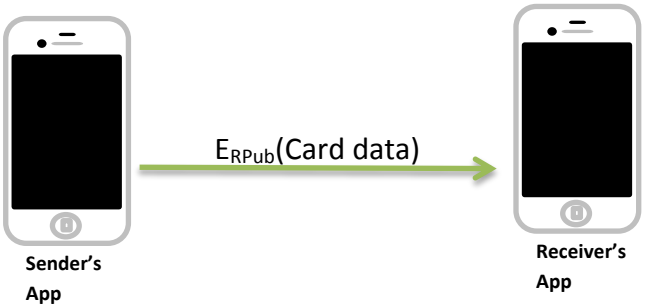
7. Future development

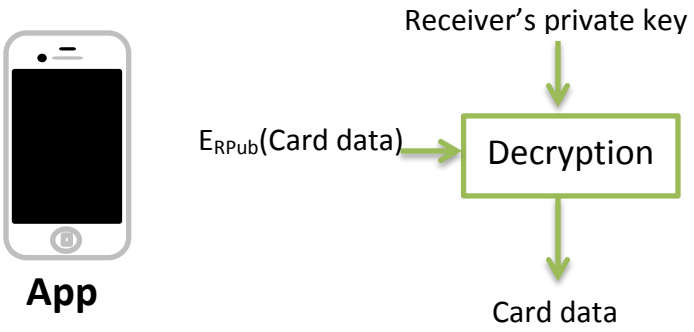
In this project, the voice signals used for data transfer are not encrypted. So, the other users using this same app may be able to receive the signals if the devices are placed close enough.

Although it is not difficult to match the signal frequencies to all characters used. The app only supports sending the signals of digit inputs (0 – 9) because of the limited time of this project. So, the app is unable to sending the encrypted data as they may contain other characters beyond the digits.

As a result, the further development of this project was to match the signal frequencies to more characters first. Then, we can encrypt the data to be sent by using the receiver's public key. The cipher will then be sent to the receiver. In this transmission process, RSA could be considered to use for the encryption. (See table 7)

Table 7

Steps:	Description
1. Sender's app encrypts the data to be sent with the receiver's public key	
2. Sender's app sends the voice signals of the cipher	

Steps:	Description
3. The receiver's app decrypts the received signals with his/her private key	

8. Appendix

A. References

United States National Institute of Standards and Technology (NIST) (2001, Nov 26). "Announcing the ADVANCED ENCRYPTION STANDARD (AES)". *Federal Information Processing Standards Publication 197.* Retrieved April 3, 2013 from http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

Ben (2010). iPhone 4 Audio and Frequency Response Limitations. *Faberacoustical.* Retrieved on Oct 27, 2012, from http://blog.faberacoustical.com/2010/ios/iphone/iphone-4-audio-and-frequency-response-limitations/

CARR, A.(2011). VeriFone Talks Smack: Does Square Actually Have Security Flaws? *Fast Company.* Retrieved Oct 14, 2012, from http://www.fastcompany.com/1736916/verifone-talks-smack-does-square-actually-have-security-flaws

Dubinsky, Z. (2010). New credit cards pose security problem: Hacker shows CBC how to crack 'contactless' MasterCard. *CBCnews.* Retrieved Oct 14, 2012, from http://www.cbc.ca/news/technology/story/2010/05/31/f-rfid-credit-cards-security-concerns.html

Gallagher, M. (2010). An iOS tone generator (an introduction to AudioUnits). *Matt Gallagher, programming – Cocoa with love.* Retrieved on Jan 31, 2013 from http://www.cocoawithlove.com/2010/10/ios-tone-generator-introduction-to.html

Gallagher, M. (2010). Sliding UITextFields around to avoid the keyboard. *Matt Gallagher, programming – Cocoa with love.* Retrieved on Apr 4, 2013 from http://www.cocoawithlove.com/2008/10/sliding-uitextfields-around-to-avoid.html

Gotmerchant.com (2012). The Different Types of Credit Card Machines. *Gotmerchant.com*. Retrieved on Jan 30, 2012 from http://www.gotmerchant.com/types_of_credit_card_machines.php

Inman, Kurt (2013). What Is a Secure Hash Algorithm? *wiseGEEK*. Retrieved on April, 2, 2013 from http://www.wisegeek.com/what-is-a-secure-hash-algorithm.htm

Lo, Cliff (2012, March 08). Family of five arrested over Octopus card fraud. *South China Morning Post.* Retrieved on Jan 30, 2013, from http://www.scmp.com/article/994811/family-five-arrested-over-octopus-card-fraud

Reilly J. (2012, April 17). Aspiring model, 23, used stolen credit cards to pay for thousands of pounds of cosmetic surgery. *MailOnline.* Retrieved Nov 8, 2012, from http://www.dailymail.co.uk/news/article-2130879/Faileigh-Cooper-23-used-stolen-credit-card-details-pay-cosmetic-surgery-London-clinic.html

HowStuffWorks.com (2008). How does a magnetic stripe on the back of a credit card work? *HowStuffWork.* Retrieved Oct 24, 2012, from http://money.howstuffworks.com/personal-finance/debt-management/magnetic-stripe-credit-card.htm>

OSCARVGG (2010). How to make an iPhone App – Part 6: Saving Data. *IOS, TUTORIALS*. Retrieved April 4, 2013 from http://mobileorchard.com/how-to-make-an-iphone-app-part-6-saving-data/

Pirraglia, W. (2012). How Do Credit Card Processing Machines Work? *eHow.* Retrieved Oct 14, 2012, from http://www.ehow.com/how-does_4897435_credit-card-processing-machines-work.html

RFID Journal (2003, April 9). RFID Smart Cards Gain Ground. *RFID News*. Retrieved on Jan 30, 2013, from http://www.rfidjournal.com/article/articleview/374/1/1/

Rouse, M. (2008). DTMF (dual tone multi frequency). *SearchNetworking*. Retrieved Oct 27, 2012, from http://searchnetworking.techtarget.com/definition/DTMF

Schneier, B. (2005, February 18). Cryptanalysis of SHA-1. *Schneier on Security.* Retrieved Apr 3, 2013 from http://www.schneier.com/blog/archives/2005/02/cryptanalysis_o.html

Strickland, J. (2012). How Near Field Communication Works. *howstuffworks*. Retrieved Oct 14, 2012 from http://electronics.howstuffworks.com/near-field-communication5.htm

The MathWorks (2004). Dual-Tone Multi-Frequency (DTMF) Signal Detection. *MATLAB® 7.0.4.*Retrieved Feb 2, 2013 from http://www.mathworks.com/products/demos/signaltlbx/dtmf/dtmfdemo.html

Wikibooks (2013). Acoustics/Noise from Cooling Fans. *Wikibooks.* Retrieved Apr 12, 2013 from http://en.wikibooks.org/wiki/Acoustics/Noise_from_Cooling_Fans

Williams, Ross N. (1993). A Painless Guide to CRC Error Detection Algorithms V3.00 (9/24/96). *Rocksoft(tm) Pty Ltd*. Retrieved Apr 3, 2013 from http://www.repairfaq.org/filipg/LINK/F_crc_v31.html#CRCV_002

B. Hearable frequency table for the text representations before the experiment on testing frequencies that are difficult to detect

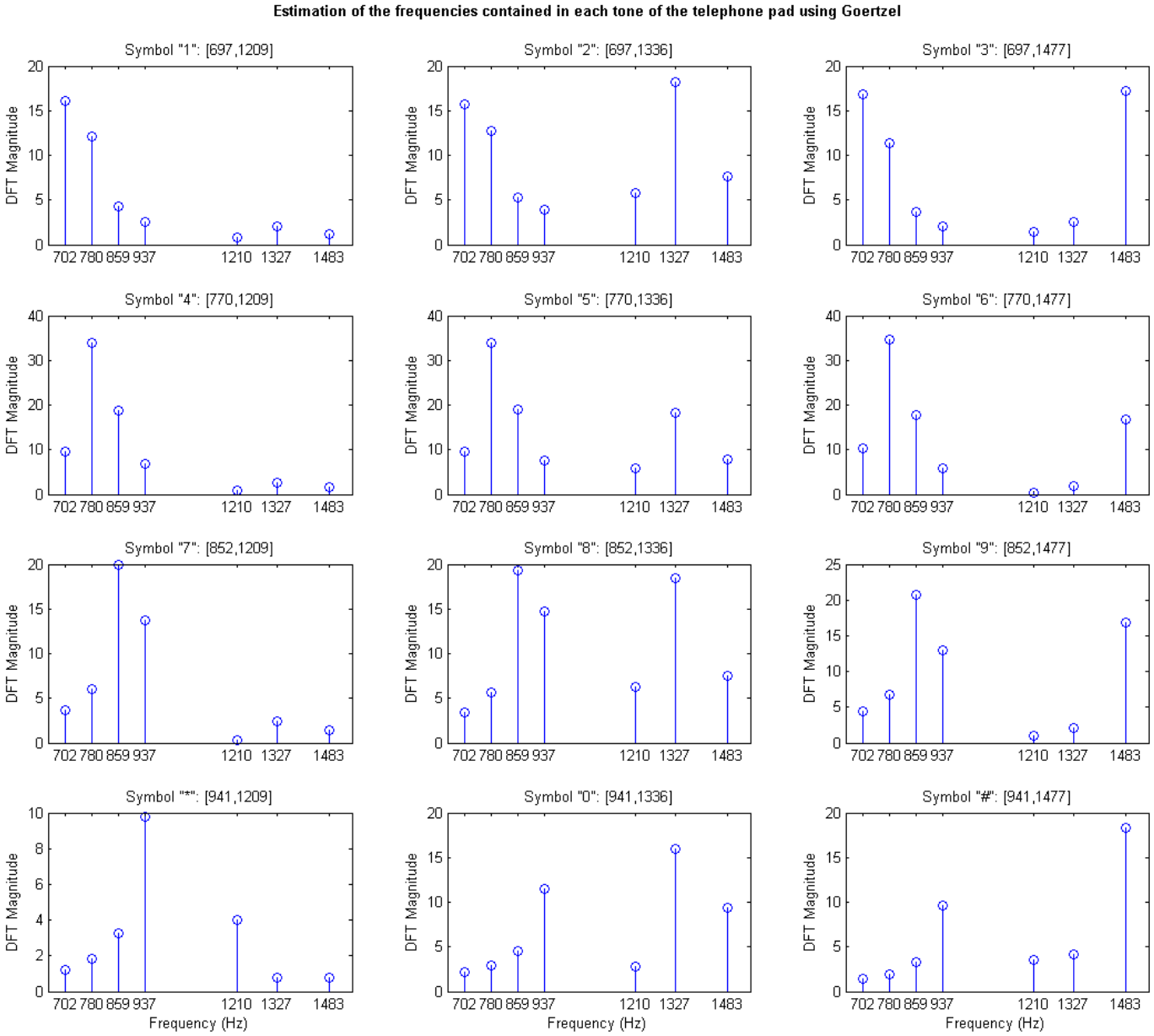
Text	Times	Frequency assigned for signal generation	Text	Times	Frequency assigned for signal generation
0	1st	1000 Hz	8	1st	4200 Hz
	2nd	1200 HZ		2nd	4400 Hz
1	1st	1400 Hz	9	1st	4600 Hz
	2nd	1600 Hz		2nd	4800 Hz
2	1st	1800 Hz	Start	1nd	9000Hz
	2nd	2000 Hz		2nd	9200Hz
3	1st	2200 Hz	End	1nd	9200 Hz
	2nd	2400Hz		2nd	9000Hz
4	1st	2600 Hz	Separator	1nd	9400Hz
	2nd	2800 Hz			
5	1st	3000 Hz			
	2nd	3200 Hz			
6	1st	3400Hz			
	2nd	3600 Hz			
7	1st	3800 Hz			
	2nd	4000 Hz			

C. Actual frequency detected after delivering between two devices

Text	Times	Frequency assigned	Frequency Detected	Text	Times	Frequency assigned	Frequency Detected
0	1st	1000 Hz	990.52 – 990.53	8	1st	4200 Hz	4198.97
	2nd	1200 Hz	1205.86		2nd	4400 Hz	4392.77
1	1st	1400 Hz#	1399.66	9	1st	4600 Hz	4608.11
	2nd	1600 Hz#	1593.46		2nd	4800 Hz*	4801.90
2	1st	1800 Hz#	1808.79				
	2nd	2000 Hz	2002.58 - 2002.59				
3	1st	2200Hz	2196.39				
	2nd	2400 Hz*	2390.19				
4	1st	2600 Hz*	2605.52				
	2nd	2800 Hz	2799.32				
5	1st	3000 Hz	2993.11-2993.12				
	2nd	3200 Hz	3208.45	Start	1nd	9000 Hz	9000.88
6	1st	3400 Hz*	3402.25		2nd	9200Hz	9194.68
	2nd	3600 Hz	3596.04	End	1nd	9200 Hz	9194.68
7	1st	3800 Hz	3789.84		2nd	9000Hz	9000.88
	2nd	4000 Hz	4005.17-4005.18	Separator	1nd	9400Hz	9410.00 - 9410.10

5800 Hz	5792.43
5600 Hz	5598.63
5500 Hz	5490.96 -5490.97
5000 Hz	4995.70-4995.71
4500Hz	4500.43 - 4500.44
3500 Hz	3509.91 - 3509.92
3000 Hz	2993.11 - 2993.12
2500 Hz	2497.85 - 2497.86
1500 HZ	1507.32-1507.33

D. Frequency contained in each tones of Dual Tones Multi Frequency



E. Hearable frequency table for the text representations after the experiment on testing frequencies that are difficult to detect

Text	Times	Frequency assigned for signal generation	Text	Times	Frequency assigned for signal generation
0	1st	1000 Hz	8	1st	4200 Hz
	2nd	1200 HZ		2nd	4400 Hz
1	1st	5600 Hz	9	1st	4600 Hz
	2nd	5800 Hz		2nd	4800 Hz
2	1st	1800 Hz	Start	1nd	9000Hz
	2nd	2000 Hz		2nd	9200Hz
3	1st	2200 Hz	End	1nd	9200 Hz
	2nd	2400Hz		2nd	9000Hz
4	1st	2600 Hz	separator	1nd	9400Hz
	2nd	2800 Hz			
5	1st	3000 Hz			
	2nd	3200 Hz			
6	1st	3400Hz			
	2nd	3600 Hz			
7	1st	3800 Hz			
	2nd	4000 Hz			

F. Monthly Logs

26 Mar 2013

In March, I continued to focus on the development of user interfaces for my iPhone app. After the reviewed with Dr. Wong, I decided to add in more functions in order to make the application more user friendly.

For example, my new design allows users to store the records of more than 1 credit card. And they can choose one for transaction. The transaction records will also be stored automatically that can be reviewed in the History page.

Besides it, I had also worked on the accuracy of transferring sound signals. E.g. add in some special signals as the start signals for transmitting signals.

27 Feb 2013

In February, I began developing the user interface of the application after dealing with the problem of duplicated frequencies and project integration. For example, I set up the variable that to be passed between different view controllers and the flow of different functions.

Besides it, I began adding the function of user authentication as to protect the user's card. The user has to register a card account before transactions. Once the account is created, users have to log in each time for payments. But my preliminary design allows registering 1 card only, I will change this design and allow users to add more than 1 card after discussing with Dr. Wong.

26 Jan 2013

In this month, I have finished developed the key function of collecting and recognizing voice signals produced. As it is too complicated to recognize dual tone signals, Dr. Wong and I agreed to use signal tones signals instead for data transfer. As to solve the problems of duplicated signals, we decided to define 2 frequencies for each character in case the characters are used consecutively.

After developing the key functions of producing and collecting sound signals, I began to design the user interface for my project. In the beginning, I tried to combine the classes for the key functions into one signal project. Then, I carried on design change of user's view between different pages of the apps. I would start the part of authentication later.

27 Dec 2012

In this month, I have produced dual-tone sounds by using the app. The app is apple to produce sounds according to the input and each sound is combined with 2 different frequencies.

Then, I begin to develop the other part of the app that requires the iPhone to collect and recognize the sound frequencies. However, I met a bottleneck, as the app is not able to detect more than 1 frequency at a time. The mechanism for recognizing dual-tones frequencies seems to be very complicated. As a result, I have discussed it with Dr. Wong. And Dr. Wong agreed that we might develop the app for recognizing single tones only.

26 Nov 2012

In the early November, I began to try programming in objective C through the use of X-code.

In the beginning, I tried to follow the instruction on the apple website to create a simple iPhone app successfully.

Then, I carried on searching for the resources on the Internet for the libraries that can be used for producing tones in different frequencies through the phone microphones.

After it, I created a simple app that is able to produce sounds after the pressing of the key.

However, I have not yet be able make the phones to produce a sound with multiple tones at the same.

26 Oct 2012

The new topic is about the credit card transaction via iPhone. The direction of the topic is to make use of sounds for data transfer so that no swiping of credit card is need for purchase.

In the past month, I have searched many technologies used daily for the card transactions. Traditionally, clients pay for their purchase by swiping cards. Recently, there are new ways for the payments such as the use of RFID, NFC and the square wallet.

I am also reviewing the theory of DTMF that assigns each number of digit different frequencies for dialing phone calls.

G. Source Codes

You can download the source codes from the link below:

https://www.dropbox.com/s/9f5ibf9uupqeumb/eWallet.zip