Smartphone-based smartcard reader

Final Report

T.D. Stewart
13002610

Submitted as partial fulfilment of the requirements of Project EPR402
in the Department of Electrical, Electronic and Computer Engineering
University of Pretoria

November 2017

Study leader: Mr. D. Ramotsoela

Part 1. Preamble

This report describes the work I did in designing and implementing a smartphone based smart card reader system. The system was built using predominantly first principle and discrete components

Project proposal and technical documentation
This main report contains a copy of the approved Project Proposal (Part 3 of the report) and technical documentation (Part 5 of the report). The latter provides details of the circuit diagrams, component datasheets, simulations, software flow diagrams, and software code. I have included this appendix on a CD that accompanies this report.

Project history
This project is not an extension of any other previous projects. This project draws inspiration from similarly available commercial products. The design and implementation are my own and done predominantly using first principle techniques.

Language editing
This document has been language edited by a knowledgeable person. By submitting this document in its present form, I declare that this is the written material that I wish to be examined on.

My language editor was Mr. Richard Princen

_____ _____
Language editor signature *Date*

Declaration

I, Thomas David Stewart understand what plagiarism is and have carefully studied the plagiarism policy of the University. I hereby declare that all the work described in this report is my own, except where explicitly indicated otherwise. Although I may have discussed the design and investigation with my study leader, fellow students or consulted various books, articles or the internet, the design/investigative work is my own. I have mastered the design and I have made all the required calculations in my lab book (and/or they are reflected in this report) to authenticate this. I am not presenting a complete solution of someone else.

Wherever I have used information from other sources, I have given credit by proper and complete referencing of the source material so that it can be clearly discerned what is my own work and what was quoted from other sources. I acknowledge that failure to comply with the instructions regarding referencing will regarded as plagiarism. If there is any doubt about the authenticity of my work, I am willing to attend an oral ancillary examination/evaluation about the work.

I certify that the Project Proposal appearing as the Introduction section of the report is a verbatim copy of the approved Project Proposal.

_____ _____
T.D. Stewart Date

TABLE OF CONTENTS

LIST OF ABBREVIATIONS

PIN	Personal identification number
HMI	Human-machine interface
FPGA	Field-programmable gate array
RISC	Reduced instruction set computing
ARM	Advanced RISC machine
EEPROM	Electrically erasable programmable read-only memory
LCD	Liquid crystal display
EMV	Europay, MasterCard, Visa
ISO	International Organization for Standardization
SIM	Subscriber identity module
ETU	Elementary time unit
ATR	Acknowledge to reset
RSA	Rivest–Shamir–Adleman
DES	Data encryption standard
3DES	Triple DES
XOR	Exclusive OR gate
PLL	phase locked loop
UART	Universal asynchronous receiver-transmitter
GPIO	General purpose input/output
PWM	Pulse-width modulation
LED	Light emitting diode
SPP	Serial peripheral interface
RAM	Random access memory
JTAG	Joint test action group
GUI	Graphical user interface
XML	Extensible Markup Language
UML	Unified modelling language

Part 2. Summary

This report describes the work done in the design and construction of a smartphone based smart card reader for e-commerce purposes, with a focus on security principles.

What has been done

An investigation on smart card design, operation and communication protocols. Investigation and development of a smartphone application. Hardware design from first principles of a battery powered smart card terminal with a keypad and liquid crystal display with firmware code developed from first principle. An Android smartphone application developed to facilitate financial transactions and keep a history of the transactions. A Bluetooth wireless interface with communication protocol for interfacing the terminal and smartphone application. A software implementation of 3DES encryption as a Python simulation and an implementation for operation on 8-bit microprocessor in C. Design and construction of a smart card interface from first principle to provide the operating requirements and communication protocols for interfacing with a smart card device. Design and construction of an 8-bit smart card with 4096 bytes of EEPROM memory from first principle with accommodating firmware and encryption code. A breakdown of the features of the final system is detailed below followed by a picture of the final system.

- Set transaction name and cost with smartphone application
- Connect to terminal from application
- Verify transaction details on terminal and accept charge
- Successfully connect smart card to terminal
- Enter PIN of length 4 – 6 digits long
- Clear PIN for re-entry
- Send PIN for verification on smart card
- Smart card determines PIN correctness
- Smart card verifies sufficient and insufficient balance
- Terminal and smartphone receive correct transaction result
- Application stores transaction details correctly
- Application can successfully clear transaction history
- Cancel transaction from application and terminal

What has been achieved
The system can successfully emulate a financial transaction. The merchant uses the smartphone application to request a payment, the customer uses the smart card terminal to accept the payment by connecting the smart card and entering the cards PIN. The smart card will use the PIN to decrypt the smart card memory to verify the PIN is correct, check the cards balance, deduct the funds, pay the merchant, and encrypt the new card balance. The merchant receives confirmation of the payment and the transaction is added to a payment history on the smartphone application.

Findings
An important discovery was that 3DES encryption is a hardware designed encryption system and that software implementations of the system require high amounts of flash memory to operate fully, but can operate with an 8-bit variable limitation.

Contribution
A variety of new hardware and software systems were worked on for the implementation of the system. The system is made up of the smart card terminal, smartphone application, and smart card.

The smart card terminal and smart card made use of new hardware, specifically a keypad and an Atmel microcontroller. The microcontroller used a new programming and debugging system and the Atmel microcontroller programming had to be learnt and implemented. Smart card protocol theory had to be learnt so successful smart card interfacing could be achieved. Some of the hardware design used some commercial products for inspiration but all design and implementation were done from first principle. Most system design and software development were by the student, with some software aspects using libraries provided by the hardware development platform, whilst others being built from first principle.

Python had to be learnt with help from a friend to develop a simulation of the encryption system. 3DES encryption had to be studied more in depth for a software implementation in both Python and C.

Android Java native interface and XML had to be learnt to develop the smartphone application. Investigation on wireless interface systems and an investigation on software implementations for the development of the wireless interface. The use of Bluetooth hardware and code implementation for use with a microcontroller and a smartphone application.

Part 3. Project identification: approved Project Proposal

This section contains the problem identification in the form of the complete approved Project Proposal, unchanged from the final approved version.

1. Problem statement

Motivation. Smart cards are plastic cards that have an embedded microprocessor. These smart cards provide a cheap tool for users to securely identify themselves and use services such as access control, financial transactions, and cell phone airtime and data [1]. Financial payments will be the focus of this project and they commonly use contact smart cards also known as chip smart cards. These cards require input of a personal identification number (PIN) to compare with the PIN that is stored on the microprocessor, to verify that the payment is being made by the authorised account holder.

Merchants use terminals to accept smart card payments for goods or services purchased. These terminals require users to enter their PIN and verify that it is the same PIN stored on the chip using cryptographic algorithms, this identifies the user while maintaining the secrecy of the PIN. Merchants insert the buyers smart card into the terminal and enter the payment details, they then pass the terminal to the buyer to enter their PIN to validate the payment.

There are disadvantages with these terminals as they are expensive and are often part of a financial service scheme that charges transaction fees for payments such as Visa, MasterCard and PayPal. These schemes are aimed specifically at more complex financial transactions between up to four independent parties (buyer, seller, buyer bank, seller bank) and charge fees to some of the parties and share these fees amongst the other parties according to pre-specified rules. Another disadvantage of these terminals is that they need to interface with computers to access other business software solutions.

The problem to be addressed in this project is to develop a smart card terminal that interfaces with a smartphone device, rather than a computer, and accesses a smartphone application that allows for financial transactions. The aim is to charge no fees on the payments, but retain the chip and PIN encrypted validation scheme to enable user authentication and safeguard the secrecy of the PIN.

Context. There are some commercial smart card terminals that interface with smartphones/tablets. Some examples are the Verifone e105 [2] that interfaces using the audio jack of the smartphone/tablet; the Verifone e255 [3] that uses Wi-Fi and PayPal Here [4] that uses Bluetooth.

Fee charging systems are not ideal for two party payment systems like universities-students, where food payments are done through a pre-paid account system.

Technical challenge. The most technically challenging part is the design and construction of the smart card interface as smart cards require specific voltages, current, and timing signals to operate correctly. Other technically challenging parts include the interfacing of a variety of different hardware components that each have their own specific requirements and developing security algorithms that can operate efficiently on hardware components with limited speed and memory.

Limitations. The system requires good use of processing and memory on the smart cards, as smart card unit costs increase with performance, creating implementation cost increases for bulk card purchases. Budgetary restrictions will limit the quality, speed, and memory of hardware that can be used throughout the system.

2. Project requirements

ELO 3: Design part of the project

2.1 Mission requirements of the product

The mission requirements of the system are detailed below.

- The system must be able read from and write to the chip smart card.
- The system must interface with a smartphone application.
- The smartphone application must be able to request payments from the terminal and keep a digital receipt of the transaction.
- The system must use a secure PIN system for customer payments.
- The system must connect with a smartphone via a wireless interface.
- The system must provide a human machine interface (HMI) for the customer to confirm payments and enter PINs.
- The system must be battery powered and rechargeable.

2.2 Student tasks: design

The design tasks required for the system is summarised below.

- An investigation on requirements to drive a smart card, capabilities, and communication protocols.
- The design and construction of a chip smart card interface to facilitate communication between a smart card and a processing unit such as a microcontroller or a field-programmable gate array (FPGA).
- The integration of different hardware components and the firmware to allow for correct interfacing of these components.
- An investigation into different security algorithms and coding of a security algorithm to facilitate PIN verification on 8-bit microprocessor smart cards.
- The development of a smartphone application that uses a wireless interface to external hardware.

ELO 4: Investigative part of the project

2.3 Research questions

The research questions that will be addressed in this system is summarised below.

- What is the most suitable combination of hardware components to be implemented in the system?
- What cryptographic methods are used for PIN verification in smart card systems and what advantages and disadvantages does each algorithm have?

2.4 Student tasks: experimental work

The experimental work that will be done is detailed below.

- The analysis and evaluation of cryptographic algorithms for PIN verification systems.
- The testing of hardware and software components in a systematic approach to ensure correct functionality and that specifications are met.
- An evaluation of hardware design and implementation.
- An investigation and evaluation of possible security vulnerabilities of the completed system.

3. Functional analysis

The system will consist of a smart card, a smart card terminal, and a smartphone application. A functional block diagram describing the functional units and their interconnections are shown in Figure 1.

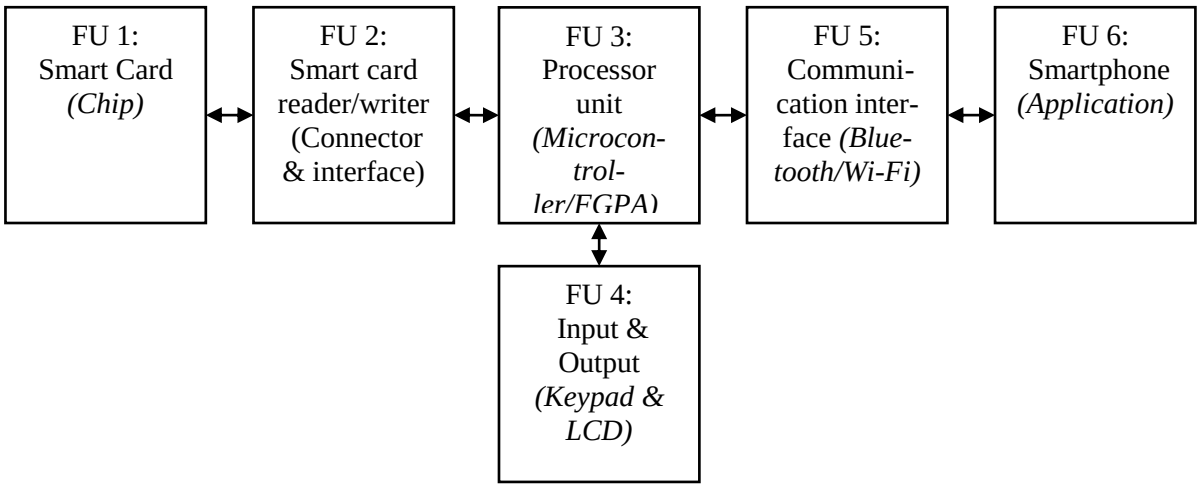

Figure 1
System functional diagram

Functional unit 1 is the smart card device. This device consists of a microprocessor and electrically erasable programmable read-only memory (EEPROM). The smart card will have firmware to operate the device and store information on the device. It will also have an algorithm to decrypt a PIN passed to it and then compare it to the PIN stored on the card. Once compared it will pass information to the terminal whether a payment can be made or not.

The system terminal consists of functional units 2 to 5. Functional unit 3 is the main processing unit of the terminal and will serve as the communication interface between functional units 2, 4, and 5. Functional unit 3 will also run the encryption algorithm that will be used when communicating with the smart card. Functional unit 2 will serve as the interface between the smart card microprocessor and the terminal. It will provide the correct voltage and current for the smart card to operate and provide the correct clock for the smart card to communicate correctly. Functional unit 2 will be built from first principle. Functional unit 4 will serve as the HMI by providing a keypad with; digits 0 to 9, clear, enter and cancel buttons. It will also provide a liquid crystal display (LCD) to provide instructions and information to the customer. Functional unit 5 will serve as a wireless interface between the smartphone device and the terminal. Functional unit 6 is the application on the smartphone which will serve as the HMI for the merchant and allow for the merchant to request payments from the terminal and receive confirmation on transaction success.

Functional unit 3 is broken down further into functional units of software as shown in Figure 2. Functional unit 3.1 will serve as the communication protocol between the terminal and the smart card. This functional unit will serve to translate data between the terminal and the smart card. Functional unit 3.2 is the main software of the terminal and will consist of all the firmware code to operate the terminal's hardware and serve as the main function for all the code. Functional unit 3.3 will be the code that handles all the I/O of the terminal (keypad & LCD). Functional

unit 3.4 will be the cryptographic algorithm code for encryption of data communication between the smart card and the terminal, specifically for the PIN verification procedure. This cryptographic algorithm will need to be designed to operate on 8-bit microprocessors since most smart cards have 8-bit microprocessors. Functional unit 3.5 will be the code that communicates with the wireless interface and will serve as the communication protocol between the smartphones and the smart card terminal.

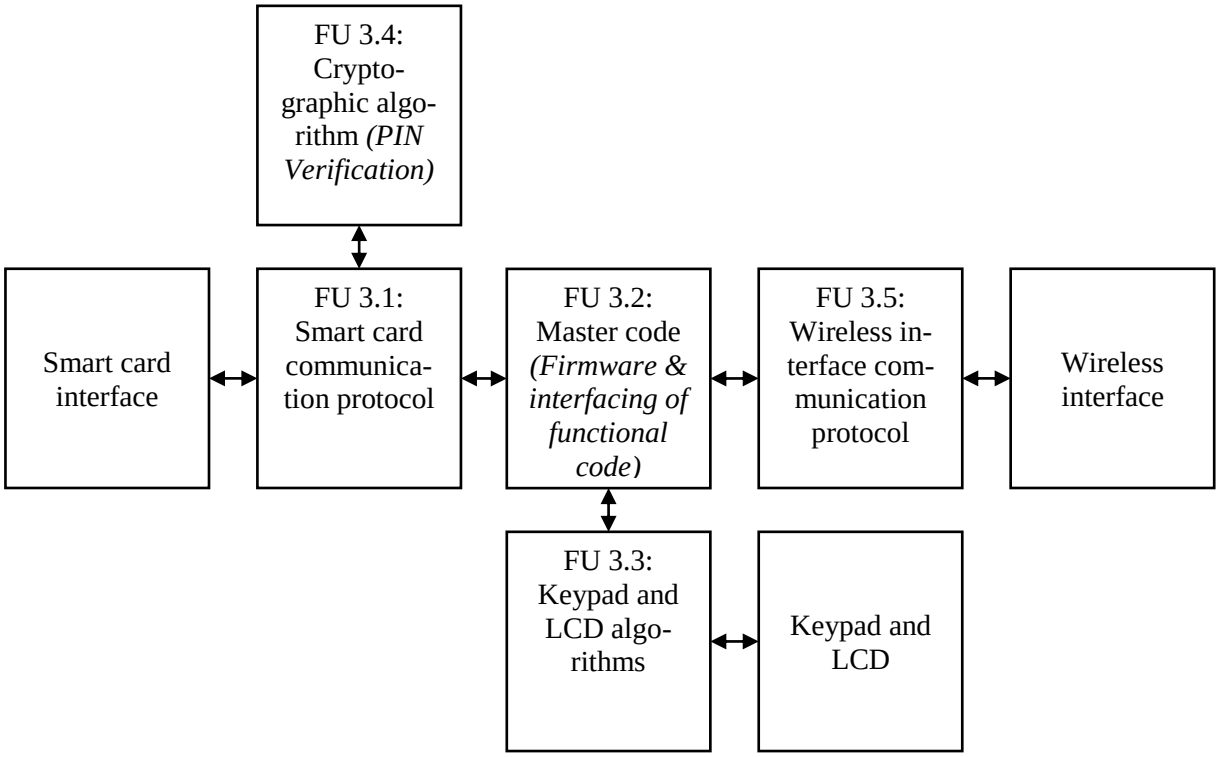

Figure 2
Breakdown of functional unit 3: Processor Unit

4. Specifications

4.1 Mission-critical system specifications

The specifications critical for the system are detailed in Table 1 below.

SPECIFICATION (IN MEASURABLE TERMS)	ORIGIN OR MOTIVATION OF THIS SPECIFICATION	HOW WILL YOU CONFIRM/ MEASURE THAT YOUR SYSTEM COMPLIES WITH THIS SPECIFICATION?
The smart card interface must provide the correct operating voltage (4.5 - 5.5 V), current (50 mA) and clock (1-5 MHz) to interface correctly with a smart card.	These are the specifications that are defined in the (Europay, Mastercard, Visa) EMV 2000 and International Organization for Standardization (ISO) 7816-3 for chip smart cards used in payment systems [5].	An oscilloscope will be connected to the interface to measure the voltage, current, and clock timing to verify correct specifications. A smart card will also be read to verify correct functionality.
The system must correctly read a string of 256 Bytes from the smart card, this data should remain consistent every time the card is read.	This specification is to ensure that there are no errors in the data being read from the smart card and that reading the data can be repeated.	A random 256 Byte set of data will saved and loaded onto the card's memory. The card will then be read from multiple times and the string compared to the original to ensure this is functioning correctly.
The smart card terminal must interface to a smartphone via a wireless interface with data rates of at least 9600 bps at a range of at least 2 metres.	This is to ensure that the device can be used at a functional range and that the data rate of the interface is not a bottle neck to the system since smart cards have a baud rate of at least 9600 as defined by the ISO 7816 standards [5].	The terminal and smartphone will be placed 2 metres apart and a file transferred. An oscilloscope will be attached to the terminal to measure the incoming and outgoing data rate.
The terminal and smart card will use an approved cryptographic algorithm as defined in the EMV 4.3 book 2 and ISO 9564-1 for PIN verification of PINs with length of four to six digits. This must operate on 8-bit microprocessor smart cards.	The standards state the types encryption methods that should be used in financial systems and that PINs are support between four and twelve digits but recommend financial services to limit the pin length to six digits [6].	The encryption algorithm will be verified using an verified version of the algorithm. The algorithm will be used to encrypt a set of data 256 Bytes large and have the verified algorithm decrypt it. It will also decrypt a message that was encrypted by the verified algorithm. This will check if encryption and decryption work correctly.
The system must be battery powered and rechargeable. The battery charge must last for a period of at least 4 hours or 100 transactions.	This is to allow for the device to remain useful for a reasonable period without being charged. This allows the device to function with a smartphone even if there is no power source available.	This will be verified by measuring the time for a full charge to deplete and the amount of transactions until a full charge depletes.

Table 1
Mission-critical system specifications

4.2 Field conditions

The field conditions where the system should be able to operate are detailed in Table 2 below.

REQUIREMENT	SPECIFICATION (IN MEASURABLE TERMS)
The system must operate indoors in temperatures ranging from 10 ℃ and 35 ℃ and humidity less than 60%.	The system is designed for merchant use indoors.
The system wireless functionality must still operate in a non-lab environment.	The system is aimed to be used at merchant stores where it will be surrounded by wireless noise from a variety of different devices.

Table 2
Field conditions

4.3 Functional unit specifications

The specifications for each functional unit in the system are given in Table 3below.

SPECIFICATION	ORIGIN OR MOTIVATION
FU 1: The smart card must have a memory capacity to store the required firmware and allow the encryption algorithm to run correctly.	This is to ensure that the smart card can hold all the required data and run the encryption algorithm.
FU 2: The smart card interface must be able to connect to the selected smart card and be able to interface with the chosen processor unit.	These values need to be correct for the smart card to operate correctly but are dependent on the model of the smart card that will be selected.
FU 3: The processor must meet processing and memory specifications to allow for integration with all other interfaces and be able to run encryption algorithms for PIN verification.	The processor needs to be able to run all the connected interfaces at the same time and must not bottleneck the system. The processor must also be able to run the required security algorithm without running out of memory or taking too long.
FU 4: The human machine interface must allow for entry of a PIN of at least four to six digits long and provide visual feedback through a LCD display.	This is to allow for user entry of the PIN with a visual feedback and confirmation.
FU 5: The wireless interface must match specifications for data rate and range requirements whilst minimising power consumption to interface with the processor unit and smartphone.	This is to ensure the wireless interface in the terminal can support the mission critical specifications for data rates and battery life requirements.
FU 6: The application must run on an Android smartphone device with Android 6.0 or higher.	This will ensure that the device operates on a variety of different Android smartphones.

Table 3
Functional unit specifications

5. Deliverables

5.1 Technical deliverables

The deliverables for the project are shown below in Table 4

DELIVERABLE	DESIGNED AND IMPLEMENTED BY STUDENT	OFF-THE-SHELF
Smart card		X
Smart card firmware	X	
Smart card connector		X
Smart card read/write interface	X	
Processing unit (Microcontroller, FPGA)		X
Keypad and LCD for human interface		X
Wireless interface (Bluetooth, Wi-Fi)		X
Smart card terminal	X	
Smart card terminal application, firmware, and interface code	X	
Security algorithm for PIN authentication	X	
Smartphone device		X
Smartphone application (user interface & functional code)	X	

Table 4
Deliverables

5.2 Demonstration at the examination

The final demonstration will consist of and be performed as follows:

1. The demonstration will start with taking a new smart card and writing a user profile to the smart card. This will be done by connecting the smart card to the reader/writer and using software to create a profile with the required information (e.g. identification number, card pin, balance).
2. The smart card will then be removed. The terminal will then be connected to a smartphone via the wireless interface.
3. A smartphone application will then be launched, and a payment will be simulated. This will be done by requesting a payment of a cost that will be entered on the smartphone.
4. When the terminal receives the payment request, it will prompt the user to connect the smart card. The terminal will ask for a PIN to be entered and check if the PIN is correct. This will be done with an incorrect PIN first and then with the correct PIN. When the correct PIN is entered it will make the payment.
5. The payment success confirmation will be shown on the terminal and ask the user to remove the smart card. A confirmation will be shown on the smartphone which will conclude a successful transaction.

6. References

[1] K. Mayes and K. Markantonakis, Smart cards, tokens, security and applications. Springer Science & Business Media, 2007.

[2] Verifone, Payware Mobile e105, 2013. [Online]. Available: http://www.verifone.co.za/media/4215671/pwm-e105-ds-a4.pdf

[3] Verifone, VERIFONE e255 FLEXIBLE MOBILITY, 2015. [Online]. Available: http://www.verifone.co.za/media/2877397/pwme255-ds-ltr.pdf

[4] PayPal. (2017) Paypal Here: Credit card reader | point of sale and mobile credit card processing. [Accessed 2017/04/2]. [Online]. Available: https://www.paypal.com/us/webapps/mpp/credit-card-reader

[5] W. Rankl and W. Effing, Smart card handbook, Third ed. John Wiley & Sons, 2004.

[6] EMVCo. (2011) Book 2: Security and key management. [Accessed 2017/04/18]. [Online]. Available: https://www.emvco.com/specifications.aspx?id=223

Part 4. Main report

1. Literature study

1.1 OVERVIEW

This project is focused on developing a smart card payment system. This system will be delivered with a working smart card and a smart card terminal that can interface with a smartphone. The system should be able to demonstrate a simple financial transaction. A merchant should be able to request a payment and its cost from their smartphone. This will then prompt the smart card terminal to request the payee's smart card to be inserted and check if the payee has the required balance. This will then prompt the payee if they want to accept this charge. If the payee accepts the charge, a PIN will be requested and be used to confirm ownership and check the balance on the card. If the PIN is correct and the balance is available, the amount will be deducted from the card and the merchant will receive a prompt on their smartphone that the payment has been made.

1.2 SYSTEM APPROACHES

The system is a payment system that uses a smart card and a smart card terminal that interfaces with a smartphone. These three required components as detailed in the overview make up the approach to the overall system already. The variations of this system are how each subsystem will be approached and achieved.

1.3 SUBSYSTEMS

The following information is an investigation and discussion of the literature and knowledge required to design and implement this system.

1.3.1 SMART CARD

There are two types of smart cards; memory cards, and microprocessor cards [1]. Memory cards are simple cards designed with simple memory functionality. They are designed for simpler systems and store relevant information in an encrypted or unencrypted form based on the application. These cards are sometimes made that once data is written to it, it cannot be changed. Microprocessor cards are more advanced by adding a microprocessor to the system. The microprocessor allows more functionality to be added to the card and allows the card to participate in the transaction process by providing the ability to perform computational functions such as encryption, decryption, and error detection. There are a variety of benefits and different use cases for each type of smart card.

There are two types of connection interfaces for these smart cards. Contact and contactless cards. Contact cards are smart cards that are inserted into a smart card reader and are used in performing a task such as a financial transaction, identifying the holder, or accessing a mobile telephone network such as a cell phone subscriber identity module (SIM). These cards are typically used with another security system such as a PIN. Contact cards use a standard for connection pads as shown below in Figure 1 taken from [2]. It is made up of eight different electrical contact areas. Voltage pin (V_{CC}) which supplies voltage power which can vary depending on the card, typically 3 or 5 volts but can differ. Ground (GND) is the ground connection. Clock (CLK) provides the operating signal for the smart card to operate. Reset (RST), and input/output (I/O) used for communicating with the device. Another voltage line (VPP), which was used for

programming the EEPROM of the smart card, but is no longer used. Then two more reserved for future use lines (RFU), these were added to support future functionality such as universal serial device (USB) protocols.

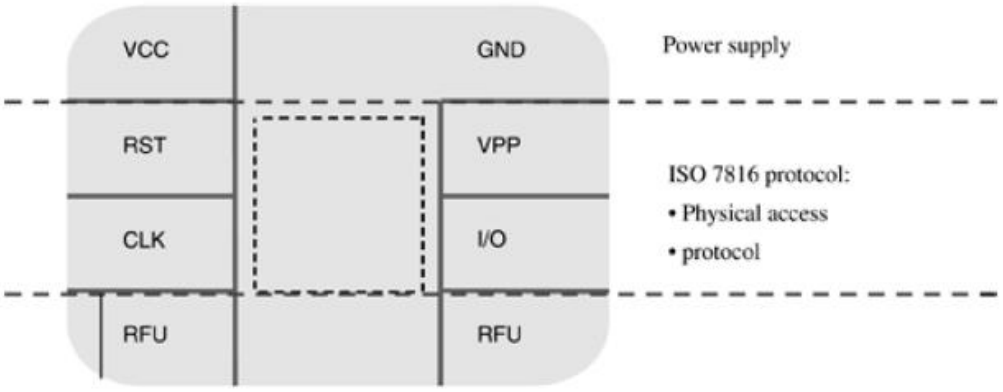

Figure 1
Contact smart card

The contactless card is very similar but instead of inserting it into a reader, the card can be held close to a radio frequency receiver and the data is transferred. The contactless ones are typically used for applications like public transport systems, ski passes, and any other application that speed of transaction is important. Contactless smart cards work with very similar connections but do it over a radio frequency interface. Figure 2 taken from [1], shows a diagram detailing how a contactless microprocessor card is structured. The communication lines shown are the same as the contact card, except the unused contact lines are not represented in the contactless system.

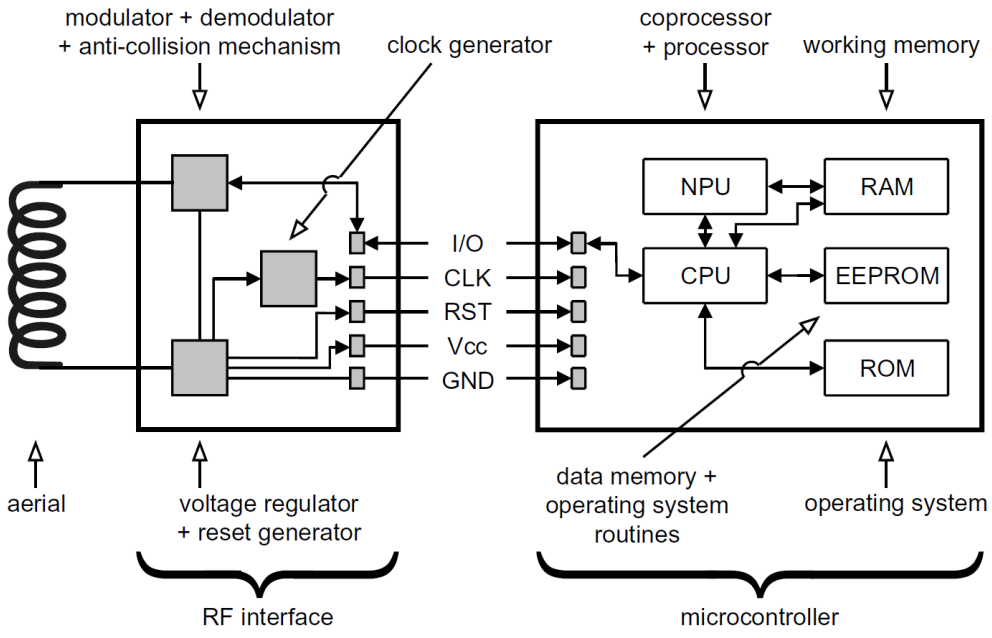

Figure 2
Contactless smart card

The choice of card used is typically based on the application. Where security is of the greatest importance, contact cards are normally the choice since sacrificing the speed of a transaction is worth the extra security level provided by other security features. Contactless cards can be used from ranges up to one metre away which is can be a security vulnerability since a person may have their card read without their knowledge. The contactless cards provide quick transaction speeds for a more relaxed security system. Some cards utilise both these types of interface, called dual-interface cards. Figure 3 taken from [3] shows the hardware structure of a micro-controller card.

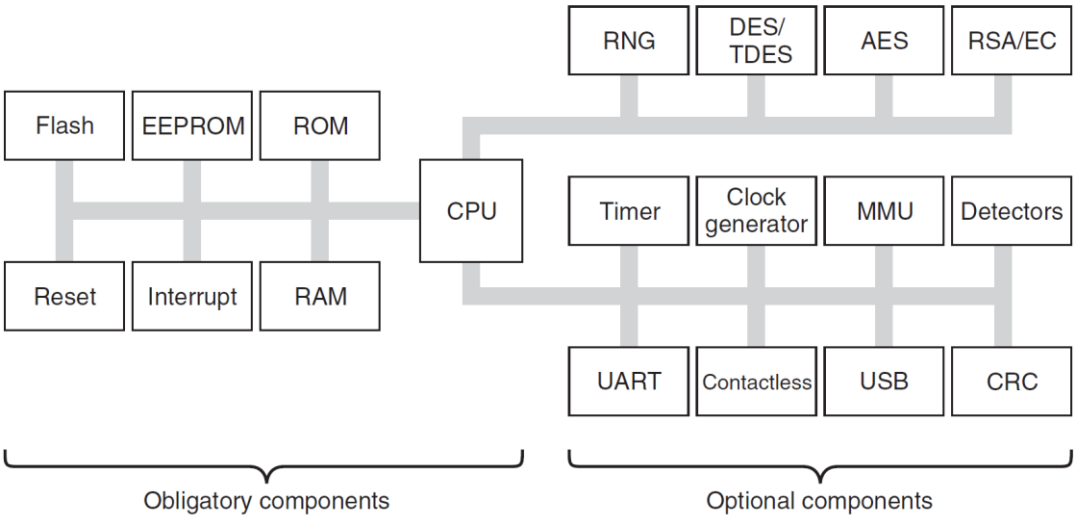

Figure 3
The hardware components and optional components of the microcontroller inside a microcontroller smart card

1.3.2 SMART CARD COMMUNICATION PROTOCOL

Microprocessor smart cards use their own communication protocol defined in the ISO 7816-3 standards [4]. Memory smart cards use I^2C to access the EEPROM on the card. The communication protocol used for microprocessor cards are the same for contact and contactless cards.

There are a variety of protocols define in ISO 7816-3. Asynchronous half duplex character transmission protocol, named T=0, and asynchronous half duplex block transmission protocol, named T=1. These two protocols are the currently supported protocols by smart cards but there are other protocols defined for future use.

These protocols use an important value named the elementary time unit (ETU). This variable is calculated by taking a predefined constant of 372 and dividing it by the clock signal that the card receives, f_i, this is shown below in Equation 1.

$$ETU = 372/f_i, \quad (1)$$

The ETU defines the nominal bit duration that the smart card uses for data transfer. The bit rate of data transfer of the smart card is the inverse of the ETU value. Typical operating frequencies for smart cards are 3.57 MHz and 4.92 MHz since these were cheap oscillating crystals available at the time. This supplies bit rates of 9596 bps and 13225 bps respectively. These bit rates are close to standard baud rates for serial communication and fall within acceptable error for

serial communication. There are some smart cards that come with an internal clock and operate with a default ETU value shown in Equation 2, this is a bit rate of 9600 bps.

$$ETU = 1/9600, \quad (2)$$

Communication with a smart card starts off with a process called answer to reset (ATR). This process facilitates the setup of communication between a smart card and the device it is interfacing with. This process starts off in T=0 protocol and determines if T=1 protocol is available and determines the optimal frequency to use. This then restarts the connection at the new settings.

There are a variety of commands in this protocol that are detailed and explained in the ISO 7816-3 specifications. These commands are not required for operation but are used to make communication simpler.

1.3.3 CRYPTOGRAPHIC ALGORITHMS

There are two cryptographic algorithms that are supported by the EMV standards for PIN use [5] which are Rivest-Shamir-Adleman (RSA) and Triple Data Encryption Algorithm (3DES). These algorithms both have their advantages and disadvantages. 3DES is a symmetric algorithm, meaning the same key is used for both encryption and decryption. RSA is asymmetric algorithm using different keys for encryption and decryption. RSA is considered a more secure algorithm in comparison to 3DES (depending on key strength) but is typically more expensive to implement [6]. Calculation complexity is different for each algorithm where RSA focuses on high number division and remainder values, where 3DES focuses on using bit manipulation such as exclusive OR gate (XOR) operations and left shifting.

Both these encryption algorithms have software and hardware implementations. RSA requires a high amount of flash memory to operate in software and requires support for high bit numbers due to high number division operations. 3DES is an algorithm that was designed as a digital hardware implementation and makes software implementations more difficult to emulate in typical software environments. 3DES doesn't require as much flash memory as RSA but does require bigger software implementations in comparison.

1.3.4 SMART CARD INTERFACE

There are a variety of smart card interfaces. The type of interface that should be used is dependent on the type of system. The interfaces are designed for a specific smart card or they are built supporting a variety of different ones typically ISO 7816 cards. The interface can consist of a variety of different components that allow for better operation with a smart card. These components can be a variety of things such as antennas, error detection circuits, phase locked loops (PLL), frequency dividers, buffers, voltage regulators, logic level shifters, and frequency generators. The requirements for interfacing depend on the smart card and the type of system that it needs to interface with. Interfaces are built to interface by a variety of methods such as USB or RS232.

Smart card interfaces can support a huge variety of different smart cards with a range of requirements, or they support a small range aimed for a specific card. The main requirements for a smart card to operate will vary depending on the smart card.

1.3.5 SMART CARD TERMINAL

Since a smart card terminal is required for transactions involving smart cards, a processing unit is needed for computations and other hardware. This could be achieved by using an advanced reduced instruction set computer (RISC) machine (ARM) but this will require development of all the other systems in the device. Another possible option is to use a microcontroller or a FPGA, these systems are already built and provide many features built-in.

1.3.6 HUMAN MACHINE INTERFACE

Many smart card systems have another layer of security to verify that the person using the card is an authorized user. There is a variety of ways to add another layer of security to this system. Many financial systems currently use a personal identification number (PIN) system where the card owner enters a secret PIN to verify that the transaction is being made by the card owner. There are some other systems that could also achieve this, such as biometric security systems such as fingerprint reading or iris detection [7]. PIN's have a disadvantage that a user's PIN can be learnt and used, if users are not careful with it. If someone finds out a user's PIN, they can then use their card to make transactions. Biometrics don't have this issue since it is extremely difficult to get around security features. Depending on the system it may be required that the terminal not receive sensitive information from that smart card, such that when the smart card and smart card terminal are made from different parties. When this is the case, microprocessor cards are used so that security algorithms can be run on the card without revealing any private information using cryptosystems. In some systems this is not necessary, such as when the terminal and cards are made by the same party.

The user output for a smart card terminal can be simple, such as using light emitting diodes (LED's) to show the transaction stage and what steps are required. A simple LCD screen can also provide the information in more detailed format. The differences in these systems are mainly cost related, where LED's are cheap, and a LCD is more expensive.

1.3.7 WIRELESS INTERFACE

The system is going to interface wirelessly with a smartphone. There are a few options to how this can be done. Bluetooth, Wi-Fi, and Wi-Fi direct are all possible options for the system. Bluetooth is a system many smartphones possess and provides simple way of connecting the terminal to a smartphone providing three different protocols based on the application [8]. It provides the benefit of being able to directly connect to the terminal in a secure fashion. Wi-Fi is another possibility since many smartphones support Wi-Fi. Wi-Fi can support much bigger ranges and data transfer speeds in comparison to Bluetooth [9]. The problem is that Wi-Fi requires a separate hardware system operate, increasing the cost and requiring the operator to have a secure Wi-Fi network to operate. Wi-Fi direct is a system that allows for direct connection with the benefits of normal Wi-Fi, but many smartphone devices don't support W-Fi direct.

1.3.8 SMARTPHONE

The system requires a smartphone to be used with the terminal. The three most common operating systems for smartphones are Android, iOS, and Windows. These operating systems have their advantages and disadvantages but not many that are relevant to this system. The choice of the smartphone is going to be focused mainly on accessibility and development complexity. The phone hardware requirements are specifically going to be set by the other selected hardware components of the system. There are a variety of software development options for each

smartphone. Each smartphone operating system has its own software library and development tools. There is a variety of cross platform tools such as Xamarin, Qt, and Sencha. The development tools all provide a variety of different benefits and challenges.

1.4 RELATED WORK

There are a few market products that are like the proposed system. The Verifone e105 [10] is a smart card payment system that interfaces with a smartphone or tablet. The device takes contact smart cards and magnetic strip cards and has a keypad to accept a user's PIN. The device interfaces with a smartphone/tablet via the audio jack of the device. This then connects to the Verifone smartphone/tablet application.

There are a few other products like the Verifone e255 [11] and PayPal Here [12]. These systems are similar both taking both contact and contactless smart cards. They both also connect to smartphones and tablet devices using their own applications. These devices are made to interface using Bluetooth. These products are different in the amount of parties used. The Verifone device is built to involve the customer, merchant, customer bank, and merchant bank. This allows the merchant to use any bank they choose. The PayPal Here requires that the merchant use PayPal as the merchant's bank.

These products are like the proposed system except are made using the customers bank as a party involved. The proposed system is to develop a simpler system where the customer will have a pre-paid value on their card with a personal PIN and will deduct the amount from the card when a payment is made.

1.5 APPLICATION OF LITERATURE

The investigation for each subsystem provided important information for designing each subsystem and the requirements to allow for correct operation and integration. The investigation also provided the information that helped decide between design alternatives.

The smart card literature is very detailed, though some aspects of the literature was not relevant to the design and implementation of this system. Many aspects of the literature were written for designing and creating a smart card that meets ISO 7816 systems, such as the physical dimensions, registration of applications, and using cards in multi-application environments. The information that is important is how smart cards operate and how to interface with them, specifically the electrical and communication requirements. The type of smart card is important when determining the electrical requirements for interfacing since they do have vastly different requirements. The communication protocol of smart cards is important when supporting a wide variety of smart cards, specifically the ATR protocol. The ATR protocol is aimed at optimising communication between the smart card and terminal. The electrical characteristics are important for the design of the interface and understanding the requirements to allow for successful connection to the smart card.

The smart card terminal is the middle component to which the smart card and smartphone needed to interface. The main aspects that are relevant for the terminal are the that it met the requirements of the other subsystems and can interface through its required connections, such as universal asynchronous receiver-transmitter (UART), General-purpose input/output (GPIO) pins, and pulse width modulation (PWM). The subsystems that needed to be integrated into the

terminal are a human machine interface, a wireless interface, and the smart card interface. Wireless interface components typically use UART based communication, where the human machine interface systems do support a variety of different interfacing methods but typically use GPIO. The smart card interface requires a few components to operate such as a voltage supply, clock signal, and data signal.

The encryption algorithm was to be implemented on the smart card to encrypt the private information that will be stored on the smart card. Smart cards have very limited processing and memory resources. The encryption on smart cards such as bank cards are done with hardware encryption engines to allow for faster transactions and removing the resource requirements from the microprocessor. Investigating each algorithm was important in determining which algorithm would perform better in a resource limited environment specifically a software environment. The encryption algorithm keys were to use the entered PIN for the card. The literature for each algorithm is very detailed on how the operation of the algorithm is achieved.

The smartphone was an important aspect to the system for determining the best option for interfacing with external hardware with regards to hardware and software. The smartphone mainly needs to meet the wireless interface requirements and that it can develop and run a payment application. The software development tools chosen were dependent on the chosen smartphone platform.

2. Approach

2.1 OVERVIEW

The overall system approach was limited by design. The system was designed as different subsystems that needed to interface to make up the final system. Though each subsystem has different approaches in terms of design and implementation. Each subsystem was required to interface with some of the other subsystems, this design requirement was kept in mind throughout the design process. The original system approach is detailed below in Figure 4.

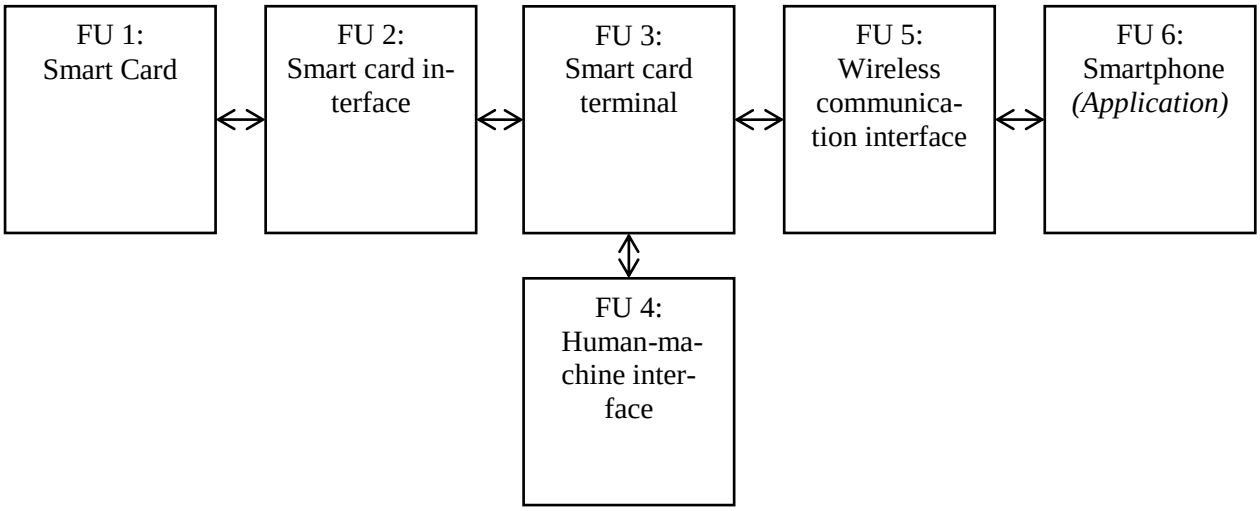

Figure 4
System functional diagram

2.2 SUBSYSTEMS

The initial approach for each functional unit of the final system is detailed in its own section with a discussion on the selected approach and its alternatives. The order of the headings details the timeline in which each subsystem was implemented.

2.2.1 FUNCTIONAL UNIT 3: SMART CARD TERMINAL

The smart card terminal was the subsystem that served as the middle-man between all the other devices so the decision on implementing a working terminal first helped ease development for all the other subsystems.

The terminal processing could be implemented with a variety of different processing platform options. The system could've used a processor chip such as an ARM or a FPGA. Though these options increase development time since it acts as only a processor and all other required system peripherals would need to have been implemented separately. The ARM processor is the ideal approach when development time is long, and all the required features can be developed to required specifications, providing a cheaper terminal as a final product which could be integrated into a final board for mass production. The use of an ARM also increases software complexity and a product needed to be delivered in a limited amount of time. A FPGA choice was not ideal since these chips are built for parallel processing tasks which did not suit this subsystem.

A more feature providing system was the better alternative. A microcontroller or an FPGA development board provided most of these system requirements whilst saving on development costs and time.

The subsystem could also have been implemented on a single board computer such as an ODROID which provided the processing power and the system peripherals. The single board computers provide a very easy development platform since they offer more powerful tools to ease development. These single board computers are built to be usable in many scenarios and comes with a wide variety of peripherals and software features. These types of systems are costlier and require an operating system to function, which would've been ideal if the terminal needed to run multiple different applications, but it did not. This subsystem only needs to perform a specific task, so having a very general-purpose device suited the system design.

The microcontroller appeared as the best option, providing enough pre-built features, easing the development, and providing enough computational power, whilst serving as a more cost-effective device in comparison to the single board computers.

The original software plan for the smart card terminal is broken down in Figure 5. The smart card terminal serves as the master of the entire payment process. Functional unit 3.2 would be this master code. It was to serve as the interface and communication protocol between all the other subsystems as shown in functional units 3.1, 3.3, and 3.4.

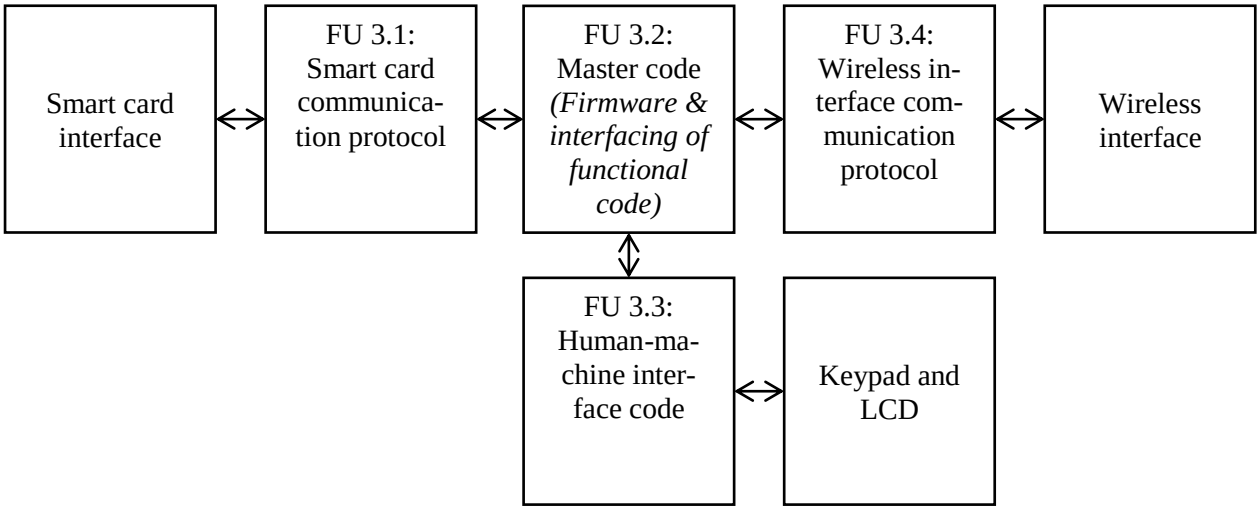

Figure 5
Breakdown of functional unit 3: Smart card terminal software

2.2.2 FUNCTIONAL UNIT 4: HUMAN-MACHINE INTERFACE

This system was based on a financial based system therefore having an extra layer of security to ensure that the card is used by the original owner presented a way of reducing fraud in cases that the card is lost or stolen. A variety of advanced identification systems such as fingerprint reading, iris detection, or speech recognition could have been used, these systems would have been costlier financially and implementation wise. A user defined PIN is used in current financial systems since it provides a decent layer of security that is up to the owner of the card to keep their PIN secure. The PIN system appeared as a more cost-effective solution requiring only a keypad instead of using biometric scanners and algorithms for identification.

The terminal needed to relay instructions and information to the user. A LCD was an option providing information to users providing easy to understand instructions and details about the systems current state. LCD's implementations are more complex in comparison to some other options such as LED's displaying system states. The benefits of more detailed instructions and information made the LCD the choice. LCD's do come in a wide variety of options and sizes at a wide range of cost. The information that needed to be displayed was not in huge quantities, so a small simple LCD provided the benefits at a reasonable cost.

2.2.3 FUNCTIONAL UNIT 5: WIRELESS COMMUNICATION INTERFACE

There are a variety of wireless systems that could have been used that meet the data rates and operating distance for the system. Bluetooth appeared as the best option since many smartphones have this system built and uses its own encryption system. Wi-Fi is also a cryptographically secure option (when setup correctly) built into many smartphones but requires an extra system, a Wi-Fi hotspot, to be present to allow for device interfacing which was not ideal. Wi-Fi direct was an option much like Bluetooth but is not present in many smartphones so did not appear to be a good option.

2.2.4 FUNCTIONAL UNIT 6: SMARTPHONE APPLICATION

Since Bluetooth was the chosen wireless communication interface it was important to get a smartphone that supported it. The application was to be originally built using a cross platform library to allow support for multiple devices across Android, iOS, and Windows devices. A software functional breakdown of the application was to be made once a thorough understanding on how phone applications and specifically Bluetooth on phone applications operate.

2.2.5 FUNCTIONAL UNIT 1: SMART CARD

The system requirement was to use a contact smart card. A Contactless smart card could have been used, serving the same function in the system, but these cards can be connected to from a distance without the owner's knowledge which is not ideal for a financial system. Some newer bank cards do make use of these contactless systems, though it has a lot of restrictions and regulations.

Encryption on financial smart cards is done via hardware due to limitations of the smart card microprocessor. The encryption for this system was to be a software implementation, therefore the encryption algorithm was to be developed first with a Python simulation. This was the approach to ease development in a simpler language with better debugging capabilities whilst helping build a better understanding of the encryption algorithm. The Python simulation was to provide information on processing and memory requirements of the algorithm.

An 8-bit smart card is one of the requirement of the system, limiting variable sizes to 8-bits in length. This limitation made RSA a difficult choice since calculations on variables above 8-bits is required. 3DES was chosen over RSA since it is a bitwise operating algorithm and would function better on an 8-bit microprocessor.

2.2.6 FUNCTIONAL UNIT 2: SMART CARD INTERFACE

The smart card interface was planned to be the final subsystem to be implemented, where an off the shelf interface was to be used during initial development to help ease the development

process and allow the other subsystems to be built into a working state. Once the other subsystems had reached a working state this was to be designed from first principles and built.

2.3 DESIGN TOOLS

The design tools used throughout the system design and implementation are detailed below.

Atmel Studio was used for the development of the microcontroller software. It provided the tools and software to develop, debug and implement the software on the microcontroller.

Android Studio was used for the development of the smartphone application.

JetBrains PyCharm was used as the Python development platform that was used to simulate the encryption system.

Autodesk Eagle was used for designing the circuits of the system. It also provides printed circuit board creation tools but was not used.

2.4 OFF-THE-SHELF HARDWARE

Atmel ATMega1284P was used as the smart card terminal microcontroller. This is an 8-bit microcontroller. An 8-bit controller was chosen because the smart card would be 8-bit and there was no benefit in using a more advanced controller for this subsystem since it didn't need to perform any intensive calculations. This controller also provided the 2 required UART modules for it to interface with other devices and enough GPIO pins for the human-machine interface devices and some other features. It also has a PWM system to help produce a clock to interface with the smart card device. The device was programmed using an AVR JTAGICE MKII programming device.

RN-42 Bluetooth 2.0 Module with a SPF breakout-board was used for the wireless interface. The Bluetooth module works at a supply and communication level of 3.3 V, so the breakout board serves as a voltage regulator and logic-level shifting circuit. This device interfaced using UART with a microcontroller.

A Displaytech 16x2 character LCD is used for the smart card terminal display. The smart card terminal made use of a EOZ 16-key keypad for PIN entry and other commands.

The smart card was built from first principle due to design decisions later, so it also made use of the Atmel ATMega1284P microcontroller. Smart card microcontrollers make use of serial peripheral interface (SPI) programming, but the available programming devices didn't have working SPI interfaces so the same microcontroller as the terminal was used due to its availability. The device supported two UART modules which allowed for debugging through UART communication during development. It also has a large flash memory capability which supported the encryption algorithms requirements. It also has a built in EEPROM of 4 kB to store data on.

A Samsung S5 was used for the phone application. This phone features are 2.5 GHz quad core processor, 2 GB of RAM, internal storage of 32 GB, and has Bluetooth. This phone was chosen for its Bluetooth capabilities and was readily available.

3. Design and implementation

3.1 SMART CARD TERMINAL HARDWARE AND PERIPHERALS

The smart card terminal was the first system designed and implemented. The first task was to build the circuits and interfaces to allow for the microcontroller to operate, debug, and be programmed. Once those systems had been completed the next focus was to implement working human-machine interface to ease development for testing and debugging of the system, by allowing for visual feedback through the LCD and device control via buttons on the keypad. The next step was to get the wireless interface working with the terminal, so development of the smartphone application could begin. Once a working application and smart card interface was complete the final terminal firmware was completed.

3.1.1 MICROCONTROLLER OPERATING HARDWARE

The microcontroller that was used for running the terminal required some basic things to allow for proper operation and debugging. These small systems are detailed in brief below.

The microcontroller is a 5 V powered device. The system needed to be battery powered, so a 5 V voltage regulator was used as its power source. Two capacitors were attached to smooth the signal. This allowed two rechargeable 9 V battery to power the terminal. This system is shown below Figure 6.

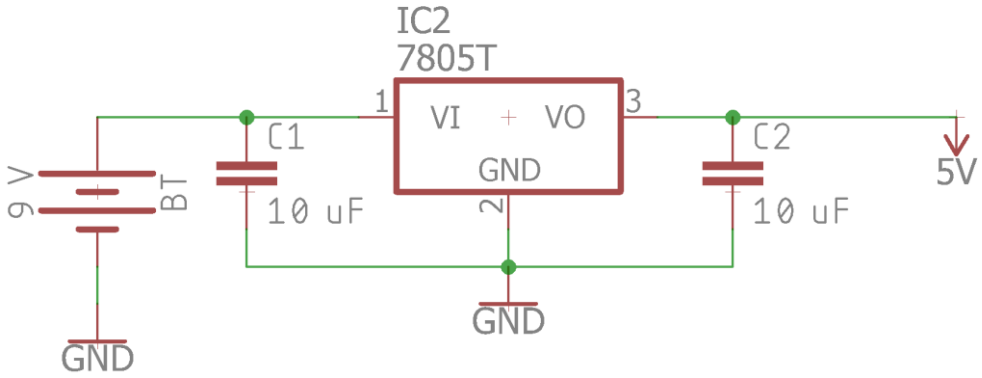

Figure 6
Voltage regulator circuit for smart card terminal

The ATMega1284P has an internal oscillator of 8 MHz which met the processing requirements of the terminal. In later development an external 16 MHz oscillator was attached and the internal one disabled as the internal oscillator was unstable and produced many errors when performing UART communication. This external oscillator required capacitors ranging from 18 pF – 22 pF to operate correctly. This circuit is shown in Figure 7.

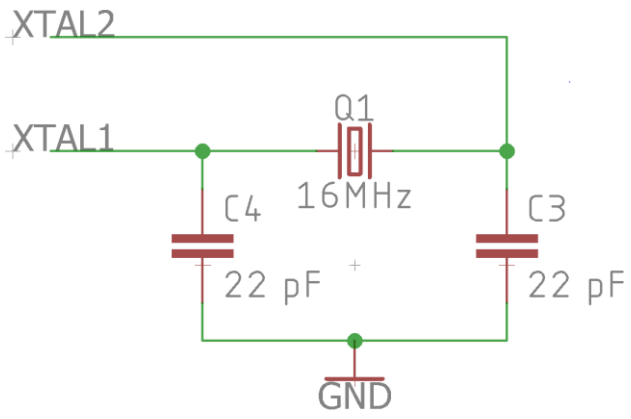

Figure 7
External oscillator circuit

The device uses a Joint Test Action Group (JTAG) programming interface. An Atmel AVR JTAGICE MKII was used for programming and debugging. JTAG standard interface is a 5-pin connection. The programmer also required some other connections such as power and the reset signal to the microcontroller to be able to program and debug. A 10-pin connection header is used for the programmer.

The 10-pin header is detailed in Figure 8. The test reset pint (TRST) is used when daisy chaining devices. The reset pin of the microcontroller uses a 10 kΩ pull up resistor to operate correctly, since the reset line is pulled low to reset the device.

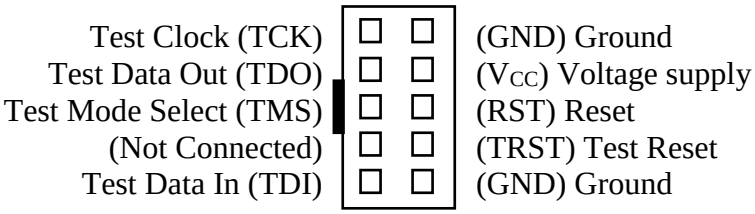

Figure 8
JTAG interface pinout

3.1.2 LIQUID CRYSTAL DISPLAY HARDWARE AND SOFTWARE

The LCD is a device that interfaces with a microcontroller with mainly GPIO pins and uses its own communication protocol to display data and perform instructions. This is broken down into a description on the hardware and how it was connected and the software library that was developed from first principle for the device to operate.

3.1.2.a Hardware

The terminal made use of a 16 x 2 LCD, supporting 32 characters. This LCD operated by reading data from data pins and reading control signals to determine what to do with the data. The LCD used the 5 V supply that the microcontroller used. The contrast of the LCD was adjusted by attaching a potentiometer from the voltage supply and connecting it to contrast pin on the

LCD. The LCD had a backlight but was not used to save battery power. Further description of the pins of the LCD is shown in Table 1

Pin NO.	Symbol	Name	Description
1	A	Backlight anode	This is to provide a positive voltage to operate the backlight of the LCD.
2	K	Backlight cathode	This is to provide a negative voltage to operate the backlight of the LCD.
3	V_{SS}	Ground	This is to provide a ground reference, so the device can operate correctly.
4	V_{DD}	5 V logic supply	This voltage is to power the digital circuitry and display characters on the LCD.
5	V_O	Contrast voltage	This voltage signal is to adjust the contrast of the LCD (3.3 – 3.7 V).
6	RS	Data and instruction signal	When signal is high (5 V logic) the LCD is receiving data to display. When signal is low (0 V logic) the system is receiving instructions.
7	R/W	Read and write signal	When signal is high (5 V logic) the LCD is in read mode. When signal is low (0 V logic) the LCD is in write mode.
8	E	Enable signal	Enable signal used to inform LCD that a command has been sent.
9	DB_0	Data bit 0	Data signals used for data and instructions. 8-bit interfacing mode makes use of all data bits. 4-bit interfacing mode uses DB_4 – DB_7.
10	DB_1	Data bit 1	
11	DB_2	Data bit 2	
12	DB_3	Data bit 3	
13	DB_4	Data bit 4	
14	DB_5	Data bit 5	
15	DB_6	Data bit 6	
16	DB_7	Data bit 7	

Table 1
LCD pins and description

The LCD operates using a simple process. The values RS, R/W and DB are pulled high or low depending on the desired outcome. When the enable signal is pulled high for at least 500 ns and then pulled low. When this occurs, the LCD takes the input information and produces the result depending on the values of RS, R/W and DB. These commands can range from writing a specific character to the screen, clear the entire screen, setting the cursor position, setting the cursors visibility, and many more. To determine when the LCD is ready to receive data it has a busy flag which can be read from the LCD's memory with a command.

3.1.2.b Software

A software library was written to allow quick and easy operation of the LCD on the terminal. Some of these functions were not used in the final implementation of the system but were added during development if the need for them arouse. A list of all methods and a short description of them is shown in Table 2

Method	Description
+ init_LCD(): void	This method performs all the microcontroller port initialisation and sets the interface mode and display modes for the LCD.
+ lcd_Busy(): bool	This method checks if the LCD is busy or not, checking whether it is ready to receive an instruction or data.
+ lcd_Command(char**): void**	This method sends a command to the LCD defined by the input variable it receives and setting all the ports for a LCD command.
+ lcd_Write(char**): void**	This method writes the received data to the LCD display.
+ lcd_String(char***): void**	Writes a received string to the LCD display.
+ lcd _Clear(): void	Clears the LCD screen and returns cursor to beginning.
+ lcd _Return(): void **+ lcd _Return_l2(): void**	Returns the LCD cursor to beginning on line one and line two respectively.
+ lcd _DisplayOn_CursorOff(): void **+ lcd _DisplayOn_CursorOn(): void** **+ lcd _DisplayOn_CursorBlinking(): void** **+ lcd _DisplayOff(): void**	These methods are used to turn on or off the LCD display and set the cursor settings.
+ lcd _ShiftLeft(): void **+ lcd _ShiftRight(): void**	Shifts the cursor left or right on the display.
+ lcd _SetFunction_8bit_1l_5x7(): void **+ lcd _SetFunction_8bit_2l_5x7(): void** **+ lcd _SetFunction_4bit_1l_5x7(): void** **+ lcd _SetFunction_4bit_2l_5x7(): void** **+ lcd _SetFunction_8bit_1l_5x10(): void** **+ lcd _SetFunction_8bit_2l_5x10(): void** **+ lcd _SetFunction_4bit_1l_5x10(): void** **+ lcd _SetFunction_4bit_2l_5x10(): void**	These functions determine three important settings on the LCD. The data interfacing mode (8 or 4 bit), the amount of lines used on the display (1 or 2) and the number of pixels used for each character (5x7 or 5x10 pixels).
+ lcd _Startup(): void	This is used to display a LCD initialisation and start up message on LCD.

Table 2
LCD library description

3.1.3 KEYPAD HARDWARE AND SOFTWARE

The keypad is broken into two sections, hardware and software. The hardware section which describes how the keypad operates and how it interfaces with the smart card terminal microcontroller. The software component describes the library and the state machine that allowed for correct data to be read from the keypad.

3.1.3.a Hardware

A 16-key keypad is used for operating the smart card terminal. The keypad works by using 8-in system. The 8 pins are setup with 4 pins connected to the rows and the other 4 pins to the columns of the keypad. This created a 4 x 4 connection matrix, with each key acting as a switch.

When a key is pressed it bridged the connection between a row and a column. This is information was used to detect which key had been pressed. A circuit diagram of this is shown in Figure 9 below.

The key detection was achieved over two steps. First the rows were set to outputs and to digital low, grounding anything connected to it. The columns were then set to inputs with a pull up resistor activated on the inside. When a button was pressed the column of that button will be pulled low as it is pulled down by one of the rows outputs. The column inputs were then read, finding the input with a value of 0, this helped determine which column in which the button was pressed. This was only half the information needed to determine the pressed button. The second step was the same as the first with the outputs and inputs are switched so that the rows are the inputs and the columns are the outputs. This helped determine the row of the key. With both the column and row information the microcontroller could determine the button that was pressed. This method only worked when only one button is pressed, when multiple buttons are pressed it cannot correctly detect the pressed buttons.

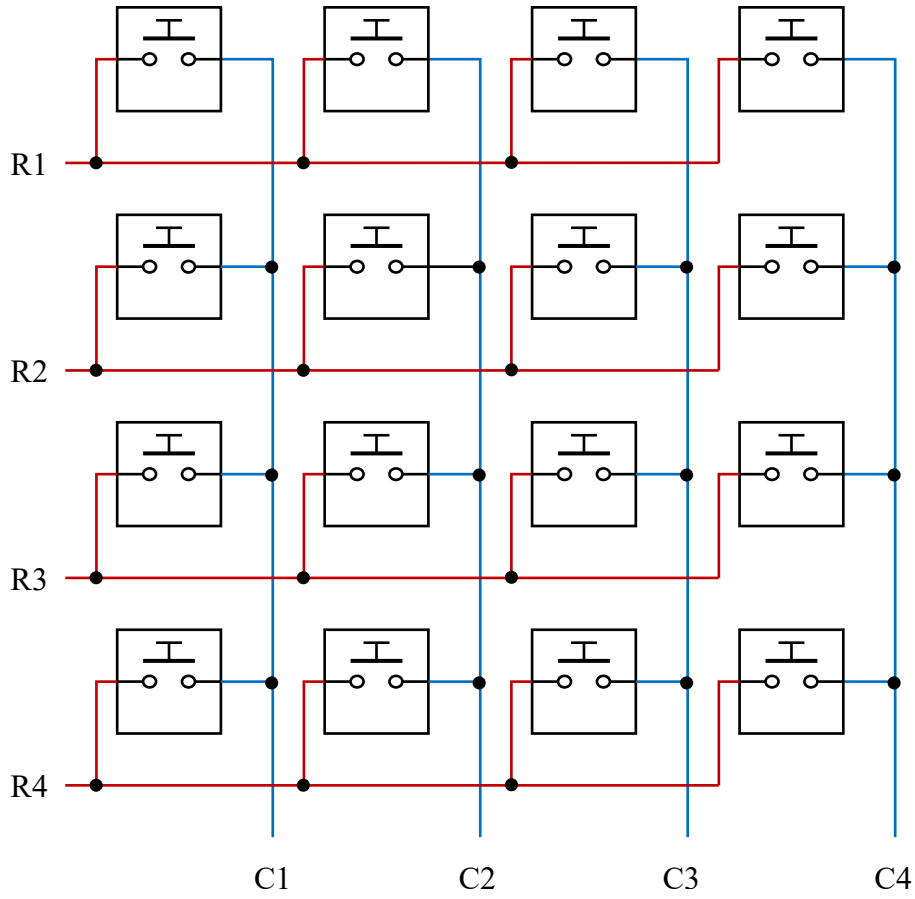

Figure 9
Keypad circuit diagram

A table detailing the keys and data combinations is shown in Table 3 below.

Col / Row	C_1 **1110**	C_2 **1101**	C_3 **1011**	C_4 **0111**
R_1 **1110**	**1** 1110 1110	**2** 1110 1101	**3** 1110 1011	**F** 1110 0111
R_2 **1101**	**4** 1101 1110	**5** 1101 1101	**6** 1101 1011	**E** 1101 0111
R_3 **1011**	**7** 1011 1110	**8** 1011 1101	**9** 1011 1011	**D** 1011 0111
R_4 **0111**	**A** 0111 1110	**0** 0111 1101	**B** 0111 1011	**C** 0111 0111

Table 3
Keypad keys and their data combinations

3.1.3.b Software

The keypad can operate with just the hardware component and simple software to control the ports for setting the input/outputs of the pins and reading their values. The problem is that the device will constantly read the button being pressed on the keypad as multiple presses and may pick up erroneous data when buttons are pressed or released. To fix these problems a software debouncing system was made to prevent erroneous behaviour when the keys are pressed and released and to only read the key value once it has been released. A debouncing circuit could be used here instead of software, but requires a circuit for each row and column. Software can achieve the debouncing without any extra circuitry and can provide the ability to only read the key on release in the same algorithm.

The software debouncing algorithm works around a comparator interrupt in the microcontroller. Every time this interrupt condition was met the state machine would check if the state had changed and adjust it accordingly. The state machine started first in a state where the button was not pushed, it would remain in this state whilst a key value had not been detected. Once a key was detected it will transition to a detected state, then the comparator interrupt would run. This comparator acted like a small timer circuit inside the microcontroller (~50 ms) only up-dating the state machine after a certain amount of time had passed.

This timing system prevented erroneous behaviour being picked up. Once the system was in the detected state it would check if the button was still being pressed (correct detection) or not being detected (incorrect detection). If the detection was an incorrect one, it transitioned back to a not pushed state. If it was a correct detection, the system transitioned to a pushed state and the key value was read and stored. The final transition to released is similar to the first, detecting whether the button release was correct or an incorrect reading. Once a correct reading was de-tected in the released state, the key pressed was returned to the main program and the state machine returned to a not pushed state. This process is detailed below in Figure 10.

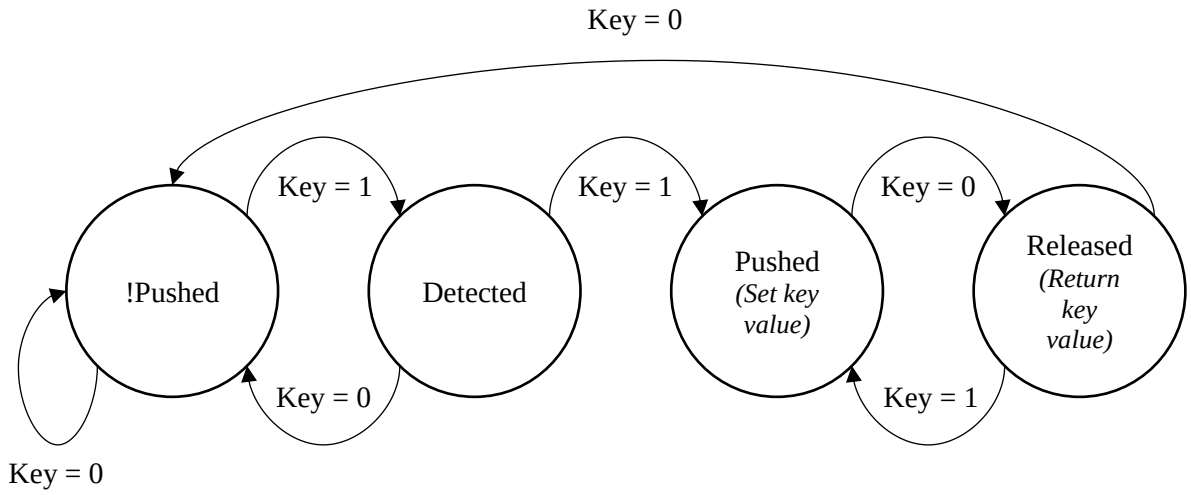

Figure 10
Debouncing state diagram

The keypad software components were written by first principle into a simple library file to allow for simple use. The debouncing state machine was not included in this library due to its need to use a comparator interrupt to operate correctly.

Method/Function	Description
+ init_Keypad(): void	Initialises microcontroller ports to allow for interfacing with keypad.
+ keypad_GetButton():uint8_t	Checks and returns the keypad button number being pressed. Returns 0 otherwise.
+ keypad_NumToChar(uint8_t**): char**	Converts the keypad button number received to a char.
+ keypad_IsNum(char**): bool**	Checks if the pressed keypad button is a number button. Used for pin entry purposes.

Table 4
Keypad library description

3.1.4 WIRELESS INTERFACE

The wireless interface used a RN-42 Bluetooth module, this is a Bluetooth V2.0 class 2 module. This device can transmit data at up to 115200 bps at a range of up to 10 metres. This module connects to a microcontroller using UART. This interface is a type of serial communication where each device has a receive (RX) and transmit (TX) connection to transmit binary data. The communication lines are connected by connecting RX of the device to the TX of the other and vice versa. Once the hardware is connected some communication settings need to be set on both devices to facilitate communication. These settings are:

- Number of data bits,
- Parity bits,
- Synchronisation bits (also known as stop bits),
- Baud rate, and
- Flow control settings.

When both devices have the correct settings and their data lines are set correctly, communication can begin.

The module has many pins, but only the important ones have been given since all the required functionality could be achieved with these pins. These pins are detailed in Table 5. Pins 1 and 6 was not used since no flow control was needed in the system.

Pin NO.	Name	Description
1	CTS	Clear to send. This pin is used if serial flow control is desired. Optional.
2	V_{CC}	Voltage supply. This can be in the range of 3.3 V – 6 V.
3	GND	Ground. This is the ground reference that should be shared with all other devices.
4	TX	Transmit. This is the modules transmit pin and should be connected to the RX pin of the interfacing device.
5	RX	Receive. This is the modules receive pin and should be connected to the TX pin of the interfacing device.
6	RTS	Request to send. This pin is used if hardware flow control is desired. Optional.

Table 5
Bluetooth module pins

The communication between the microcontroller and Bluetooth module was achieved using the following settings:

Data bits	Parity bits	Synchronisation bits	Baud rate	Flow control
8-bits	0-bits	1-bit	38400 bps	None

Table 6
Serial communication settings

The Bluetooth module also supports a variety of commands to change the settings of the Bluetooth device. These commands are used by first sending the string "$$$" to the module which puts it in command mode. This is where settings like the devices baud rate, name, pairing pin, and many other things can be changed. This mode is quit by sending the string "---".

The system made use of interrupt driven reading for serial communication. This was used over polling to allow for the transaction to be cancelled at any point by the smartphone application. The interrupt routine for receiving data was setup to read data that it receives until a return carriage or new line character was detected. Once one of these characters were detected a message flag would be enabled and the data would be processed by the main function. This method has a problem that if data is sent too quickly, where data is received before the previous message was processed by the main function, that the message would overlap the other or be processed with half data since the message flag is still set. A circular buffer for received data would fix this but the system worked in a message-reply methodology, so this wasn't necessary. The code snippet of the interrupt routine is shown in Figure 11.

```
ISR (UART0_RX_vect)
{
	// Read value from receive buffer
	charRead = UDR0;
	// Check if return carriage or newline
	if(charRead == '\r' || charRead == '\n')
	{
		// Set message flag
		messageRX_flag = 1;
		// Reset length counter
		UART0_rx_length = 0;
	}
	else
	{
		// Add character to message buffer
		UART0_rx_buffer[UART0_rx_length] = charRead;
		// Increase buffer length
		UART0_rx_length++;
	}
}
```

Figure 11
Code snippet of interrupt routine for receiving serial data

3.1.5 SMART CARD MICROCONTROLLER HARDWARE

A breakdown of all the pins used in the microcontroller is shown in Table 7. The firmware section of the terminal was developed fully once all the other subsystems were working.

PIN	I/O	Description
PA0 – PA7	I/O	These ports are used for the keypad. They are set as inputs and outputs for detecting the keypad presses.
PB0 – PB7	I/O	These are used for the data ports for the LCD. Used as outputs for writing data and inputs for reading data from the LCD (Busy Flag).
PD4, PD6, PD7	O	These are the control signals for the LCD. RS, R/W, and enable.
XTAL2, XTAL1	I	Used to connect the external oscillator.
PD0	I	RX pin for the Bluetooth UART communication.
PD1	O	TX pin for the Bluetooth UART communication.
PD2	I	RX pin for smart card interface UART communication.
PD3	O	TX pin for smart card interface UART communication.
PD5	O	Output pin for the PWM signal used for the smart card interface.
PC2 – PC5	I	Pins for JTAG debugging and programming.
PC6	O	Smart card interface reset signal. This output signal is pulled low to reset the smart card device.
PC7	I	Smart card interface switch for detecting when the smart card is connected to the terminal.

Table 7
ATMega1284P pin breakdown and description.

3.1.6 SMART CARD TERMINAL CIRCUIT

The final circuit diagram with its power supply, external oscillator, programming interface, keypad, and LCD was built onto one circuit.

A picture of the final circuit is shown below in Figure 12 and a picture of the terminal enclosure with the attached LCD and keypad in Figure 13.The circuit diagram for the final terminal is shown in Figure 14.

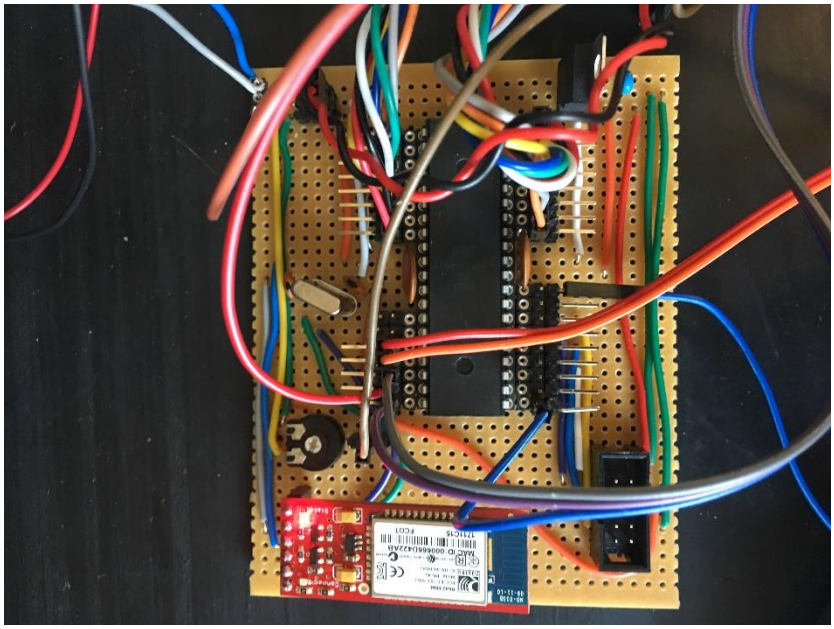

Figure 12
Smart card terminal photograph

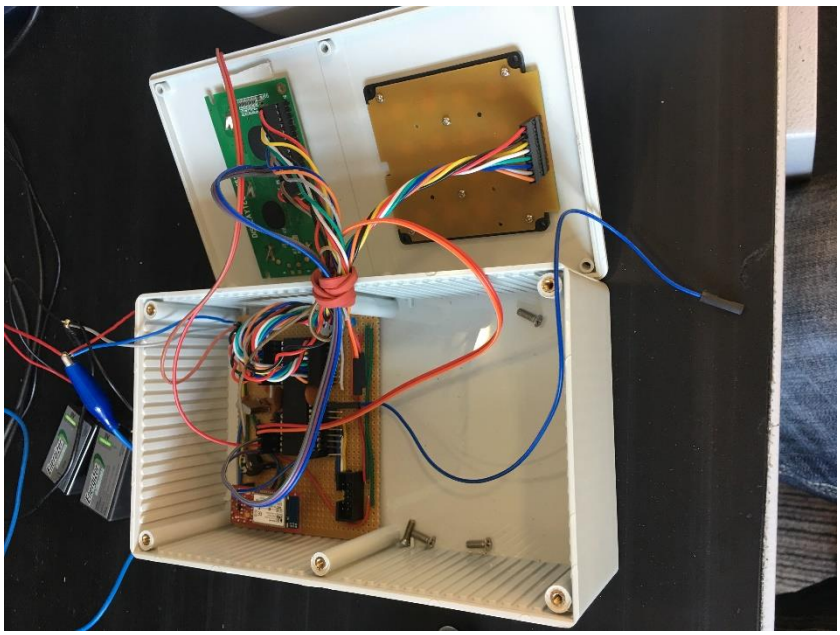

Figure 13
Photo of smart card terminal in enclosure

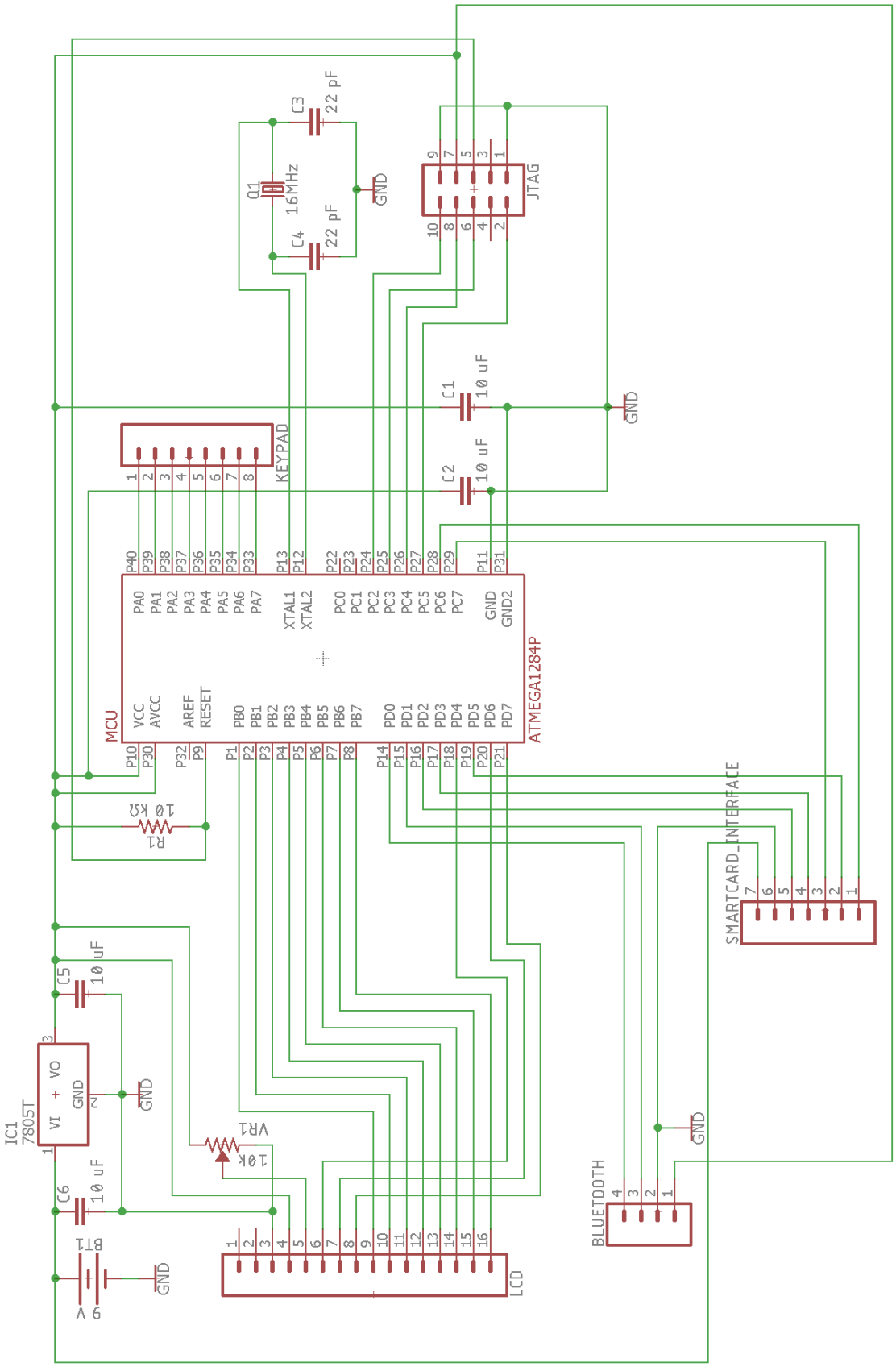

Figure 14
Smart card terminal circuit diagram

3.2 SMARTPHONE APPLICATION

The initial approach for the smartphone application was to write the application with a cross-platform system and library. This idea changed once more research on Bluetooth systems operate on the smartphone devices. Using a cross-platform library would require each operating systems Bluetooth system to be written separately and another Bluetooth service to be written in the cross-platform library. An iOS compiling device and Windows phone was not available during development so writing extra code for these systems would be time consuming and couldn't be tested.

The approached used in the final system was purely for Android and was developed using Android Studio 2.3.3. The design for the smartphone application was to achieve the following requirements:

- Connect with the smart card terminal via Bluetooth,
- Use a defined communication protocol to able to communicate the following things:
    - Payment request with both the transaction name and cost.
    - Whether the payment request was received or declined.
    - A transaction being cancelled from both the smartphone and terminal.
- Show progress of the transaction, and
- Keep a record of payments that have been received.

3.2.1 ANDROID DEVELOPMENT

Android applications in Android Studio are written in Java using a variety of Java and Android libraries. Android uses a system for applications called activities. These activities act as classes within the app and work in a parent-child method. They typically have a graphical user interface (GUI) layout assigned to each activity. A variety of activities were built focusing on different needs of the application. These activities are discussed in terms of their functionality and GUI below.

3.2.2 MAIN ACTIVITY

The main activity is the parent for the rest of the application. It serves as the start-up screen for the application and provides options to access the features of the application.

When the main activity has been created by the application it does a few things. First it checks whether the Android device is Bluetooth compatible and if it is enabled. If the device is not compatible an error message pops up informing the user that this application can't be used on the current device. If it is compatible but not turned on, the device prompts the user to enable Bluetooth. The application will then open its preferences file and set the Bluetooth preference if there is one, else it will prompt the user to pair to an available Bluetooth device. The device will also set the layout, which is written in extensible markup language (XML), and connect listeners to all the user interface elements that have functions.

The main activity supports three major functions, request payment, payment history, and settings. The request payment and payment history functions open the respective activities which is discussed later. The settings feature provides a popup with all the paired Bluetooth devices to select which one to connect to. The graphical layout of the main activity is shown in Figure 15. It is made up of three buttons for the three major functions. An example of the settings

popup is shown in Figure 16, showing all the paired Bluetooth devices and which one is currently selected.

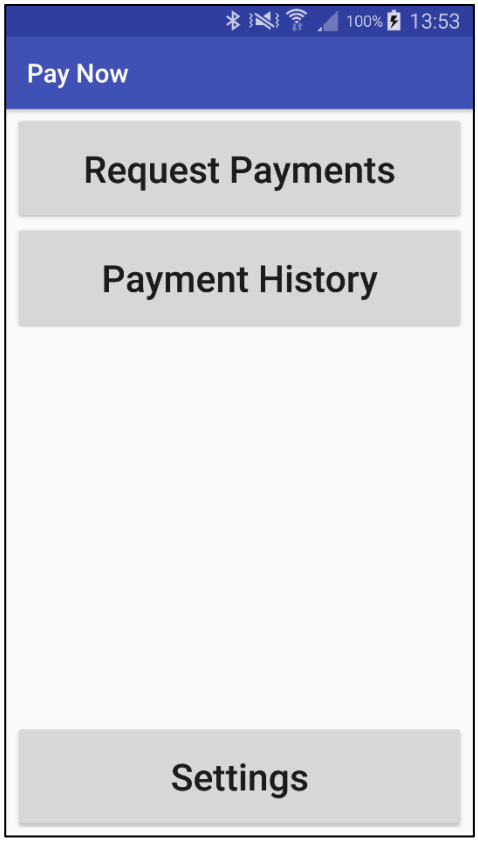

Figure 15
Main activity GUI

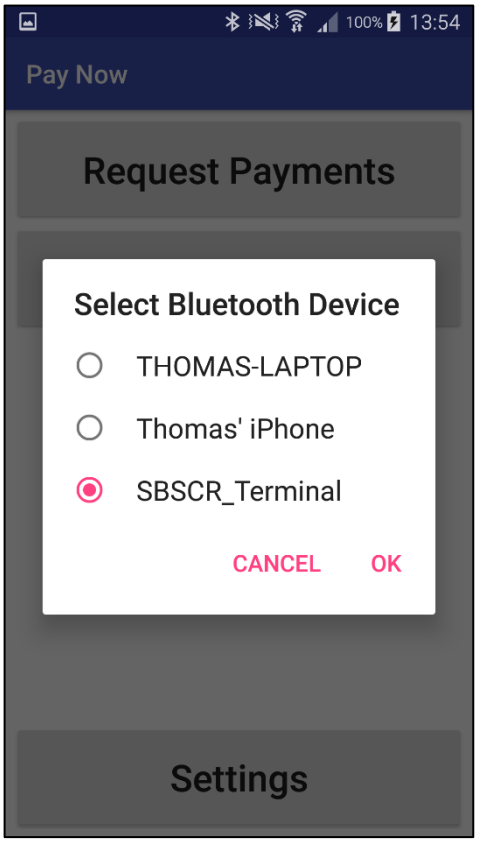

Figure 16
Main activity settings prompt

3.2.3 PAYMENT HISTORY ACTIVITY

This activity of the application is built to show a table of different payments that have been received. Each record has the following information about each received payment; payment identity number, date, time, name of product, and cost. A button to clear the payment history was added to this activity.

This activity made use of a text file that keeps the records of all the received payments that were successful. This file separated the information in the following order; identity number, name, cost, date, time using a '-' character as a delimiter and a new line for each entry. When the payment history is opened the text is read and each record is converted into a payment class. The class is used to create a custom list adapter and a special row XML layout to display this information in the table.

A delete history button was made that clears the text file and the list on the layout of all data after asking the user to confirm their selection through a prompt. Figure 17 and Figure 18 show examples the layout with some data entries and the clear history user prompt.

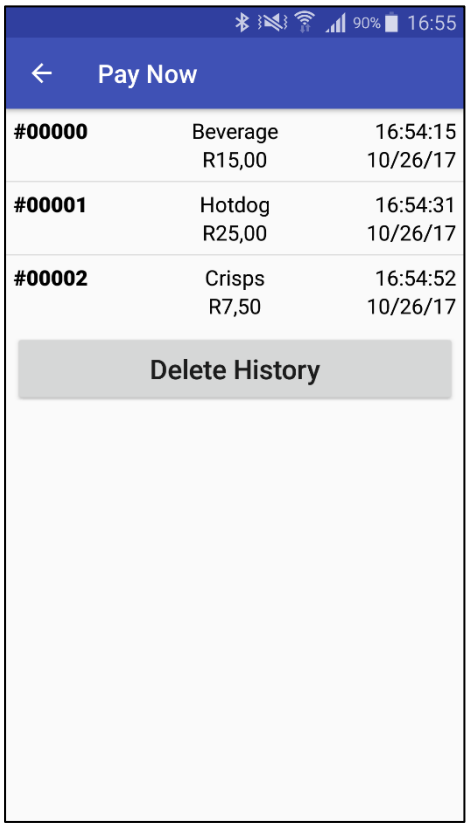

Figure 17
Payment history activity GUI

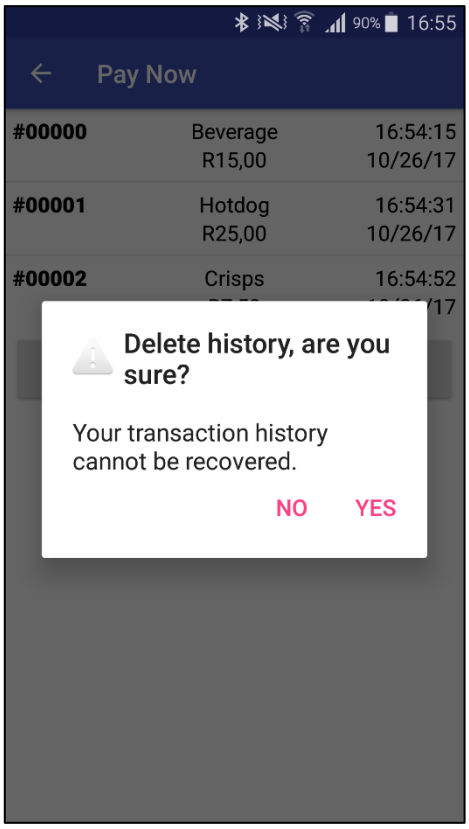

Figure 18
Payment history delete history prompt

3.2.4 REQUEST PAYMENT ACTIVITY

This activity is the focus of the application as it allows for the user to enter the transaction/product name and the cost to be sent to the terminal through two different input boxes. This takes the entered data from the input boxes and passes the to the transaction activity. This activity uses a layout as shown in Figure 19.

3.2.5 TRANSACTION ACTIVITY

This activity is a child of the request payment activity that receives the users' data for transaction/product name and the cost. This activity starts the Bluetooth service, which is discussed in greater detail later, and manages all data sent and received over the Bluetooth connection. If the Bluetooth service fails to start or a connection cannot be made, the device provides a warning and closes the activity, returning to the request payment activity. This activity uses a layout to display the information that was sent as a payment request whilst showing the transaction progress on a progress bar. It also offers a cancel button that can allow the user to cancel the payment request. Once a payment has been received this activity adds the payment to the history file and closes the Bluetooth service and returns to the request payment activity. Figure 20 shows the layout of the transaction activity and Figure 21 and Figure 22 show a successful transaction and a cancel prompt, respectively.

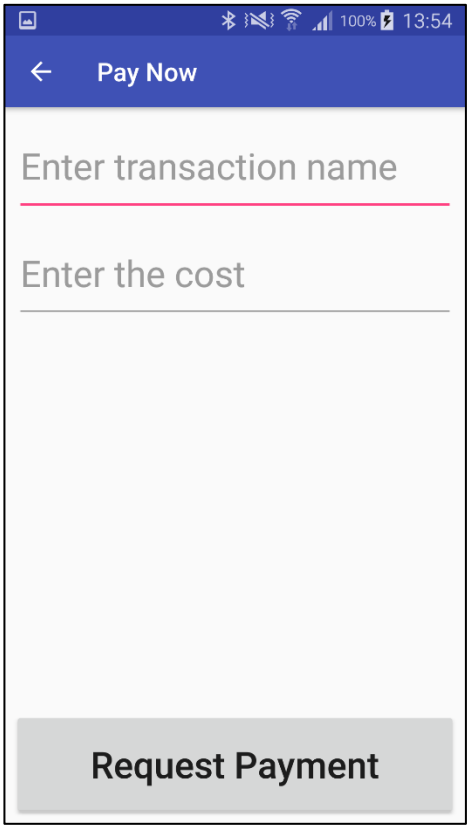

Figure 19
Payment activity GUI

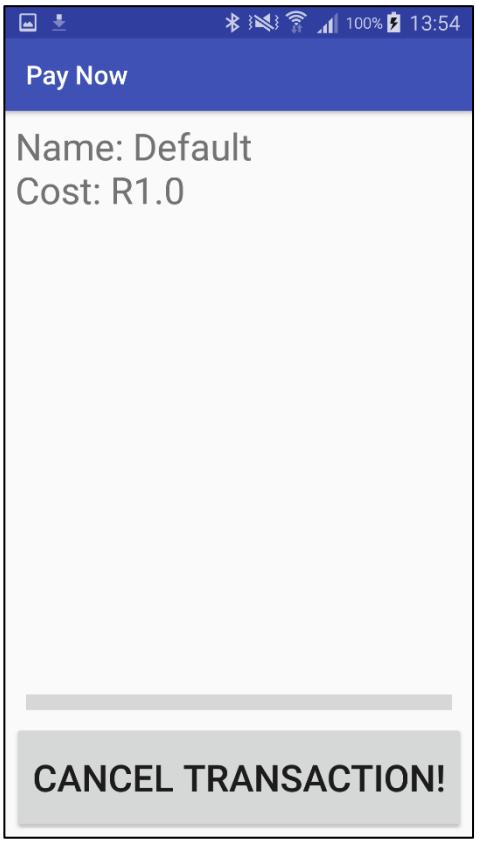

Figure 20
Transaction activity GUI

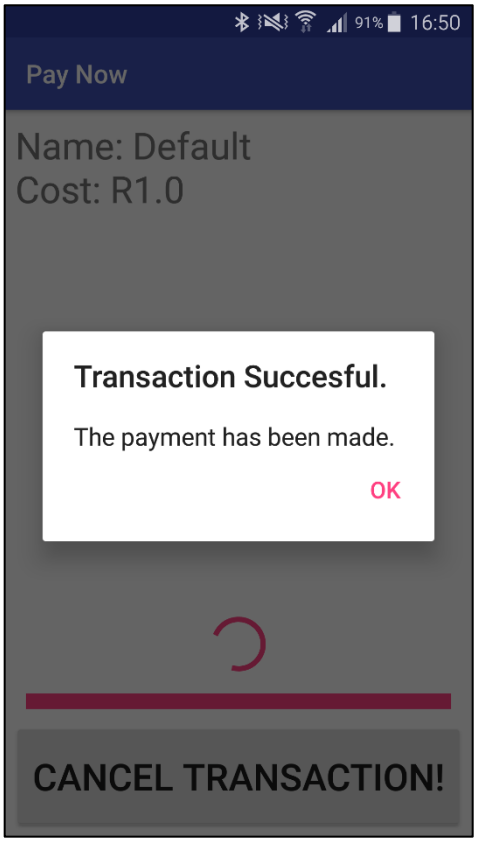

Figure 21
Transaction success message

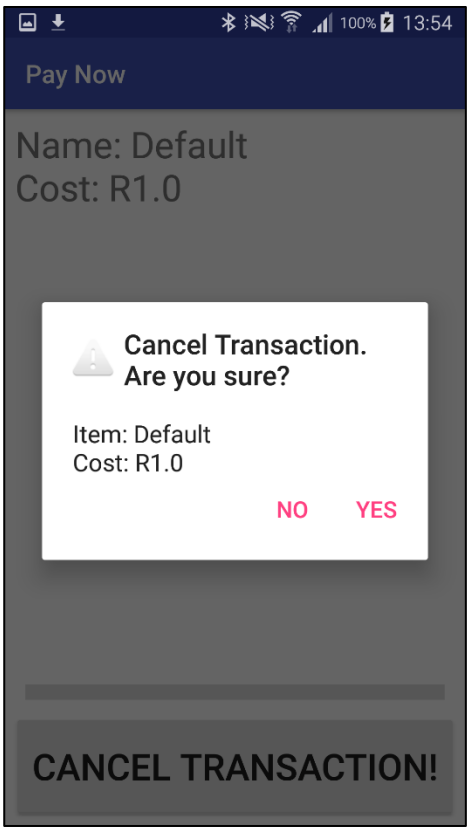

Figure 22
Transaction cancel prompt

3.2.6 BLUETOOTH SERVICE

The Bluetooth system could have been done in two ways, polling or service thread. The polling method is the simpler method, less code and lower complexity, where the transaction page would manage the Bluetooth service completely. This method is done by using the Bluetooth read and write capabilities in the code when it is needed. The problem with this method was that when waiting to receive data from the Bluetooth the application would be unresponsive. This would remove the capability for the application user to cancel the transaction and if the connection was lost the application would crash. The method that was used in the final system is the service thread system which required a lot more code and cannot be implemented within the transaction page activity.

This service allowed for the application to still run whilst Bluetooth connections are maintained in a separate thread of the system. This service method allowed for all interaction with the application to continue as normal. The Bluetooth service acts as a different processing thread and could maintain multiple different Bluetooth connections, if needed. Using a message handling service included with the Android libraries, information could be sent and received between the transaction page and the Bluetooth service. The Bluetooth service included error checking functionality to check whether the connection had been lost and reports these errors to the transaction activity.

Messages received by the Bluetooth service were immediately passed to the transaction activity which dealt with the information provided in these commands. A communication protocol was created for communication between the application and smartphone application. The protocol is shown in Table 8.

Message	Description
CONNECTED	This message is sent by the smartphone application to the terminal when a successful connection has been made. This message is sent from the terminal to the application to confirm a working connection.
PAYMENTREQUEST	This message is sent by the smartphone application to start the payment request process.
ACCEPTED	This is sent from the terminal when a payment was successful.
CANCEL	This is sent from both the terminal and application when a payment is to be cancelled.
FAILED	This is sent from the terminal when a payment has failed.
NAME:(VAR)	This is sent to the terminal when a payment request has started, with the relevant information attached as variables.
COST:(VAR)	

Table 8
Bluetooth communication protocol

3.2.7 APPLICATION UML DIAGRAM

The unified modelling language (UML) diagram is shown in Figure 23. The diagram shows all the classes that were used for the application whilst showing the most important variables and methods/functions.

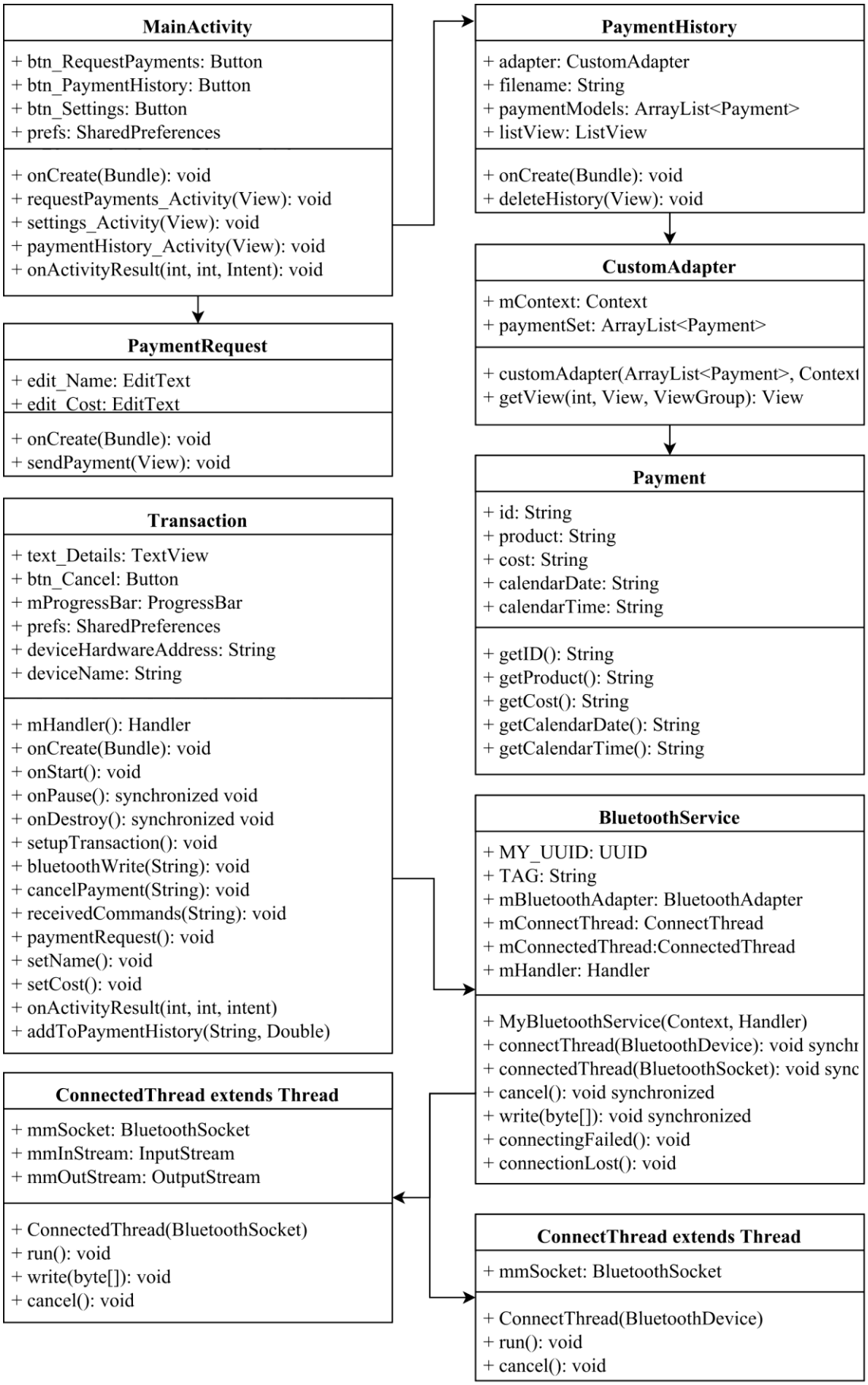

Figure 23
Smartphone application UML diagram

3.3 ENCRYPTION

3.3.1 3DES ENCRYPTION LITERATURE

The DES algorithm is made up of 3 major functions. The key schedule, which takes the key being used and generates 16 subkeys which are used throughout the algorithm. The Feistel loop function which is the structure of a variety of different block cipher algorithms including DES. The Feistel loop also requires another function known as the round function to operate fully. These steps are all part of what is known as the Feistel cipher, but DES has its own implementations of every step.

3.3.1.a Key schedule

The key schedule for DES is a function that generates sixteen subkeys that are used later in the Feistel loop and round function. It takes a 64-bit key that goes through a variety of steps to generate the sixteen subkeys.

First the 64-bit key is put through a reordering table, called the permuted choice-1 (PC-1) table. This table performs two functions. It reorders the bit array and reduces the 64-bit array into a 56-bit array. This reordering table works by moving the index of the bits throughout the array. For example, the PC-1 table moves the 1st bit (MSB) of the key to the 8th bit and the 2nd bit to the 16th bit and so on. This is done for all the bits in the key producing a newly ordered array. It reduces the key from 64-bits to 56-bits by not performing the reordering of every 8th bit in the original key and discarding it. These bits are used as parity bits for the original key and are not used in the encryption process. The substitution tables are constant meaning that the reordering of the array, index positions, will always be the same.

Once the new 56-bit reordered key is produced it is separated in half producing two 28-bit arrays. These arrays are used to generate the 16 subkeys. These arrays go through 16 rounds of operations. In each round each array goes through a circular left-shift. A circular left-shift is when the index of every bit in the array is moved left, each bit becoming more significant, and the MSB is moved into the position of the open LSB. An example of this process for an 8-bit array is shown below in Figure 24.

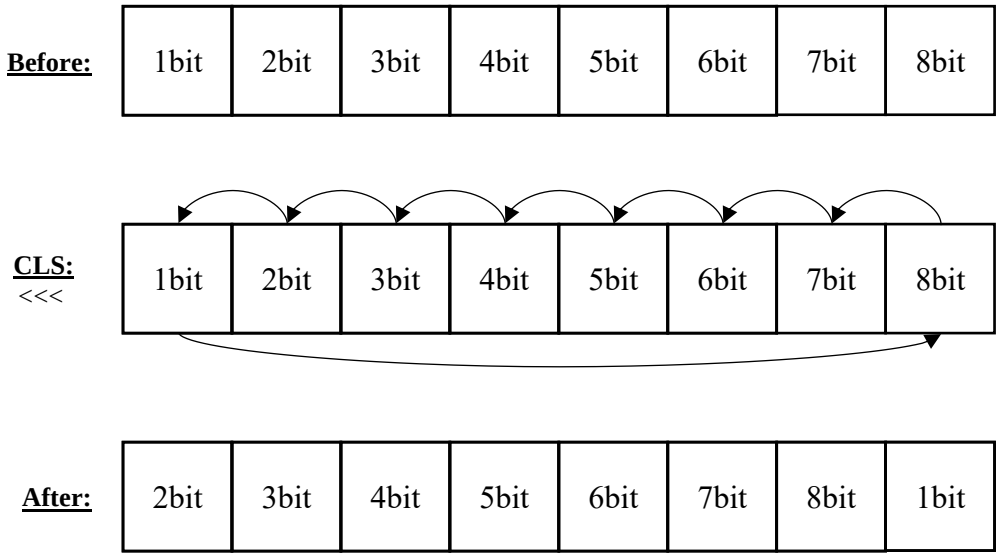

Figure 24
8-bit circular left shift

The number of circular left-shifts performed is dependent on the round number. Table 9 shows a breakdown of circular left-shifts that are performed. N being the round number and R(N) being the number of circular left-shifts.

N	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
R(N)	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

Table 9
Rotations in key schedule algorithm

After the shifting has occurred the result is put through another reordering table, permuted choice-2 (PC-2). This table operates in the same method as the PC-1 table but in a different index order and takes a 56-bit array input to produce a 48-bit array. This 48-bit array is stored as a subkey. The previously shifted arrays are then shifted again once or twice, depending on the round, with each result going through the PC-2 table to generate another subkey. This is done for 16 more rounds, generating 16 subkeys. This process is shown in Figure 25.

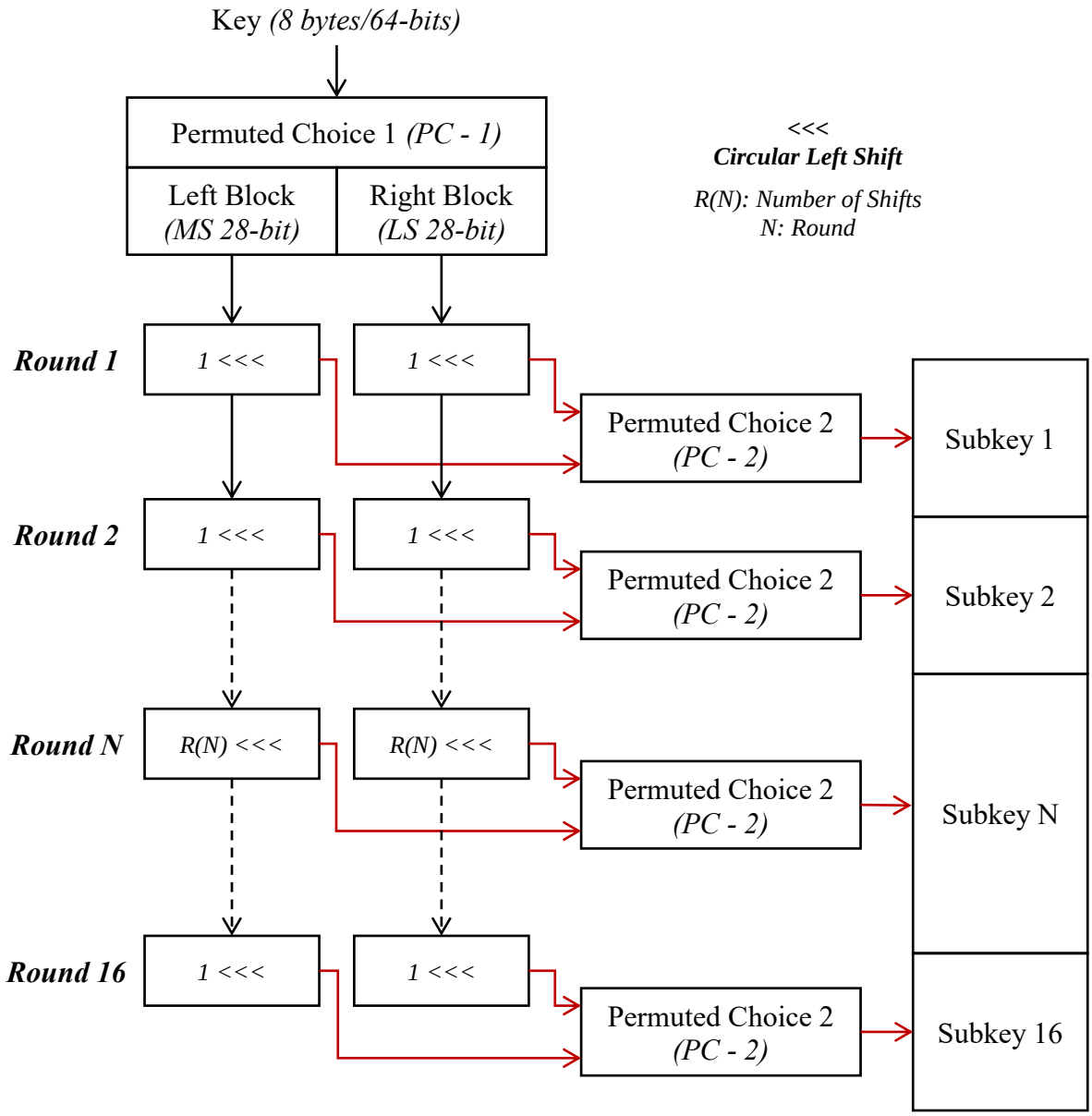

Figure 25
DES key schedule

3.3.1.b Feistel loop

The Feistel loop serves as the main component of the DES algorithm. This algorithm works by using the 16 subkeys that are generated using the key schedule and putting half of its data and a subkey through a round function for 16 rounds.

The Feistel loop starts by taking the initial plaintext message of 64-bits and puts it through a reordering table known as the initial permutation (IP) table. This table is like the reordering tables used in the key schedule, but it doesn't reduce the array size. Once the data has been reordered, it is split in half (32-bit) to form two arrays the left (L_0) and right array (R_0). These arrays are then put through the loop.

The loop occurs 16 times, using the 16 subkeys over these rounds. The first step is to use the right array in the round function, which is discussed in greater detail later in this report. The round function uses the subkey generated for the first round and the right array of data as inputs for this function. The output of the round function is then bitwise exclusive or (XOR) with the left array of data. Once the output of the XOR calculation is done the left array and the right array (which should be unchanged since it is used by the round function but not changed) switch positions. The new left array becoming the right array (R_1), and the unchanged right array becomes the left array (L_1). This completes one iteration of the loop. The loop is then repeated 15 more times. The new right array is used as an input with the 2^{nd} subkey to generate the new round function output. This output is XOR'd with the left array, and the new left array is switched with the right array. This is repeated 16 times until all the subkeys have been used. The only change is that the left and right array are not switched at the end of the 16^{th} iteration of the loop.

The final step is to take the left and right array, that was not switched after the last iteration, and combine them to a single array. This array then goes through a reorganisation known as the final permutation (FP) table. The reason the left and right arrays are not switched after the last iteration of the loop is the FP table has this switch already built into its values. Once the table has reorganised the data the plaintext has been encrypted using DES. This process is shown in Figure 26.

Decryption uses the same algorithms and functions as the encryption, the only difference is that when performing the Feistel loop the subkeys are used in inverse order. To decrypt the data, the subkey is generated in the same way as the encryption. Then the encrypted data is put through the Feistel loop function the only change required is that for the first round you use the 16^{th} subkey instead of the first for the round function, second iteration you use the 15^{th} subkey for the round function. This will continue in the same fashion until the sixteenth iteration uses the 1^{st} subkey in the round function. This is shown in Figure 27.

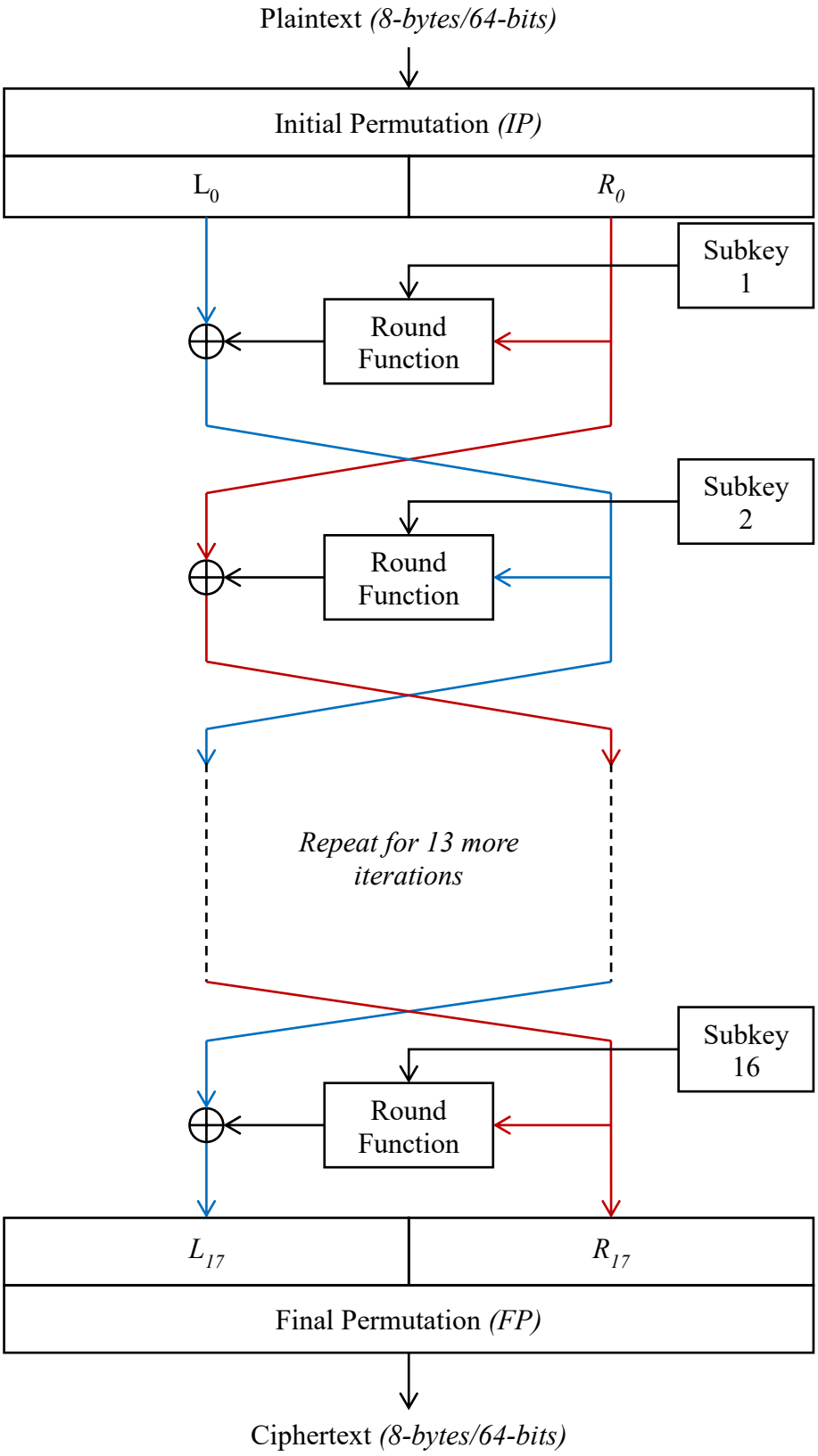

Figure 26
Feistel loop for DES encryption

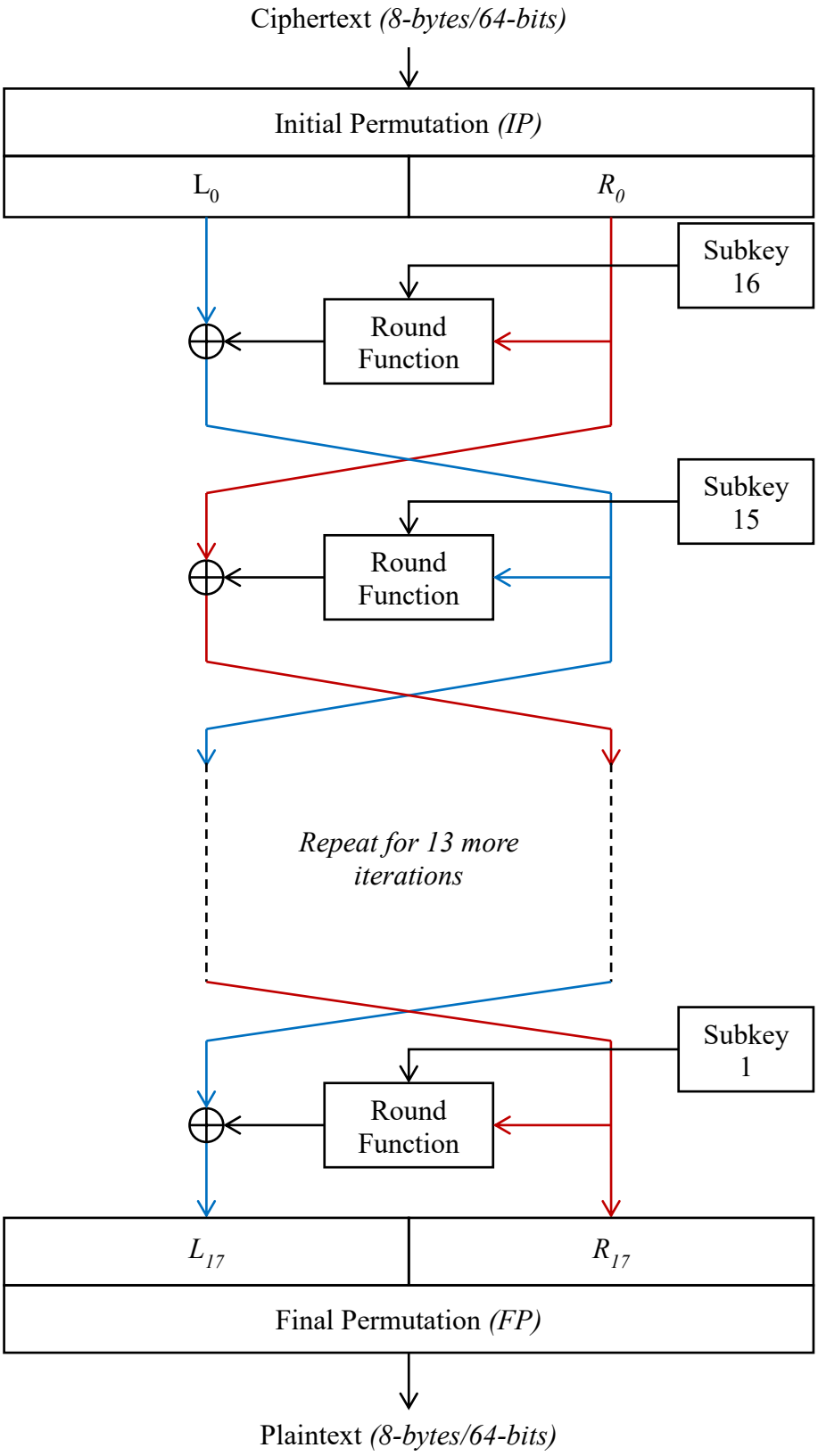

Figure 27
Feistel loop for DES decryption

3.3.1.c Round function

The round function is an important part of the Feistel loop. The round function is done sixteen times in the Feistel loop and uses the 16 subkeys that are generated by the key schedule.

The round function operation starts by taking the right array passed to it by the Feistel loop and performs a reordering table called the expansion (E) table. This table is like the other reordering tables but also expands the bit array from 32-bits to a 48-bit array to match the size of the subkey.

Once the right array is expanded it is XOR'd with the passed subkey. The result of the XOR is then split into 8 arrays of 6-bits in length. These arrays all go through substitution tables, each array of the 8 have its own substitution table (S-Box 1 to 8). These arrays operate by taking the value of the 6-digit array and replacing it with a specific 4-bit value depending on each S table. The table works by separating the 6 bits into two groups. The X group which is most significant bit and the least significant bit. The Y group is made of the 4 remaining middle bits. The 6 bits make up the form $X_0\ Y_0\ Y_1\ Y_2\ Y_3\ X_1$. Each X group has 16 values depending on their Y values. This assigns values to all 64 combinations of 6-bits. Each value is assigned a different 4-bit value depending on the table. The most significant 6-bit array uses the first substitution table, the group of 6 bits uses the second substitution table. this continues the until the least significant 6-bit array uses the eighth substitution table. All the 4-bit arrays are then concatenated, S-Box outputs 1 – 8, and go through a final reordering table, the permutation (P) table. This result is then used in the Feistel loop. A visual representation of the round function is shown in Figure 28.

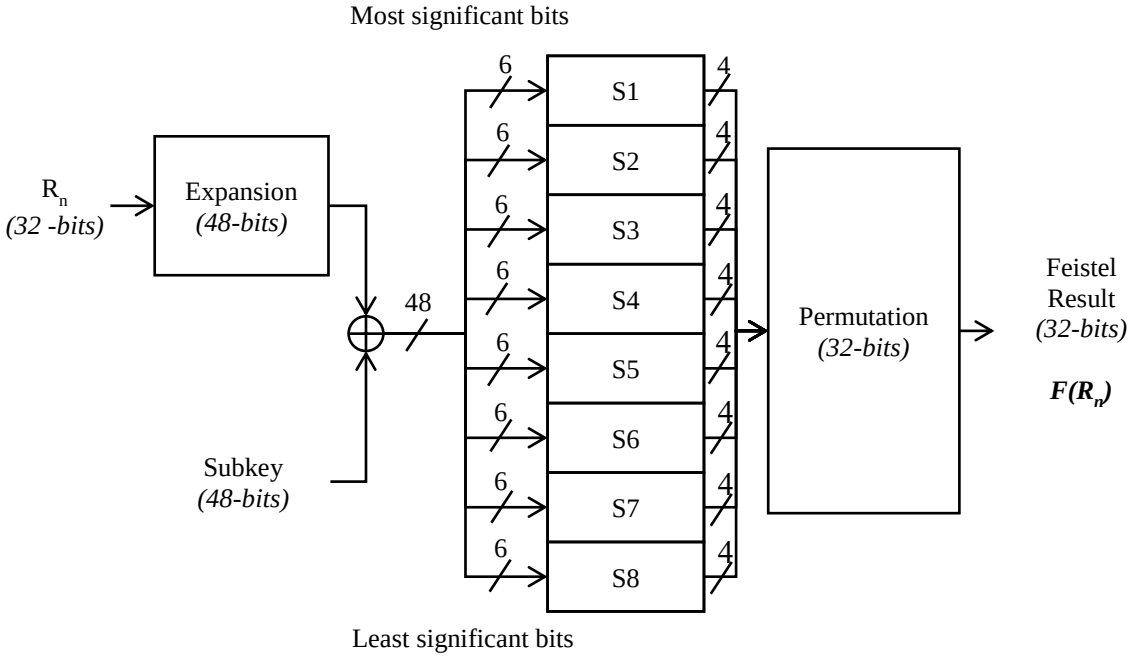

Figure 28
Round function

3.3.1.d 3DES algorithm

The process described earlier is for DES encryption, this system used 3DES. This encryption algorithm is an improved version of DES that is backwards compatible. The process is done with three unique keys and puts the plaintext through three iterations of the DES algorithm. The first iteration is the DES encryption process using the first key. The second iteration runs the new ciphertext through DES decryption, instead of encryption, but using a second key. The final step is to take the data and put it through another DES encryption, using a third key. This process is a lot more secure than normal DES. A visual representation is shown in Figure 29.

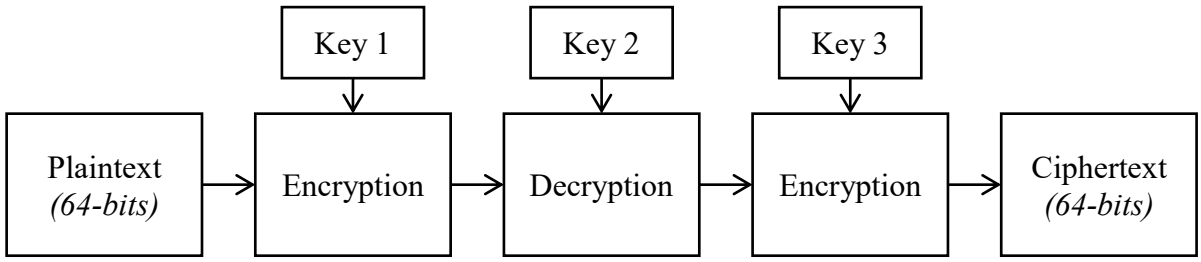

Figure 29
3DES encryption process

The decryption process for 3DES is very similar but the reverse operations are done with the keys used in opposite order as shown in Figure 30.

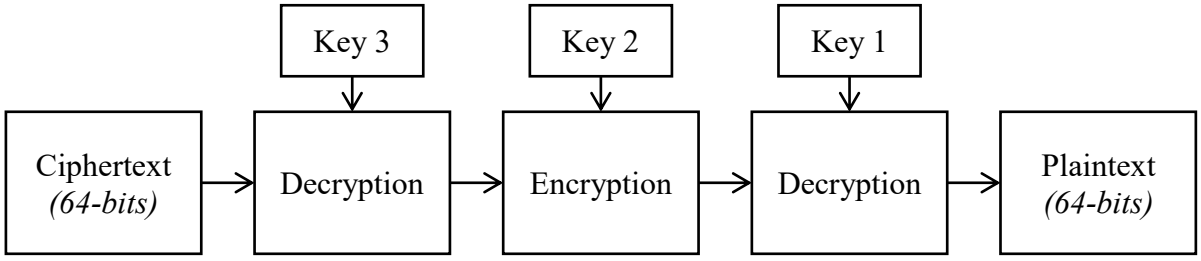

Figure 30
3DES decryption process

3.3.2 ENCRYPTION SIMULATION

The approach was to build the cryptographic algorithm in Python to help ease the development of the system for later use in the smart card. Once the Python system was verified as working correctly it allowed for debugging by comparing the results, with the smart card version, at different points of the algorithm.

The 3DES process was discussed earlier so a breakdown of the functions and methods used to achieve this system is shown in Table 10. The NumPy library was used to ease the development of the algorithm for the smart card usage. The algorithm required to operate on an 8-bit device, meaning it had an 8-bit variable limit, so NumPy allowed for creation of variables of specific bit size. Specifically, uint8 was used. Some array functions from NumPy were used, specifically pack bits and unpack bits, allowing for easier conversion from byte values to bit values.

Methods/Functions	Description
+ **bit2byte_1DArr (**bitArr**)**	This method takes a bit array input of any index size multiple of 8 and returns an array with all byte values of every 8-bits from most significant to least significant bit concatenated into a single array.
+ **byte2bit_1DArr (**byteArr**)**	This is the opposite of the bit2byte_1DArr. This takes an input of an array of bytes, converts every byte into 8-bits and concatenates each bit to an array. This bit array is returned, always being 8 times the index size of the input byte array.
+ **key_AddPad (**key_plain**)**	This function is used to pad any key that is less than 8-bytes in length so that it can be used in the algorithm.
+ **plaintext_AddPad (**message_plain**)**	This function is used to pad any message part that isn't 8-bytes long. This is to ensure that that message byte length is divisible by 8, allowing the algorithm to work correctly.
+ **encryptionDES (**key_byteArr, message_byteArr**)**	This function performs a single iteration of DES encryption, taking in a key and plaintext to encrypt. The ciphertext is returned.
+ **descryptionDES (**key_byteArr, message_byteArr**)**	This function performs a single iteration of DES decryption. It receives a key and message variable. The plaintext is returned.
+ **encryption3DES (**key1_byteArr, key2_byteArr, key3_byteArr, message_byteArr**)**	This performs a 3DES encryption iteration. Performs a DES encryption with key1, decryption with key 2, and encryption with key3 on the message parameter.
+ **decryption3DES (**key1_byteArr, key2_byteArr, key3_byteArr, message_byteArr**)**	This performs a 3DES decryption iteration. Performs a DES decryption with key3, encryption with key 2, and encryption with key 1 on the message parameter.
+ **shift_left (**inputBitArr, subkey_num**)**	This performs a circular left shift on the input bit array it receives. It performs this circular left shift 1 or 2 times depending on the subkey number.
+ **reordering_table (**input_bitArr, table**)**	This performs the reordering table function. It takes the input bit array and the reordering table to be used. It creates a new array of correct size and returns this array with the reordered values.
+ **sbox_getIndex (**x0, y0, y1, y2, y3, x1**)**	This takes a 6-bit input and finds the index number to use for a substitution block
+ **substitution_sbox (**input_bitArr**)**	Function takes the entire 48-bit array and runs each 6-bit through the required S-Box after getting the required index, performing all substitutions. Converts the S-Box values from decimal to the 4-bit numbers. Returns the final 32-bit array.

+ **key_schedule (**key_byteArr**)**	This function performs the entire key schedule function, taking in an 8-byte key and returns a 16 by 6-byte array. 16 subkeys, 6 bytes each.
+ **round_function (**re_byteArr, sub-key_byte**)**	This performs the round function needed for the Feistel loop algorithm. It takes the right half array and the subkey for the round and performs the round function, using the S-Box functions and reordering tables, returning a 4-byte array.
+ **feistel_loop (**message_byteArrInput, subkeys_byteArr**)**	This is the major function for both encryption and decryption. Taking in a message in a byte array and the 16 subkeys. Performs all 16 loops of the Feistel function and its necessary reordering tables and XOR functions. This returns a byte array of the data.
+ **main_function_ECB (**key_1_byteArr, key_2_byteArr, key3_byteArr, plain_byteArr**)**	This is the main function for the entire 3DES system. This function checks key and message length and uses the padding functions if necessary. It also splits the message into 8-byte size blocks for encryption and runs all the necessary encryption and concatenates the final encrypted message. It then runs the final message through decryption and takes that result and compares it with the original message to check if the algorithm is performing correctly.

Table 10
Breakdown of Python 3DES functions and methods

3.4 SMART CARD HARDWARE, FIRMWARE, AND SOFTWARE

The initial approach was to use an off the shelf smart card and a smart card operating system. This approach changed due to a few unforeseen problems, but the main problem was that the encryption algorithm required more flash memory to operate then the available smart card. Smart cards have very limited processing and memory capabilities, so a decision was made to build a smart card for the system from first principles to provide the processing requirements.

3.4.1 THEORY

Many smart cards have built in features as described in the literature review. So, a microcontroller providing many of these features would allow for smarter development. An ATMega1284P was used, the same microcontroller as the terminal. This was used since it was readily available with the required programmer. It has all the features required for a smart card, such as EEPROM, internal interrupt system, communication systems etc. Some of the optional features that are sometimes implemented in smart cards was also available to be implemented.

3.4.2 CIRCUIT DIAGRAM

Many of the systems were duplicated from the terminal to save on development time. The 16 MHz external oscillator was used to allow for quicker processing of the encryption system and to stabilise the UART communication. A picture of the final circuit is shown in Figure 31. A few of the hardware features are discussed later in the smart card interface section.

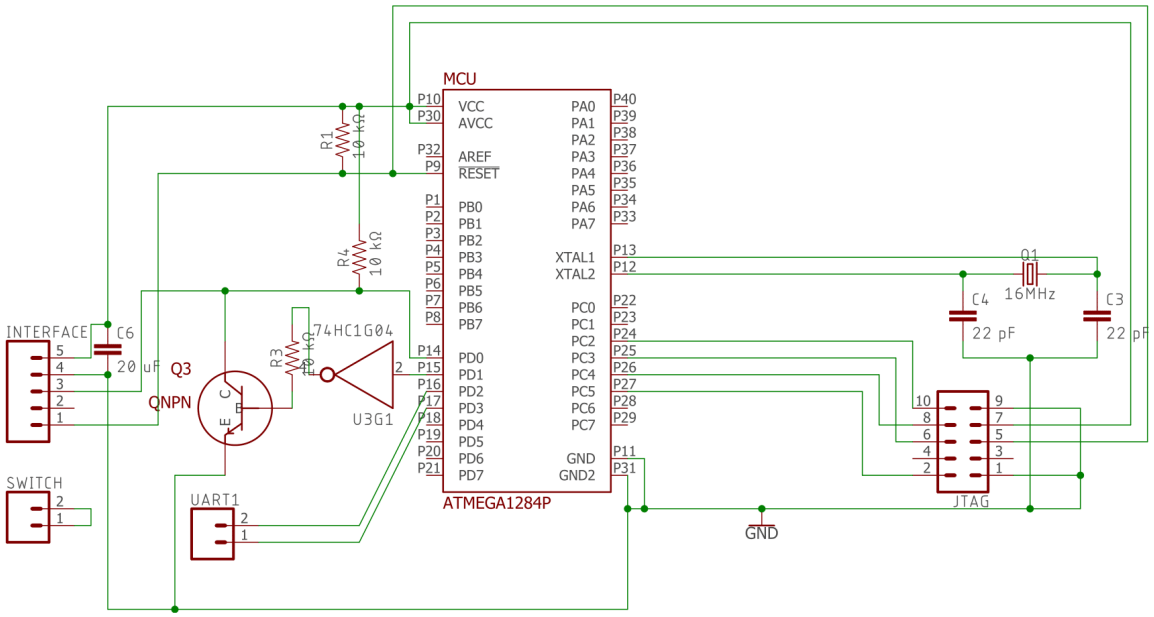

Figure 31
Smart card circuit diagram

3.4.3 SMART CARD FIRMWARE

The smart cards main requirements were that it could run the 3DES system for both encryption and decryption, communicate with the smart card terminal, read and write from its EEPROM, check the balance on the card, and deduct from that balance.

The smart card operated in a linear fashion. When the card started, it waited for a message to be received from the terminal, until the correct start message had been received it did nothing. Once it received “START” it replied with an acknowledge “ACK” to the terminal. The transaction then began. The smart card waited for the transaction cost to be passed with the message “COST:X” with X being the value. This value was then stored for later use. The smart card then sends a “PIN” request waiting for the user pin to be received. Once it is received the EEPROM memory was decrypted with the key and verified if the PIN received was correct. This verification was done by checking that the first 8 memory bytes in the EEPROM were the same as the received PIN from the terminal. If the PIN was incorrect, the smart card would decline the payment by sending “DECLINE” to the terminal. The second memory line stored the balance of the card and would be checked against the received cost to verify there was enough balance on the card. If there wasn’t enough balance “INSUFFICIENT” would be sent to the terminal, otherwise “ACCEPTED” would be sent and the cost deducted from the balance and the new balance encrypted and written to the EEPROM. Once this was done a “OFF” message was sent from the smart card to the terminal. A representation of the flow of the system is shown below in Figure 32.

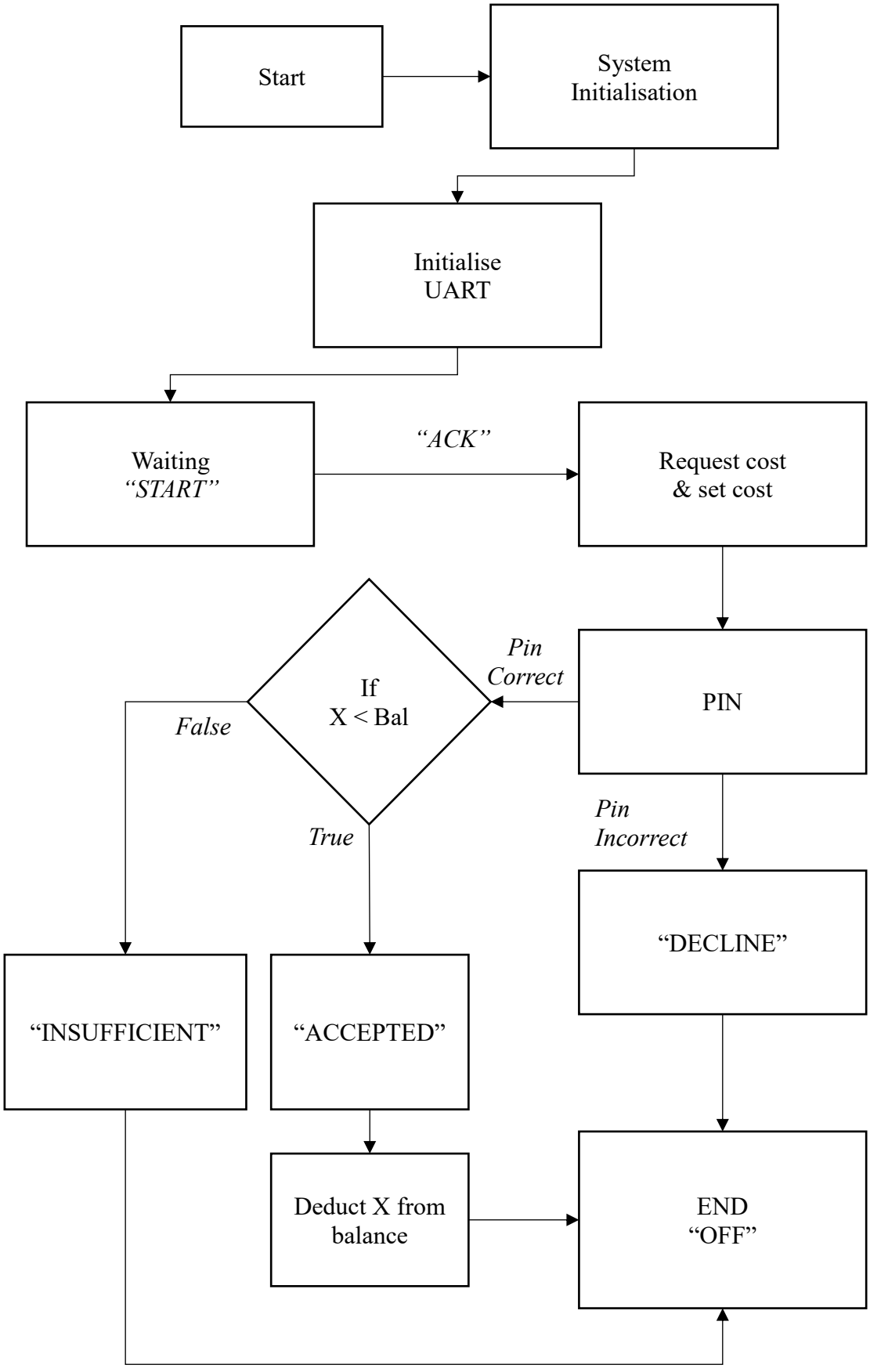

Figure 32
Smart card flow chart

3.4.4 CIRCUIT PICTURE

A picture of the final smart card circuit is shown below in Figure 33.

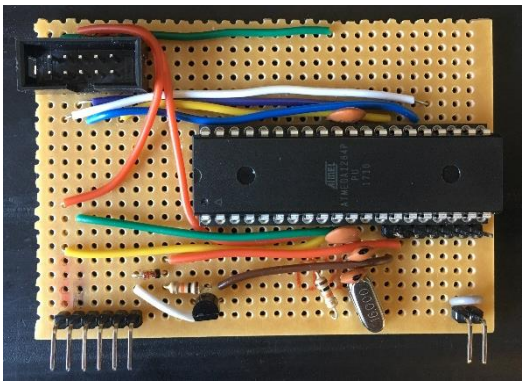

Figure 33
Photo of smart card circuit

3.4.5 ENCRYPTION SOFTWARE

The smart card operated in C so a 3DES version was written for it. It was developed as a separate library to allow for simple integration by passing a few variables and receiving the output. Many of the functions were adapted from the Python code and adapted for the different environment with the restrictions that C has with variable types and sizes.

There were two functions and a aspect that needed to be adapted for a C implementation of the 3DES encryption from the Python code. The first was the pack bits and unpack bits were created from first principle since there is no equivalent methods in C. The second thing is the memory management and handling of variables in different methods is different.

The memory management of the system was the most important thing to adapt first. Arrays in C work very differently than in Python where most of the memory management is handled by the compiler. The need to create new arrays for this encryption is important since new datasets are calculated many times throughout the encryption. C doesn't support the ability to return arrays, only pointers, so a pointer to an array can be used but these arrays need to have the memory manually assigned in the function. Doing manual assignments for this method works but this removes the memory management of the compiler. If a pointer to array has memory assigned in a function and then the pointer is returned to the calling method, the memory needs to be manually released or a memory leak will occur. The other method is to create the required arrays in the main method before the function and pass the array by reference. This allows for the compiler to manage the memory still and removes the possibility for memory leaks to occur. This method is a lot safer but is more memory intensive since these arrays will remain in memory until the end of the function, but the smart card microcontroller supported a large amount of flash memory, so this method was implemented.

The Python simulation of the encryption had libraries to perform the pack bits and unpacks bits so a C version to satisfy the encryption system was written. The pack bits system worked by taking a bit array, the size of the bit array, and the byte array to append to. This was done specifically for 8-bits to 1-byte. It operates by using a nested loop system. It runs through groups of 8-bits and takes the values and calculates the exponent values of 2 for the bit value by left shifting an 8-bit number and multiplies it by the binary value (1 or 0). This is then stored in the

byte array that was passed by reference, so it can be used by the calling method. The code for this function is shown below in Figure 34.

```
void bit2byte_1DArr(uint8_t bitArr[], size_t bitArr_Size , uint8_t byteArr[])
{
	// Byte, Bit Counter
	size_t i, j;
	for(i = 0; i < (bitArr_Size / 8); i++)
	{
		uint8_t num = 0;
		for(j = 0; j < 8; j++)
		{
			// Perform 2^bit via left shift method
			// Add all the bits to num
			num = num + (1 << (7 - j)) * (bitArr[8*i + j]);
		}
		// Store the value of num in the byte array
		byteArr[i] = num;
	}
}
```

Figure 34
Code snippet for converting bit array to a byte array

The unpack bits is done through a similar process. It runs through a nested loop system by running through each byte and converting the byte value to each of the 8-bits. It gets the binary value for each bit by taking the byte value and performing a modulus calculation with the number 2 to determine a remainder of 1 or 0. Once the remainder is determined, the byte value is divided by 2 and rounded down this. This is done eight times appending each bit to the bit array passed by reference. The code for this function is shown below in Figure 35.

```
void byte2bit_1DArr(uint8_t byteArr[], size_t byteArr_Size, uint8_t bitArr[])
{
	// Byte, Bit Counter
	size_t i, j;
	for(i = 0; i < byteArr_Size; i++)
	{
		uint8_t bin = 0;
		for(j = 0; j < 8; j++)
		{
			/*
			Convert bytes to bits by performing remainder
			calculations on 2.
			*/
			bin = (byteArr[i] % 2);
			byteArr[i] = byteArr[i] / 2;
			/*
			Each bit is added to the bit array, forming 1
			array for all bytes.
			*/
			bitArr[(i*8) + 7 - j] = (bin);
		}
	}
}
```

Figure 35
Code snippet for converting a byte array to a bit array

Table 11 shows a breakdown of all the functions with a description of what they do. Many of the functions are like the Python versions but are adjusted to work in a C environment and its restrictions.

Methods/Functions	Description
+ **bit2byte_1DArr (**uint8_t bitArr[], size_t, uint8_t[]**): void**	Converts bit array to byte array. Appends values to array past by reference.
+ **byte2bit_1DArr (**uint8_t byteArr[], size_t, uint8_t[]**): void**	Converts byte array to bit array. Appends values to array past by reference.
+ **flip (uint8_t[16][6], uint8_t[16][6]): void**	Array to reverse the order of an array. Used to reverse the order of the subkeys for decryption.
+ **encryption3DES (**uint8_t[8], uint8_t[8], uint8_t[8], uint8_t***): void**	Performs the encryption for 3DES, taking in 3 keys and the plaintext. The keys and messages are padded and message split into 8-byte groups for encryption. The data is then written to the EEPROM.
+ **decryption3DES (**uint8_t[8], uint8_t[8], uint8_t[8], uint8_t***): double**	Performs the decryption for 3DES, taking in 3 keys and the EEPROM memory. The keys and messages are padded and message split into 8-byte groups for decryption. The pin is verified, and balance returned.
+ **shift_Left (**uint8_t[28], uint8_t, uint8_t**): void**	Performs circular left shift on the array that is passed by reference. Takes in the rounds left shift array and the current round.
+ **reordering_table (**uint8_t[], uint8_t[], uint8_t*, uint8_t**): void**	Performs the reordering table function. Taking in the original array, the reordering table pointer and reordering table size and the output array reference.
+ **sbox_getIndex (**uint8_t, uint8_t, uint8_t, uint8_t, uint8_t, uint8_t**): uint8_t**	Takes a 6-bit number and finds the S-Box index to be used in the S-Box substitution table.
+ **substitution_sbox (**uint8_t[64], uint8_t[32]**): void**	Performs the S-Box substitution, taking in the input array and adds the output to an array passed by reference.
+ **key_schedule (**uint8_t[8], uint8_t[16][6]**): void**	Takes in the 8-byte key and returns a 2D array with 16 keys of 6-bytes large.
+ **round_function (**uint8_t[4], uint8_t[6], uint8_t[4]**): void**	Takes the input array, subkey and output array reference and performs the round function.
+ **feistel_loop (**uint8_t[8], uint8_t[16][6], uint8_t[8]**): void**	Takes the plaintext/ciphertext, subkeys array and the output array reference and performs the Feistel loop algorithm.
+ **main_function_ECB (**uint8_t[8], uint8_t[8], uint8_t[8], uint8_t***): void**	Function used for debugging and verification of both encryption and decryption.
+ **USART_Transmit_Char (**char**): void**	Used for debugging. Transmits char characters over the UART.
+ **USART_Transmit_String (**char***): void**	Used for debugging. Transmits strings over the UART.

Table 11
3DES library breakdown

3.5 SMART CARD INTERFACE

3.5.1 SMART CARD INTERFACE THEORY

The interface design needed to meet three requirements:

- Voltage and current supply,
- Single data line half duplex communication, and
- Clock signal for smart card operation.

A switch system to detect the smart card and a reset signal was also implemented into the interface to allow for easier interfacing.

3.5.2 SMART CARD INTERFACE CLOCK

This external oscillating clock was implemented using the smart card terminals PWM. The PWM used was the clear timer on compare match (CTC) mode, this worked by counting to a set value and when the value is reached the signal oscillates and the counter value resets. The frequency that that PWM oscillates is shown in Equation 3 taken from the ATMega1284P datasheet [13].

$$f_{OCnA} = \frac{f_{clk\_I/O}}{2N(1+OCRnA)}, \tag{3}$$

The variables in the equation are detailed below.

- f_{OCnA}, PWM oscillating frequency.
- $f_{clk\_I/O}$ systems oscillating frequency, 16 MHz in this system.
- N is a prescaler factor value.
- OCRnA is a counter value for the compare mode.

The code snippet for initialising the PWM is shown in Figure 36. The microcontroller manages the rest of the PWM.

```
void Init_PWM()
{
	// Set output port for PWM
	DDRD |= (1 << PWM_Port_P);
	// Set counter value
	OCR1A = 4;
	// Enable CTC mode and fast PWM
	TCCR1A |= (1 << COM1A0)|(1 << WGM11) | (1 << WGM10);
	TCCR1B |= (1 << WGM12)|(1 << WGM13)|(1 << CS10);
}
```

Figure 36
PWM initialisation code and values

The maximum frequency for the PWM is shown in Equation 4.

$$f_{OCnA} = \frac{f_{clk\_I/O}}{2}, \tag{4}$$

The maximum frequency that this system could operate is calculated to be 8 MHz. This was built to allow interfacing with other smart cards, but the internal 16 MHz of the built smart card was used in the final system to improve on the system speed and stabilise the UART communication.

3.5.3 SMART CARD INTERFACE DATA

The design for the smart card interface data signal was to be able to use the UART module on the terminal with the TX and RX lines connected. This would allow for the single line communication in half duplex mode that smart cards operate. A logic level converter would also be required if the smart card and terminal operated at different voltage levels for serial communication, but they both operated at 5 V, so this was not implemented.

The problem with connecting the RX and TX lines together is that the TX line is held in a high voltage state until the start bit is sent, which pulls the TX line down but when it is connected to another TX line being held high this doesn't work correctly. So, a system was developed to prevent this problem from occurring. First a NPN transistor was connected to the circuit acting as a switch between the RX line and ground with the TX line acting as the switch controller at the base. An inverter and resistor were connected to the TX line and the base of the transistor. This allowed for the TX signal to be transmitted over the RX line by pulling it to ground for the inverted TX signals. A pull up resistor was used to ensure the RX line remains high instead of in a middle state. The problem with this circuit is that a local echo of all transmissions is received. Since this system is for half duplex communication this wasn't a problem. The microcontroller UART could be set to half duplex mode or the echo can be used for verification of correct message being sent. This system was copied and applied to the smart card to simplify the system implementation of the smart card. It could have been applied as a software based system to achieve the same results but since a working circuit was developed, it was used to save on development time.

The settings used in the ISO-7816 standards for smart card communications were used, except for the synchronisation bits which are meant to be 1.5-bits, but 2-bits still operate correctly if the interfacing device doesn't support half bits. These settings are used on the smart card and the smart card terminal.

Data bits	Parity bits	Synchronisation bits	Baud rate	Flow control
8-bits	2-bits	2-bit	9600 bps	None

Table 12
Smart card serial communication settings

3.5.4 SMART CARD INTERFACE POWER, SWITCH, AND RESET

The power for the smart card was achieved using the same voltage regulator system as the terminal. It takes in the 9 V battery and provides a smooth power output for the smart card. The reset signal is connected directly to smart card terminal microcontroller output pin and the reset line of the smart cards microcontroller. This allowed the terminal to reset the smart card by pulling the signal low. The switch operated by using a digital input on the smart card terminal with a data signal held high. When the smart card was connected that signal was grounded, and the smart card terminal knew the smart card is connected.

3.5.5 SMART CARD INTERFACE CIRCUIT DIAGRAM

The final circuit diagram with the power system, clock, reset, switch, and UART data communication is shown below in Figure 37.

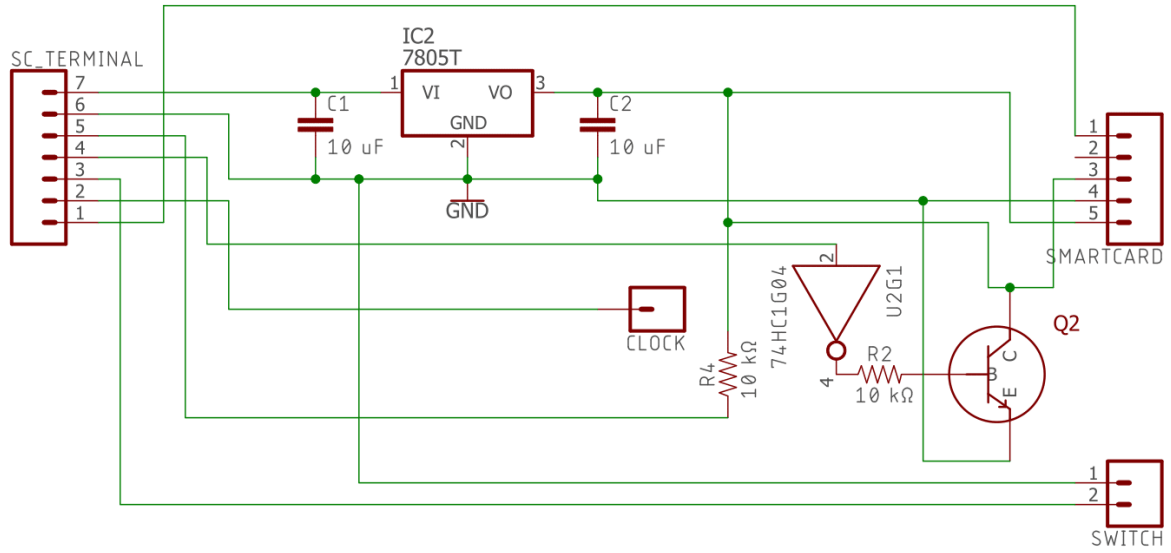

Figure 37
Smart card interface circuit diagram

3.6 SMART CARD TERMINAL SOFTWARE

The smart card terminal firmware is made up of the initialisation of all the systems and peripherals used in the smart card terminal and the main loop that controls and operates the entire system. The communication protocol for the smart card terminal was made up of a variety of different commands since the terminal interfaced with all the other subsystems. The terminals full communication protocol is shown below in Table 13 with a short description on each messages purpose and what subsystem devices it was intended for.

Message	Sent/Received + Device	Description
$$$	Sent - Bluetooth module	This is sent to the Bluetooth module to enter command mode.
CMD	Received - Bluetooth module	This is received from the Bluetooth module when the command mode is successfully entered.
---	Sent - Bluetooth module	This is sent to the Bluetooth module to exit command mode and return to normal mode.
END	Received - Bluetooth module	This is received from the Bluetooth module when command mode has been quit.
AOK	Received - Bluetooth module	This is received when a successful command has been sent and run.

K,	Sent - Bluetooth module	This is sent to the Bluetooth module to close the current connection, when a transaction completes or cancelled.
KILL	Received - Bluetooth module	This is received from the Bluetooth module when the current connection has been successfully closed.
R,1	Sent - Bluetooth module	This is sent to the Bluetooth module to restart the device. Used for debugging.
PAYMENT REQUEST	Received - Smartphone application	This is received from the smartphone application to start the payment request process.
NAME:(**X**)	Received - Smartphone application	This is sent from the smartphone application with the transaction/product name in variable **X**.
GETNAME	Sent - Smartphone application	This is sent to the smartphone from the terminal to retrieve the transaction/product name for the payment request.
COST:(**X**)	Sent & Received - Smartphone application	This is sent from the smartphone application to the smart card terminal with the cost of the payment request in variable **X**. The terminal also sends it to the smart card, so the balance can be compared and deducted.
GETCOST	Sent - Smartphone application	This is sent from the smart card terminal to the smartphone application to get the cost of the transaction.
ACCEPTED	Sent & Received - Smartphone application and smart card	This is sent from the smart card to the terminal and terminal to the smartphone to inform the devices that the transaction was successful.
CANCEL	Sent & Received - Smartphone application and smart card	This is sent from the smartphone application and the smart card terminal when a payment request transaction is cancelled.
FAILED	Sent & Received - Smartphone application and smart card	This is sent from the smart card and terminal when a payment request transaction is failed.
PIN:(X)	Sent - Smart card	This is sent from the smart card terminal to the smart card to decrypt the balance from the smart card.

Table 13
Terminal communication protocol

The initialisation starts by starting the LCD, which shows a start-up and loading message on the screen. The keypad is then loaded and the compare timer2 for the keypad debouncing is started. The smart card interface detection switch and serial is loaded and started, followed by the serial communication for the Bluetooth module and the settings and timers for the PWM. Once all the systems are loaded, the interrupts are enable by setting the global interrupts to on. The initialisation of the system is shown in Figure 38.

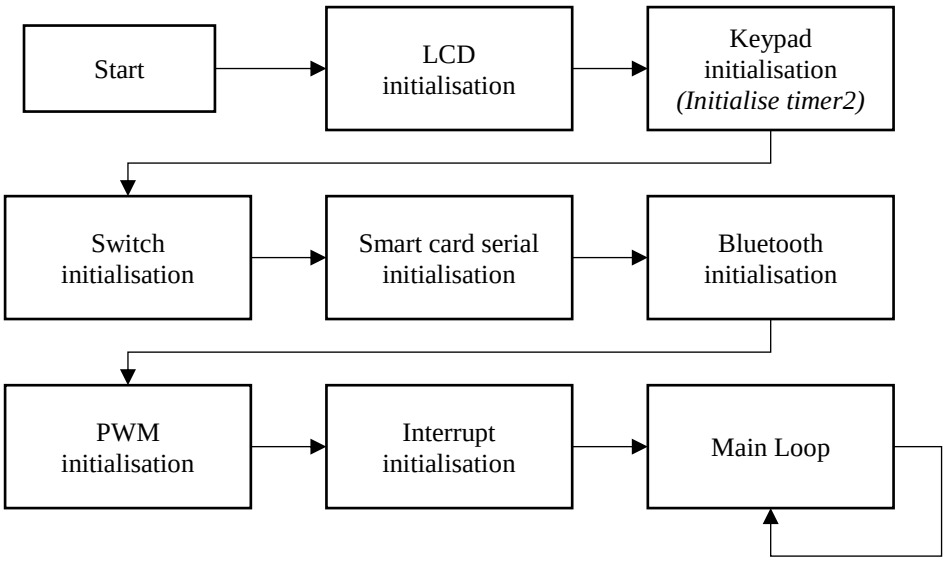

Figure 38
Smart card terminal start-up

The main loop is the part of the system that controls the transaction process. It starts by checking if a serial message has been received from the Bluetooth module, if so it performs the functions attached to each message, such as cancelling the transaction. If no message was received or the message function has been completed the device reads if a key has been pressed on the keypad, taking in which key was pressed or setting the key value to null if no key was pressed. The system then uses a state system to determine what part of the payment process the system is in. The main menu state is an idle state that waits for a smartphone application to connect and send a payment request. The main menu will also reboot the Bluetooth module if the ‘C’ key on the keypad is pressed. The request state is when a payment request has been received and asks the user if they want to accept the payment. Pressing ‘A’ will move the state into the get pin state. Pressing ‘C’ will cancel the payment request and disconnect from the smartphone application. The get pin state asks the user to insert the smart card and enter the pin for the smart card. Pressing ‘A’ will send the pin through if it is 4-6 digits long and move the system to the wait state, ‘B’ will clear the entered pin to be typed again, ‘C’ will cancel the transaction, and any number key will add to the pin number. The wait state will be idle until the smart card responds with accept, decline, or insufficient and then transition to the remove card state asking for the smart card to be removed and then start the application at the main menu state again. This process is shown in Figure 39.

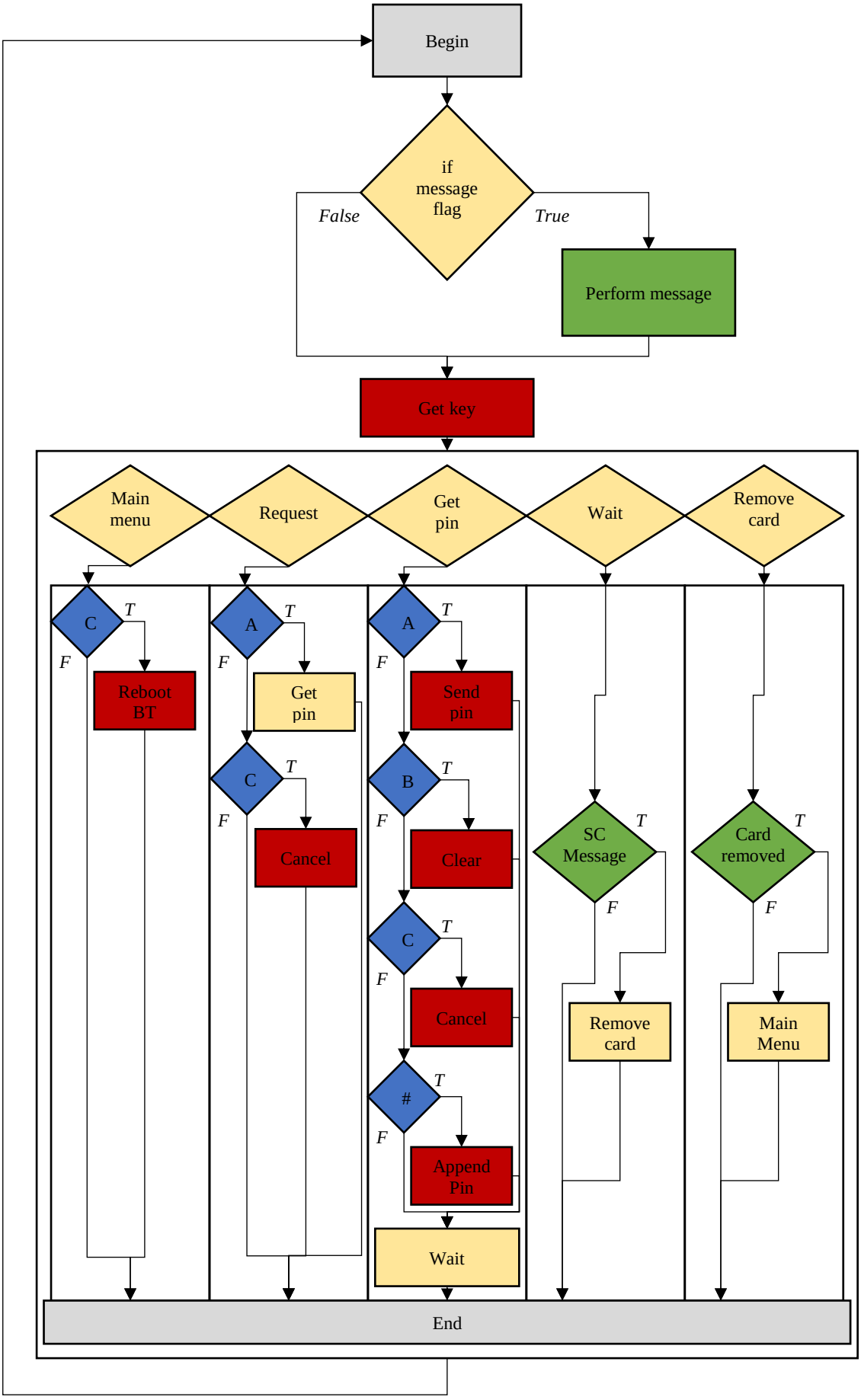

Figure 39
Smart card terminal software diagram

Many functions are used to perform the smart card terminals initialisation and main loop, these are shown in Table 14 with a short description on its purpose.

Methods/Functions	Description
+ Initialise(): void **+ Init_PWM(): void** **+ Init_Timer2(): void** **+ Init_UART0(**unsigned int**): void** **+ Init_UART1(**unsigned int**): void**	These functions all perform different initialisation parts of the smart card terminal system. UART0 is for the Bluetooth module and UART1 is for the smart card serial communication.
+ UART0_Transmit_Char(char data**): void** **+ UART0_Transmit_String(**char* string**): void** **+ UART1_Transmit_Char(**char data**): void** **+ UART1_Transmit_String(**char* string**): void** **+ UART1_Receive_Char(): char** **+ UART1_Receive_String(): void**	These functions perform many of the serial communication for both the Bluetooth module and the smart card. The transmit either characters or strings, or read character or strings from the receive buffer.
+ UART0_CMD(): void **+ UART0_KILL(): void** **+ UART0_RESTART(): void**	These are all commands used for the Bluetooth module. Entering command mode, ending the current connection, and restarting the module.
+ Keypad_GetKey(void): char	This is the keypad debouncing and reading system that was described earlier in the keypad section of the report.
+ state_MainMenu(): void **+ state_Request(): void** **+ state_GetPIN(): void** **+ state_Wait(): void** **+ state_RemoveCard(): void**	These functions set the state for the main loop, display the relevant data on the LCD and set and clear any relevant data values.
+ clear_Pin(): void	This is used to clear the pin when the states transition or when the user clears it on the terminal.
+ sc_Connected(): bool	This function determines if the smart card is currently connected by returning a Boolean value on its connection state.

Table 14
Smart card terminals function and method description

3.7 DESIGN SUMMARY

Table 15, lists all the tasks that were required for this system, if they were completed and how they were implemented.

Task	Implementation	Task completion
Design and construct a smart card terminal device that can facilitate communication between a smartphone and smart card device.	A smart card terminal was built using a microcontroller and some other operating hardware.	Completed
Design and construct a HMI for the smart card terminal to allow for instructions and commands to be displayed and received by the smart card terminal.	A keypad and LCD were used as the HMI and were bought off the shelf and all software and libraries for the keypad and LCD were developed from first principle in C.	Completed
Design and develop a smartphone application that can successfully communicate with a smart card terminal.	An Android smartphone application was developed that interfaced with the smart card terminal over Bluetooth.	Completed
Design a communication protocol between the smartphone application and smart card terminal.	A simple communication protocol was designed and built to facilitate communication between the smartphone application and the smart card terminal.	Completed
Design and develop a GUI that can facilitate a working HMI between the smartphone application and its user.	A GUI was built for the smartphone application using XML to allow for information to be displayed to the user and take in commands through systems such as buttons.	Completed
Development of an encryption system from first principles in a simulation platform to ease development for the smart card system.	An investigation was done on which algorithm best suited this system which ended up being 3DES. A Python simulation was developed from first principles to help understand, verify and ease development of the encryption system on the smart card.	Completed
Develop an encryption system from first principle that can operate on an 8-bit smart card device.	The 3DES encryption system was written in a C library and supported 8-bit microprocessor operation if there is enough flash memory.	Completed
Design and construct a smart card interface to power and successfully communicate with a smart card device.	A smart card interface was designed and implemented from first principles providing the required power and communication signals. Other features were added to allow for easier interfacing with the smart card device.	Completed
Design and build a smart card that can store encrypted data on its EEPROM and perform decryption and encryption on that data.	A smart card was designed and built from first principles using a microcontroller with built in EEPROM and some other hardware devices.	Completed
Design a communication protocol to facilitate communication between the smart card and smart card terminal.	A communication protocol was developed to facilitate communication between the smart card and smart card terminal	Completed
Design and develop a smartcard	A software system was developed in C	Completed

software control algorithm to perform the smart cards payment functionality.	to allow for transactions to be made and the balance deducted from an encrypted EEPROM.	
Design and develop the smart card terminal control algorithm to manage and control the entire system state and facilitate the entire transaction process.	A software and firmware system were developed from first principles to control the smart card terminals hardware and manage the entire systems state and current process. This was developed using C.	Completed

Table 15
Design summary

4. Results

4.1 SUMMARY OF RESULTS ACHIEVED

Description of requirement or specification (intended outcome)	Actual outcome	Location in report
Mission requirements of the product		
The system must be able read from and write to the chip smart card.	The system used a smart card developed from first principle, so an actual chip smart card was not used. The developed smart card was designed using the same interface system as a chip smart card and could successfully interface.	Section 4.2.1
The system must interface with a smartphone application.	The smart card terminal could interface with the smartphone application using a communication protocol that was developed from first principle.	Section 4.2.3 Section 4.2.6
The smartphone application must be able to request payments from the terminal and keep a digital receipt of the transaction.	The smartphone application was built allowing for payment requests, the ability to cancel payment requests at any time, and a file with all successful payment history.	Section 4.2.6
The system must use a secure PIN system for customer payments.	The system used 3DES encryption with a 4 to 6-digit pin to verify the smart card payment.	Section 4.2.4
The system must connect with a smartphone via a wireless interface.	The smart card terminal used Bluetooth 2.0 to interface wirelessly.	Section 4.2.3
The system must provide a HMI for the customer to confirm payments and enter PINs.	The system used a 16 by 2-character LCD to display information on the current system state and some simple instructions. A 16-key keypad was used to allow for pin entry and some other commands.	Section 4.2.6
The system must be battery powered and rechargeable.	The system used two 9 V rechargeable battery to power the smart card terminal and smart card interface.	Section 4.2.5
Field conditions		
The system must operate indoors in temperatures ranging from 10 ℃ and 35 ℃ and humidity less than 60%.	The system was never tested under extreme conditions but did operate indoors with temperature ranges from 14° - 32° C.	---
The system wireless functionality must still operate in a non-lab environment.	The systems Bluetooth system operated correctly in non-lab environments.	---
Specifications		
The smart card interface must provide the correct operating voltage (4.5 - 5.5 V), current (50 mA) and clock (1-5 MHz) to	The interface provided a voltage of 5.04 V and enough supply current for operation. A clock signal of 1.578 MHz is generated by the microcontroller. This clock signal	Section 4.2.1

interface correctly with a smart card.	was not used in the smart card operation since a smart card was developed from first principle and used an internal clock.	
The system must correctly read a string of 256 Bytes from the smart card, this data should remain consistent every time the card is read.	The system could correctly read from and write to EEPROM of 4096 bytes on the smart card consistently.	Section 4.2.2
The smart card terminal must interface to a smartphone via a wireless interface with data rates of at least 9600 bps at a range of at least 2 metres.	The smart card terminal can interface with a smartphone via Bluetooth 2.0 with a data rate of 38400 bps at a range of up to 5 metres.	Section 4.2.3
The terminal and smart card will use an approved cryptographic algorithm as defined in the EMV 4.3 book 2 and ISO 9564-1 for PIN verification of PINs with length of four to six digits. This must operate on 8-bit microprocessor smart cards.	A 3DES algorithm was implemented on the developed smart card that encrypts and decrypts the entire EEPROM using a PIN of four to six digits long. The encryption algorithm was not implemented on the terminal as it wouldn't serve any purpose.	Section 4.2.4
The system must be battery powered and rechargeable. The battery charge must last for a period of at least 4 hours or 100 transactions.	The system used two 9 V rechargeable batteries to operate and could last for 100 transactions but only lasted for 2 hours and 58 minutes.	Section 4.2.5
Deliverables		
Smart card	A smart card was built using an ATmega1284P microcontroller with a built in 4096 byte EEPROM.	Section 3.4
Smart card firmware	A smart card firmware and software system was developed to manage the EEPROM memory, communicate with the smart card terminal and determine the cards balance.	Section 3.4.3
Smart card connector	This was not used in the final system since a smart card was built from first principle.	---
Smart card read/write interface	A smart card interface was developed from first principle, supplying the requirements to interface with the constructed smart card.	Section 3.5
Processing unit (Microcontroller, FPGA)	An ATmega1284P microcontroller was used for both the smart card terminal and smart card.	Section 3.1.1
Keypad and LCD for human interface	A keypad and LCD were bought off the shelf. The software for the devices to operate were developed into libraries from first principle for C.	Section 3.1.2 Section 3.1.3
Wireless interface (Bluetooth, Wi-Fi)	A RN-42 Bluetooth module with a SPF breakout board was used that interfaced using serial UART.	Section 3.1.4

Smart card terminal	A smart card terminal was designed and built from first principle using a microcontroller.	Section 3.1
Smart card terminal application, firmware, and interface code	Smart card terminal code used the microcontrollers built-in libraries or were developed from first principle. I	Section 3.1.5 Section 3.6
Security algorithm for PIN authentication	3DES simulation was developed in Python and a C library was developed to operate on an 8-bit microcontroller.	Section 3.3 Section 3.4.5
Smartphone device	The smartphone application used a Samsung S5 with Android 6.0.1.	---
Smartphone application (user interface & functional code)	An Android application was developed that used Bluetooth and allowed for payment requests to be sent to the terminal and stored all successful transaction history.	Section 3.2

Table 16
Summary of results achieved

4.2 QUALIFICATION TESTS

4.2.1 EXPERIMENT 1: SMART CARD INTERFACE CHARACTERISTICS

4.2.1.a Qualification test

Objectives of test:

The objective of this test was to verify the smart card interface met the electrical requirements to interface correctly with the smart card. These characteristics are power and the serial communication.

Equipment used:

- A RS232 to TTL converter,
- Tektronix TDS 1002,
- Topward Electrics Instruments Co. Dual Tracking DC Power Supply,
- Smart card,
- Smart card terminal,
- Desktop PC with a serial connection, and
- Putty software.

Experimental parameters and setup:

The smart card interface is connected to the DC power supply, setting the output voltage to 9 V and providing enough current to allow for operation. The RS232 to TTL converter is connected to the PC and Putty is loaded with the serial settings of the smart card. The converter is connected to the smart card interface. The smart card is connected to the interface and the interface to the terminal. The oscilloscope is connected to the interface voltage supply for a measurement and connected to the clock line for a measurement. Commands are sent from Putty to the smart card to test data communication.

4.2.1.b Results and observations

Measurements:

The smart card interface converted the input supply of 9 V to a usable 5.04 V and could operate the interface and smart card. This measurement is shown in Figure 40. The clock signal of the interface was created using the terminals PWM, producing a 1.587 MHz clock signal as shown in Figure 41.

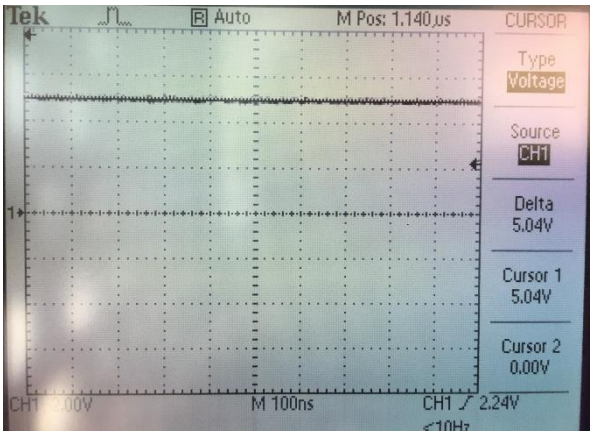

Figure 40
Voltage supply results

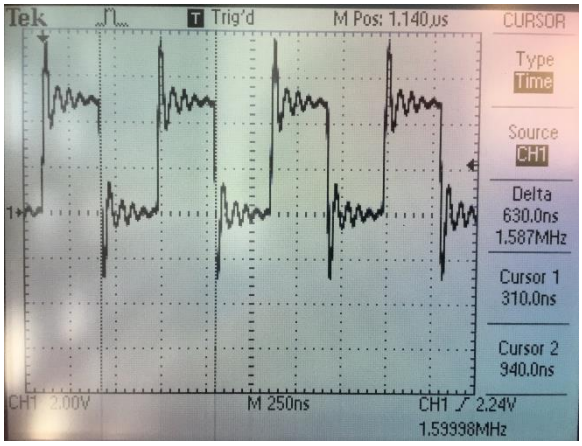

Figure 41
PWM clock signal results

After 1000 different of commands and strings where sent and received on both the terminal and smart card through the interface, producing 0 communication errors.

Description of results:

The interface met the requirements for its design, being able to supply the power for the smart card and interfacing the smart card terminal and smart allowing for working half duplex communication.

4.2.2 EXPERIMENT 2: SMART CARD EEPROM MEMORY READING AND WRITING

4.2.2.a Qualification test

Objectives of test:

The objective of this test was to verify the EEPROM reading and writing capabilities of the smart card device and that data can be stored for extended periods without power.

Equipment used:

- AVR JTAGICE MKII
- Smart card
- Topward Electrics Instruments Co. Dual Tracking DC Power Supply
- Desktop PC
- Python software

Experimental parameters and setup:

A random string consisting of lowercase, uppercase and numerical characters is generated using the Python script shown in Figure 42. The smart card is connected to the power supply's 5 V 3 A supply. The smart card is connected to the AVR programmer and the randomly generated string is written to the smart cards EEPROM. The smart card is turned off for a duration of 10 minutes. The power system is enabled again, and the data is read from the smart cards EEPROM and compared with the original string.

```
import random
import string

def RNG(size):
    values = string.ascii_lowercase + string.ascii_uppercase +
string.digits

    x = ''
    for y in range(size):
        x = x + (random.choice(values))

    return x

t = RNG(4096)
print(t)
print(":".join("{:02x}".format(ord(c)) for c in t))
```

Figure 42
Code snippet random generation

4.2.2.b Results and observations

Measurements:

This process was repeated 15 times with different 4096-byte strings producing the following results:

- Amount of write errors: 0
- Amount of read errors: 0

Description of results:

The system operated with no errors when reading and writing to the EEPROM on the smart card. The EEPROM does have a limited write cycle of 100,000 writes and would likely generate errors after this point.

4.2.3 EXPERIMENT 3: TESTING WIRELESS INTERFACE RANGE AND DATA RATE

4.2.3.a Qualification test

Objectives of test:

The focus of this test was to verify the data transfer rate and supported range of the Bluetooth device.

Equipment used:

- Smart card terminal,
- Smartphone device,
- Payment request application,
- Tektronix TDS 1002 Oscilloscope, and
- Tape measure.

Experimental parameters and setup:

The smart card terminal is turned on and placed on the work bench. A tape measure is taken and used to measure a 1 m, 2 m, 5 m, 10 m, and 20 m mark from the workbench. The smart card terminals receive signal is connected to the oscilloscope. The smartphone is taken to each measured point and is connected to the terminal via Bluetooth. Data is sent from the smartphone at each measured point and the data rate is measured on the oscilloscope and the integrity of the data is compared.

4.2.3.b Results and observations

Measurements:

After the smartphone has connected to the terminal a string of 4096 bytes is sent. This was repeated 5 times at each distance. The measurements of the data rates and data errors at different distances are shown in Table 17. An example of the data rate measurement for the character ‘A’ is shown Figure 43.

Distance	Data Rate	Errors
1 m	38460 bps	0
2 m	38460 bps	0
5 m	38460 bps	0
10 m	38460 bps	0
20 m	*Couldn’t Connect*	*Couldn’t Connect*

Table 17
Wireless interface data measurements

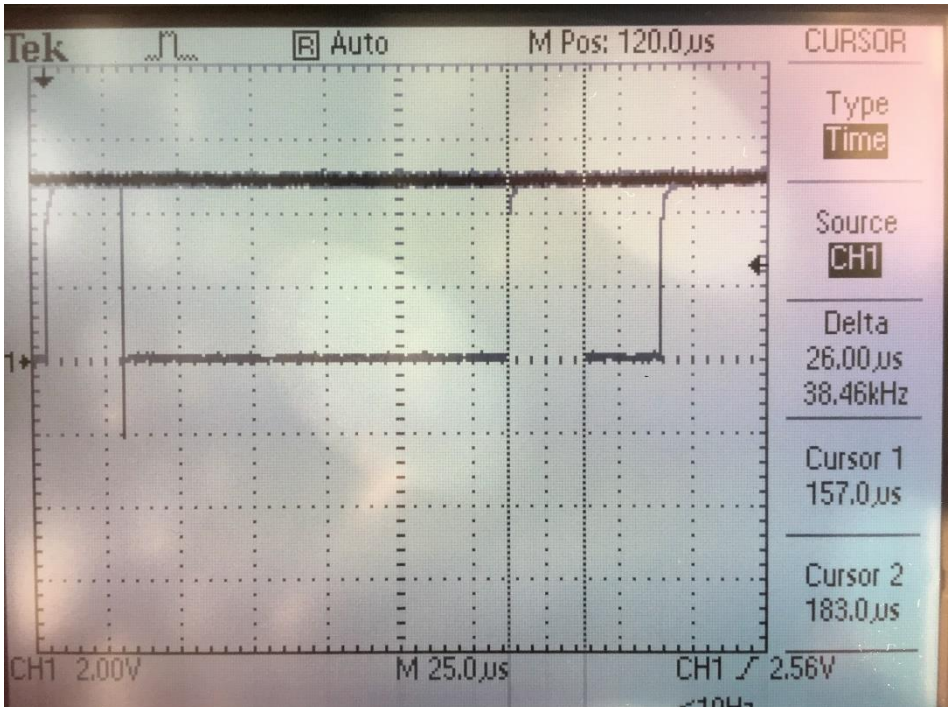

Figure 43
Data rate measurement on the character 'A'

Description of results:

The results are as expected for the Bluetooth module. The device supports data rates of up to 115200 bps and is a class 2 device with an operating range of 10 m. The device couldn't connect at a range of 20 m, so no data could be collected.

4.2.4 EXPERIMENT 4: 3DES VERIFICATION

4.2.4.a Qualification test

Objectives of test:

The point of this test was to verify that the 3DES algorithm was working correctly for encryption and decryption in all cases, with all reordering and S-Box table variables tested.

Equipment used:

- 3DES Python simulation
- Smart card
- Desktop PC with a serial connection
- Putty software
- A RS232 to TTL converter.

Experimental parameters and setup:

The Python simulation is setup and the script shown in Figure 44 is run. The script, written in C, is performed on the smart card to verify its 3DES library.

```
def DES_Verification():
    X0 = np.zeros(8, dtype=np.uint8)
    X16 = np.zeros(8, dtype=np.uint8)

    X0 = [0x94, 0x74, 0xb8, 0xe8, 0xc7, 0x3b, 0xca, 0x7d]

    X16 = [0x1b, 0x1a, 0x2d, 0xdb, 0x4c, 0x64, 0x24, 0x38]

    def ValidationTest():
        """
        Tests DES encryption & decryption
        using algorithm proposed by Ronald Rivest
        (http://people.csail.mit.edu/rivest/Destest.txt)
        """

        X = X0
        for i in range(16):
            y = key_schedule(x)
            if i % 2 == 0:
                    # if i is even, x[i+1] = E(x[i], x[i])
                x = encryptionDES(y, X)
            else:
                    # if i is odd, x[i+1] = D(x[i], x[i])
                X = decryptionDES(y, X)
        if X.all() == x16.all():
            print("DES Working")
        else:
            print("DES Not Working")

    ValidationTest()
```

Figure 44
Code snippet for DES algorithm verification

4.2.4.b Results and observations

Measurements:

The verification test passed perfectly.

Description of results:

This verification test is one proposed by [14] describes a way of verifying the system. First the amount of possible errors is determined by calculating the amount of errors due to reordering tables, circular left shifting, XOR calculations, and S-Box tables. This value is 36,568 possible errors. The problem is that when one error occurs it continues throughout the program and using the end result to determine the problem is extremely difficult. A test was set up to alternate between performing encryptions and decryptions at even and odd iterations respectively. These iterations use both the ciphertext result as the input message and key for the next operation. This is shown in Equation 5 and Equation 61.

$$X_{(i+1)} = E(X_i, X_i), \quad \text{if i is even} \tag{5}$$
$$X_{(i+1)} = D(X_i, X_i), \quad \text{if i is odd} \tag{6}$$

An initial value is picked with focus on testing the S-Box entries efficiently. This initial value is shown in Equation 7.

$$X_0 = 94\ 74\ B8\ E8\ C7\ 3B\ CA\ 7D, \quad (7)$$

Table 18 shows the breakdown of the entire process. It shows the iteration step and what the result should be and how many possible errors have not been checked. The entire table doesn't need to be checked for each iteration for full verification, if the last generated result is the same as the value in the table, the system is verified.

i	X_i	No. of errors not detected
0	94 74 B8 E8 C7 3B CA 7D	36,568
1	8D A7 44 E0 C9 4E 5E 17	14,170
2	0C DB 25 E3 BA 3C 6D 79	4,842
3	47 84 C4 BA 50 06 08 1F	2,866
4	1C F1 FC 12 6F 2E F8 42	1,550
5	E4 BE 25 00 42 09 8D 13	996
6	7B FC 5D C6 AD B5 79 7C	652
7	1A B3 B4 D8 20 82 FB 28	458
8	C1 57 6A 14 DE 70 70 97	274
9	73 9B 68 CD 2E 26 78 2A	180
10	2A 59 F0 C4 64 50 6E DB	126
11	A5 C3 9D 42 51 F0 A8 1E	94
12	72 39 AC 9A 61 07 DD B1	72
13	07 0C AC 85 90 24 12 33	52
14	78 F8 7B 6E 3D FE CF 61	20
15	95 EC 25 78 C2 C4 33 F0	4
16	1B 1A 2D DB 4C 64 24 38	0

Table 18
DES verification table

4.2.5 EXPERIMENT 5: TESTING BATTERY LIFE

4.2.5.a Qualification test

Objectives of test:

The objective of this test is to verify that the battery life of the device meets the requirements of 100 transactions or 4-hour lifespan.

Equipment used:

- Stopwatch
- Smart card terminal
- Smart card
- Smartphone and application

Experimental parameters and setup:

The entire system is setup in the general application with the two rechargeable 9 V batteries connected. The batteries are verified to be fully charged. The system is started, and a transaction

process is done and repeated for a total of 100 times. The batteries are then removed and charged to full. The batteries are then reconnected to the system and a stopwatch is started. The system is left to sit in a powered-on state until the system stops functioning correctly. The batteries are recharged one more time and the sleep features of the device are enabled. The stopwatch is used to measure the devices lifespan until the 4-hour mark.

4.2.5.b Results and observations

Measurements:

The system could successfully perform the required 100 transactions on a full charge.

Time until the battery charge dropped below functional use: 2 hours 58 minutes.

The sleep features allowed the device to remain operational after 4 hours of the batteries being plugged in.

Description of results:

The system could successfully perform the amount of required transactions. The system was only measured for the 100 transactions since a realistic simulation couldn't be made since power is used do to physical interactions with the device. The system couldn't meet the charge duration requirement of 4 hours. This measurement is with the sleep mode on the microcontrollers and Bluetooth module disabled. When the sleep features are enabled the device could last the required 4 hours. The main power draw is from the LCD and the Bluetooth module.

4.2.6 EXPERIMENT 6: FULL SYSTEM OPERATION TEST

4.2.6.a Qualification test

Objectives of test:

The focus of this test was to determine that all designed and implemented features operate in the final system as intended.

Equipment used:

- Smart card terminal
- Smart card
- Smartphone and application

Experimental parameters and setup:

The system is setup for general use. The terminal is powered on and the smart card is set ready for use and the smartphone application is launched and the Bluetooth devices are paired. All the system functions are tested in a systematic order 5 times each.

4.2.6.b Results and observations

Measurements:

All the system functions were tested and the details on the outcome of each test is detailed in Table 19.

Function	Result
Reboot Bluetooth module	100% PASS
Set transaction name and cost	100% PASS
Connect to terminal from application	80% PASS
Verify transaction details on terminal and accept charge	100% PASS
Successfully connect smart card to terminal	100% PASS
Enter PIN of length 4 – 6 digits long	100% PASS
Clear PIN for re-entry	100% PASS
Send PIN for verification on smart card	100% PASS
Smart card determines PIN correctness	100% PASS
Smart card verifies sufficient and insufficient balance	100% PASS
Terminal and smartphone receive correct transaction result	100% PASS
Application stores transaction details correctly	100% PASS
Application can successfully clear transaction history	100% PASS
Cancel transaction from application	100% PASS
Cancel transaction from terminal	100% PASS

Table 19
Functions verification table

Description of results:

These results help verify the functionality of the entire system and ensure it works as a full system. Many of the features of the system worked every time, except connecting to the terminal from the smartphone application. When requesting a payment, the Bluetooth connection is opened and waits for the module to accept the connection, sometimes the connection starts and is accepted by the Bluetooth module, but the smartphone application receives a connection denial, leaving the system in a stuck state where the Bluetooth module won't accept any more connections and needs to be rebooted.

5. Discussion

5.1 INTERPRETATION OF RESULTS

The overall system needed to provide a simple financial transaction system using a smart card for payment, a terminal for the smart card user to connect their smart card and enter their PIN. The terminal needed to interface with a smartphone application which a merchant would use to request payments and keep a transaction history of all their payments.

5.1.1 SMART CARD TERMINAL

The smart card terminal was an important aspect, serving as the middle-man for the entire system. Many aspects of the smart card terminal are to ensure that certain functionality was available and functioned correctly. The terminal included some of the smaller subsystems, such as the smart card interface, wireless interface, and the HMI. The smart card terminal met all the technical requirements for the system.

The smart card interface met the requirements perfectly, providing the required voltage and current to allow for the smart card to operate. It successfully interfaced the communication over a single data line in half-duplex by merging the UART data lines from the microcontroller using a circuit. The clock signal of 1.578 MHz was generated by the microcontroller, meeting the requirement range of 1 – 5 MHz. The device could generate a higher clock signal with small adjustments to the operating code, but this was not done since the clock signal was not required for successful interfacing with the smart card.

The Bluetooth module operated as desired but had a bug where the device would accept a Bluetooth connection, changing its state LED to connected, but the connecting device would receive a rejected connection and would not connect. This left the Bluetooth module in a stuck state, which required the current connection to be killed to allow for any other connections. When a failed connection like this occurred and the connection killed it rejected any further connection attempts from the device when this connection failure occurred, this could only be fixed with a device restart.

The human-machine interface met its design requirements allowing for user interaction through keypad buttons and allowed for entry of a user PIN. The LCD provided limited information only supporting 32 characters but could relay enough information for successful operation.

The battery life of the device met the transactions requirements but did not meet the duration requirement when left completely on. The high power consuming devices were specifically the LCD and Bluetooth module. The microcontroller and Bluetooth module support sleep modes, where the devices turn off to save power, which did save enough power to operate for the 4-hour requirement. The LCD was constantly drawing power and didn’t support any sleep functionality so building a power switch system to the LCD controllable from the microcontroller would do a lot to reduce power consumption.

5.1.2 SMARTPHONE APPLICATION

The smartphone application operated as expected. It supported the features and requirements, specifically:

- Connect and communicate with the terminal wirelessly,
- Allow for user submitted transaction names and cost,
- Keep a transaction history that can be cleared, and
- Cancel transactions.

The wireless functionality requirements that needed to be met is a data transfer rate of 9600 bps and a range of at least 2 metres away. The device operates at 38400 bps, but the Bluetooth module and smartphone Bluetooth support much greater data speeds. The terminal operating speed was the bottleneck for the speed of the wireless system, when higher data rates were used the errors on the terminal side increased and couldn't operate correctly. The device operating met the requirement of 2 metre operation distance, supporting up to 10 metres. The device operated consistently at 10 metres, but at distances above 10 metres, the ability for the smartphone to connect consistently to the terminal was reduced.

5.1.3 SMART CARD

The smart card device operated correctly, supporting the two major functions, 3DES encryption/decryption and successful EEPROM reading and writing. The smart card device utilised a powerful microcontroller, more than would be built into a smart card device. This was required since the encryption algorithm requires a high amount of flash memory for operation. Off the shelf smart cards perform encryption systems using hardware implementations.

The EEPROM system was built into the microcontroller and met the requirements of 512 bytes, supporting 4096 bytes of memory. The device could successfully read and write to the EEPROM consistently and worked well with the encryption system, since it operates in 8-byte blocks of data, allowing data to be read and written in blocks.

The encryption system operated successfully on the device, but the smart card ended up using a 16 MHz external clock. This external crystal was used since the internal crystal in the microcontroller was unstable and the UART module would produce a high amount of errors in reading and writing, it was also to reduce the time for the encryption system to operate. The external crystal is higher than in typical smart cards (1 – 10 MHz) but this could be reduced in code but would only serve the purpose of being more authentic.

5.2 ASPECTS TO BE IMPROVED

5.2.1 SMART CARD TERMINAL

The smart card terminal could be improved in a few ways. The choice of processing platform could be improved, using a higher bit device with a higher operating frequency would remove the restrictions the device had when interfacing with the Bluetooth module and other devices in the system. The use of a single chip computer would meet those performance requirements whilst providing a better software platform to develop on being able to provide more features for the terminal device. Using a better battery option would improve the systems longevity since 9 V batteries do not carry much charge.

5.2.2 SMARTPHONE APPLICATION

The smartphone application has room for some code improvements in terms of functionality and possible cross-platform support. The device software could also be built to allow for multiple connections to different terminals, so multiple transactions can be controlled from a single device. The GUI could do with some design improvements for usability reasons and for an improved user experience.

5.2.3 SMART CARD

Smart cards make use of microcontrollers which is mirrored in this system, but since a software implementation for encryption was done the use of an FPGA for the smart card is the better choice. DES is an encryption system designed as a hardware encryption and an FPGA would have been the better option to emulate this encryption in software. A software implementation of DES is more complex and very redundant, requiring a lot of code to operate correctly. Having code to emulate the hardware system, which an FPGA provides, would have reduced complexity and performed significantly better. The encryption only performs in an electronic code book method, this could be improved by using a block chaining method to improve the security of the encrypted data. Using a hardware engine for encryption would reduce the performance requirements of the system and would allow the system to be implemented on an off the shelf smart card.

5.3 STRONG POINTS

5.3.1 SMART CARD TERMINAL

There are some aspects of the smart card terminal did operate well and performed beyond
The smart card terminal keypad operates very well, exhibiting no erroneous behaviour after implementation of the debouncing system. The smart card interface, specifically the data circuit performed beyond expectations, allowing for the devices to interface in a half-duplex mode and whilst being able to communicate consistently with no errors. The interface between the smart card and terminal is at a data rate of 9600 bps which the circuit supports, but can support data rates much greater than the current one.

5.3.2 SMARTPHONE APPLICATION

The smartphone application's strongest point is the Bluetooth service, providing an asynchronous system that doesn't impact the performance of the phone application. The system could have been achieved using a polling system, but this would have reduced the usability of the app. The service would also allow the app to connect to multiple terminals, allowing for multiple transactions to be managed by the same smartphone.

5.4 SYSTEM FAIL CONDITIONS

The systems payment process is very linear and when failures occur in code the process starts again, or the devices need to be rebooted. The system has security components so if any knowledge into the encryption/decryption process, such as knowing certain code words could lead to system vulnerabilities. Brute force hacking will be ineffective for 3DES but getting knowledge in other ways is not out the question. The system is also vulnerable to any Bluetooth

vulnerabilities that do arise, but this is limited since no sensitive information is passed over the Bluetooth connection.

5.5 Design Ergonomics

The terminal was designed to be more of a desk object then a handheld one, but would operate better as a handheld device. The panel layout for the LCD and keypad are intuitive, mimicking a calculator layout, though the keypad keys aren't as clear. The keypad keys could be improved by replacing the letter keys with colour keys to mimic other financial machines. The smartphone application could be improved in mainly an aesthetic way, but the usability of the functions of the application a very clear and accessible. The smart card could be reduced in physical size and ideally would be the actual dimensions of a smart card, but this is not something within the realm of possibility for this system.

5.6 Health and Safety aspects of the design

The major health and safety concern is the typical electrical device concerns, but most of these are alleviated since enclosures are used on both the smart card terminal and smart card. The major concern is making sure the batteries are removed and managed correctly, following typical safety behaviour for batteries.

5.7 Social and legal impact and benefits of the design

The legal implications are that it needs to meet consumer electronic standards if it were to be produced as a commercial product. Most of the other legal implications are already met and been built into the Bluetooth module. The system is built to emulate a financial transaction so if the system were to be implemented under real conditions any sort of regulations or requirements would also need to be investigated and implemented. The social benefits are that if the system were implemented it would allow businesses to accept payments using the terminal device and just their smartphone, this would help reduce the start-up costs of small business and allow for businesses to operate in pop up fashion being able to accept payments anywhere.

5.8 Environmental impact and benefits of the design

The environmental issues of the system are around the disposal process and the wireless interface. The device uses a wireless interface which does contribute to electromagnetic noise in the environment, though with current research doesn't pose any sort of major impact. The device was built using components which contain lead and was soldered with a lead based solder. This lead factor and that the device uses batteries is a concern that the device is disposed of correctly since lead and batteries are hazardous. The device could be recycled at typical electronics recycling facilities and those components reused. The plastic enclosures are not environmentally friendly since disposing of plastic involves processes that are not environmentally friendly, though they could be recycled.

6. Conclusion

6.1 SUMMARY OF THE WORK

The focus of this project was to build a smart card system that could replicate a financial system whilst having the smart card terminal interface with a smartphone device. The system could be broken down into three sections. The systems are the smart card, smart card terminal, and smartphone application.

The work done for the implementation of this project is listed below.

- An investigation on smart cards and smart card standards and protocols.
- Design and construction of a terminal device that can provide informative and usable HMI features, and wirelessly interface with a smartphone device.
- C code for the firmware and software of the terminal was developed from first principle.
- Implementation of an LCD and keypad for the terminal device with code built from first principle.
- Design from first principle and construction of a smart card interface, providing the electrical signals and communication system for smart card operation and successful data communication.
- Investigation on smartphone development of an Android application that can interface wirelessly with an external hardware via Bluetooth.
- An Android application was built to request payments, whilst keeping a transaction history of all the successful payments. The application was built with a GUI developed in Android Studio.
- An investigation and a Python implementation of 3DES encryption for simulation purposes.
- Development and implementation of 3DES encryption that can operate on an 8-bit microcontroller device.
- Design and construction of a smart card device from first principle with firmware written in C that was designed and built from first principle.

6.2 SUMMARY OF OBSERVATIONS AND FINDINGS

The final system met all requirements except the power requirements. The system could perform all the designed features and performed with minimal problems.

A summary of the results is listed below.

- The smart card interface could correctly power and facilitate communication with a smart card.
- The smart card could consistently read and write 4096 bytes of EEPROM memory.
- The terminal could interface with a smartphone application with data rates of 38400 bps at a range of up to 10 metres.
- The 8-bit smart card could encrypt and decrypt the EEPROM using 3DES with a PIN of 4-6 digits.
- The device could perform 100 transactions on a full battery charge but couldn't last the required 4 hours, only lasting 2 hours and 58 mins. That was with all sleep features disabled, but can last the 4 hours required when sleep is enabled.

6.3 SUGGESTIONS FOR FUTURE WORK

Future work for this exact project and system is limited in extended scope. The focus of future projects should be on different aspects of the smart card system. A project could focus on using an off the shelf smart card and develop an operating system for the smart card supporting a variety of different features. A file management system could be developed and an application installation and control system. Developing an operating system that allows for a variety of different application software to be installed and run on the same smart card would allow for a single smart card to be used for many tasks instead of having a card for each application. Some example applications could be developed to demonstrate this functionality and explore the use cases for multi-application smart cards.

7. References

[1] W. Rankl and W. Effing, Smart card handbook, Third ed. John Wiley & Sons, 2004.

[2] T. M. Jurgensen and S. B. Guthery, Smart cards: the developer's toolkit. Prentice Hall Professional, 2002.

[3] W. Rankl, Smart Card Applications: Design models for using and programming smart cards. John Wiley & Sons, 2007.

[4] International Organization for Standardization, "Identification cards -- Integrated circuit cards -- Part 3: Cards with contacts -- Electrical interface and transmission protocols", *International Organization for Standardization*, Nov. 2006. [Online]. Available: https://www.iso.org/standard/38770.html [Accessed: Aug. 28, 2017].

[5] EMVCo, "Integrated Circuit Card Specifications for Payment Systems Book 2 Security Key Management", *EMVCo*, Version 4.3, Nov. 2011. [Online]. Available: https://www.emvco.com/wp-content/uploads/2017/05/EMV_v4.3_Book_2_Security_and_Key_Management_20120607061923900.pdf [Accessed: Aug. 8, 2017].

[6] G. Singh and Supriya, "A study of Encryption Algorithms (RSA, DES, 3DES and AES) for information security," *International Journal of Computer Applications (0975 – 8887)*, vol. 67, no. 19, April 2013. [Online]. Available: https://pdfs.semanticscholar.org/187d/26258dc57d794ce4badb094e64cf8d3f7d88.pdf [Accessed: Jul. 15, 2017]

[7] K. Mayes and K. Markantonakis, Smart cards, tokens, security and applications. Springer Science & Business Media, 2007.

[8] Bluetooth. (2017) Bluetooth how it works. [Online]. Available: https://www.bluetooth.com/what-is-bluetooth-technology/how-it-works. [Accessed: Jul. 23, 2017]

[9] Wi-Fi Alliance. (2017) Discover Wi-Fi Specifications. [Accessed: Jul. 23, 2017]. [Online]. Available: http://www.wi-fi.org/discover-wi-fi/specifications

[10] Verifone, Payware Mobile e105, 2013. [Online]. Available: http://www.verifone.co.za/media/4215671/pwm-e105-ds-a4.pdf

[11] Verifone, VERIFONE e255 FLEXIBLE MOBILITY, 2015. [Online]. Available: http://www.verifone.co.za/media/2877397/pwme255-ds-ltr.pdf [Accessed: Jun. 15, 2017]

[12] PayPal. (2017) Paypal here: Credit card reader | point of sale and mobile credit card processing. [Online]. Available: https://www.paypal.com/us/webapps/mpp/credit-card-reader [Accessed Apr. 2, 2017]

[13] Microchip, "8-bit AVR microcontroller", ATmega1284P datasheet, Oct. 2016.

[14] Ronald L. Rivest, "Testing Implementations of DES", *csial.mit.edu*, Feb. 1985. [Online]. Available: http://people.csail.mit.edu/rivest/Destest.txt. [Accessed Sept. 15, 2017].

Part 5. Technical documentation

This main report is supplemented with technical documentation. This provides more detail on the software that was used in the experiments, including program listings, a user guide and circuit designs. This section appears on the CD that accompanies this report.

The CD is organized into the directories listed below.

Main report
Part 5: Technical documentation
Software
References
Datasheets
Author
Circuit Diagrams
Circuit Photos
User Manual