Point and Pay
Final Report

December 12, 2005

18-551

Group 5
Kevin Borries (kborries@cmu.edu)
Joohoon Lee (jool@andrew.cmu.edu)
Jun Han (junhan@cmu.edu)

Background

Credit cards have been widely used for over 20 years, because they are more convenient than checks and more secure than cash. However, credit cards have limited security measures and cannot display important information like account balance. With modern handheld technology like PDAs and cell phones, information rich and secure financial transactions can take place.

Financial transactions have been moving in the direction of being more convenient and secure. Credit cars are more convenient than cash, but they have security issues. Credit cards require approval at the point of sale, but a stolen credit car can be used to make fraudulent purchases before the card is reported as stolen. A thief can also get your credit card number and security ID on the card just by looking at the card. Debit cards add better security by requiring a password before authorization, but they also have sensitive account information displayed on the card.

Infrared data transfer is the popular choice for handheld financial transactions, because it is cheap to implement and includes a standard (IrFM) that incorporates existing technology.[1] Infrared has a short transmit range, less than 3 feet, so the user will need to be close to the receiver. This short range can improve security, but may be a disadvantage in some applications like a drive-thru. Infrared does not have the interference problem that is present in RF systems, and consumes less power than the RF wireless communication.

Infrared for financial transactions has been established in South Korea by SK Telecom. SKT calls its product Moneta which successfully replaced the traditional credit card payment system.[2] Moneta turns your cell phones into a convenient and secure financial transaction system. Other telecommunication companies also started similar service, and all the companies are currently in process of deciding the standard that can

[1] Gordon-Lathrop, Sue. "Proximity_Payments – The Quest for Revolution in Electronic Payments", 2003 http://corporate.visa.com/md/dl/documents/downloads/Proximity_Payments.pdf
[2] "Moneta Card, SK Telecom" - www.monetacard.co.kr

be used by any mobile providers. Once the standard is established, it will be easier to make use of this new system.

Project Overview

The goal of this project is to use two C67 EVMs to drive a wireless infrared link modeled for financial transactions. The project is broken into three main parts:

1) Data reception and transmission with error correction coding
2) Link/Protocol control
3) Encryption and decryption

The first part takes care of transferring data between the two EVMs. The second part interprets what the received data means, and controls when the EVMs transmit. The third part adds a layer of security to the transmissions.

The inspiration for this project comes from the IrFM standard, but there are some key differences. First, a simpler link/protocol layer was implemented. Secondly, error correction was added, where it is not included in the standard. Third, a stronger encryption method is used here than with the standard.

Data Reception and Transmission

Hardware

The purpose of the hardware is to convert transmitted data from the EVM to infrared, and back again. The original hardware flow diagram is shown in Figure 1. The original design includs three main components: the amplifier with the voltage converter, the MCP2120 SIR encoder, and the TFDU4140 infrared transceiver.

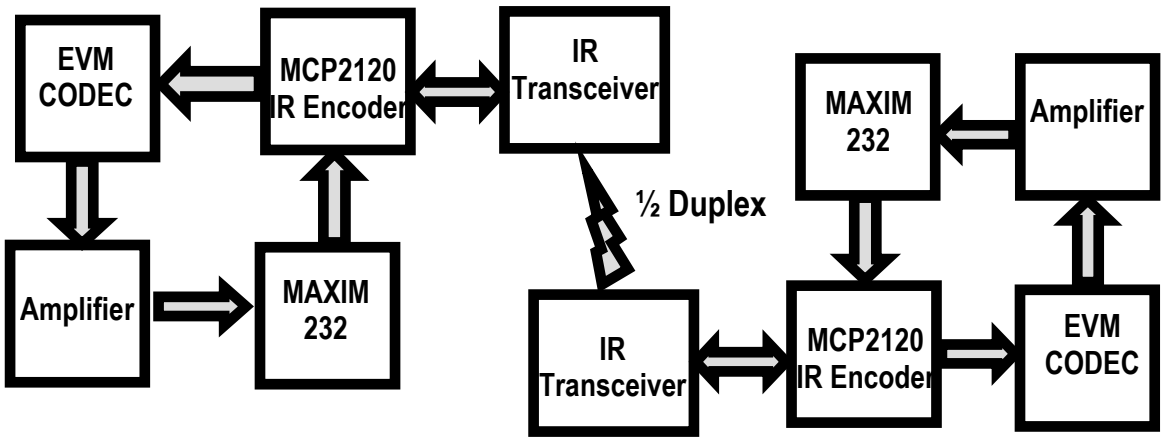


Figure 1. The original hardware flowgraph.

The codec to the EVM produces polar non-return to zero (NRZ) square waves that represent the binary data. This signal is amplified with an op-amp and is converted to CMOS levels (0-5 V) with a MAX232 voltage level converter. This CMOS signal is used by the MCP2120 to encode the data to the SIR standard. In the latter stages of the project,

the encoder would not properly encode the signals from the EVM. The encoder does work properly with signals though the UART, but not with the amplified signals from the EVM. Therefore, the second hardware design used the signals from the UART of the PC to transmit the data.

The SIR encoder converts the NRZ signal to a short return to zero pulse, Figure 2. The transceiver emits an infrared signal when the input is high and is off when the input is low. This form of modulation is called on-off keying (OOK). Short pulses are beneficial to an infrared system because it consumes minimal power. Most infrared transceivers work specifically with the SIR standard, so the encoder chip is necessary to convert the NRZ waveform to short pulses.

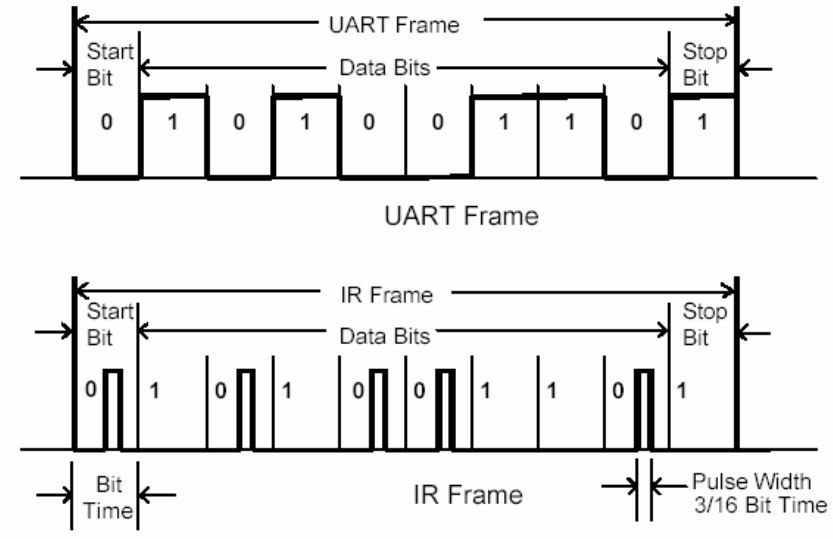


Figure 2. An example of a UART frame and a corresponding SIR frame (Picture from ref. 3).

The slowest bit rate that can be used by the transceiver is 9600 bits/s. The SIR standard calls for a pulse of 3/16 the original bit length[3]. Therefore, the maximum pulse duration used by the transceiver is (3/16/9600 = 19.5 μs). To produce this short of a pulse a frequency of 51.2 kHz is needed. The codec on the EVM has a maximum sampling frequency of 48 ksps, so the maximum frequency that the codec can construct is well below the 51.2 kHz required. Therefore, the codec cannot produce the pulses necessary to drive the transceiver without the encoder chip.

The receive side used the same basic setup with some minor changes. The transceiver is very sensitive, and will pick up on noise when no signal is being transmitted. This noise will confuse the SIR encoder/decoder into thinking that there is an incoming signal at all times. This encoder will not encode the message properly because it is interrupted to try to decode the incoming noise. This problem is solved by adding a latch on the transmitted signal to turn off the input while transmitting. The latch is implemented with a 74LS139 demux chip. Basically, this chip sends out the inverse of the input signal when the control line is low, and a constant high signal is sent when the

[3] Infrared Data Association. IrDA Physical Layer Specification v1.4**,** May 30 2001, www.irda.org

control line is high. This chip is chosen because the received signal needs to be inverted anyway.

The control line should be high when transmitting and low at every other time. The original design uses a diode and an RC filter to convert the amplified input signal to a constant 5V signal. However, the second version of the hardware uses the UART to transmit. The UART includes a data-terminal-ready (DTR) line which is used to control the transmit latch.

Eventually the encoder/decoder chips all went bad, possibly because of leaks in the breadboard, or clock fluctuations. With only one encoder/decoder chip, the full infrared system cannot be implemented. Therefore, the infrared link is currently simulated by connecting the transmitted signals from the MAX232 directly to the codec. The noise is very low in the system so this simulated channel is an accurate representation of the actual infrared system.

Receiver Program

The purpose of the receiver program is to decipher what bits have been sent to the codec. There are three main parts to the program, the filter the correlator and the decimator. The filter removes all of the unnecessary parts of the signal and prepares the signal for the correlator. The correlator uses the filter data to find when to start receiving data. Lastly the decimator resamples the data and converts the data into binary.

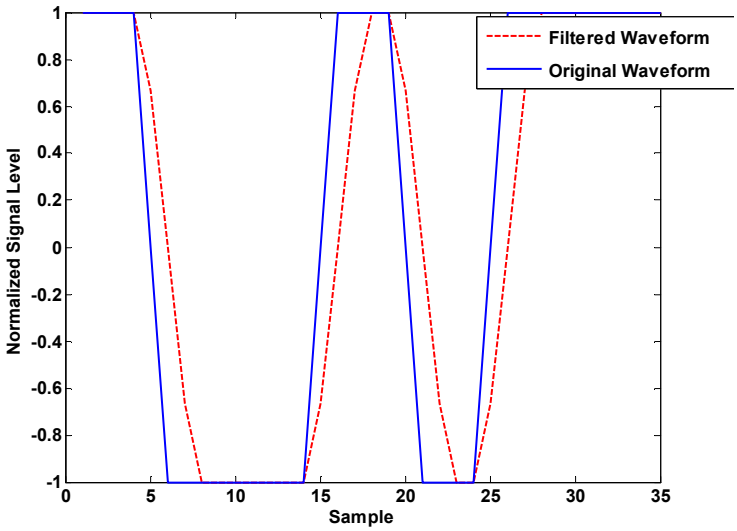


Figure 3. Filtered and un-filtered square waves. Notice the peak every five samples that can be found through correlation.

The sampling rate for the codec is set to 48ksps, so with at a data rate of 9600bits/s, there are 5 samples per bit. The filter chosen is a three sample averaging filter. Only three samples are used so that samples that occur on the transition between bits will be ignored. The speed of this filter is important because it is recalculated for every sample (20.8 μs). Speed is optimized with this filter because multiplies are not needed. The input signal will look like square waves. By averaging each point with the last two samples, the edges of the squares will not be as sharp, Figure 3. The peaks in these values will help the correlator find the optimum point to start sampling the signal. Because the

transition samples are ignored, the peak values will be consistent, making the correlator threshold more dependable.

The next step is to correlate the signal with the start bytes. As mentioned in the hardware section, when no data is being transmitted the input signal appears to be noise. A correlation sequence of 16 bits is needed to detect the start sequence of transmitted data and ignore the noise. The noise usually manifests itself as a low bit and 7 high bits at 9600 bits/s, because the encoder chip filters out all other frequencies outside of the baud rate. There is also some other random distortion coming from the clock and other cross-talk in the board which is unpredictable noise. The bit start sequence is chosen to be '10000110' repeated. There are enough zeros to be different from the noise, and there is also a '1010' section to produce peaks to help with timing.

The correlation is performed after every sample is read and filtered, so like the filter, speed must be optimized. The correlator only looks at every last fifth sample. These previous samples are either added or subtracted together depending on what the start sequence would be at that location. The current bit is correlated with a zero so it will be subtracted. The sample from five samples ago is added because it is correlated with a one. This sequence is repeated for 8 bits (8*5=40samples). If the correlation sum is above a threshold the process is repeated for the next 8 bits. If that correlation sum is above the threshold, than the program begins reading in the signal as transmitted data. This scheme of only reading 8 samples and using every fifth bit reduces the number of cycles used to calculate the correlation.

The decimation is done by a similar method to the correlation. After 40 samples are gathered into the input buffer, the program finds the best place to sample the data. A sum of the absolute value of every fifth sample is taken. This sum is compared with other sums for a delay of 0-4 samples. The delay with the highest sum is used to decimate the samples. The samples are converted to bits by interpreting a positive value as 1 and a negative value as 0. This delay synchronization is necessary because there is some drift between the clocks in the transmitter and receiver.

The delay is also used to shift the index of the received data buffer to prevent data overlap. The optimal delay is 2 because a shift of two samples in either direction can be read correctly, without missing bits or reading bits twice. If the delay value is greater than two, then the index is help up for a sample or two. If the delay is too low then the index is moved up, and the missing spots are replaced by the previous samples.

There are a couple of minor problems with the codec that were solved before implementing the receiver program. The input signal swings between 0-5 V. The codec is able to filter out the DC component, but not right away. This bias causes the peak-to-peak amplitude to be low, because the signal will reach the upper limit of ADC. Therefore a bias parameter is added to the receiver to zero out the DC component. Also, 50 junk bytes are sent before the actual data is sent to help "warm up" the codec.

Error Correction Coding

There are two main fields of error correction codes, block and convolutional codes. Both convolutional codes and block codes can give about the same performance[4]. Convolutional codes were chosen for this project because they appear to be faster,

[4] Rappaport, Theodore S. "Wireless Communications Principles and Practice 2nd ed.". 2002 Prentice Hall, Upper Saddle River, NJ.

because of the continuous nature of the code. The algorithms for coding and decoding are relatively simple for the lower order convolutional codes.

The strength of coding needed is dependant on the bit error rate in the system. The bit error rate was tested by sending a known sequence of ASCII characters through the infrared system. Each line of 70 characters was visually checked against the data sent. In close range, no errors were detected. When the transceivers were more than 40cm apart or 20 degrees off axis, the bit error rate grew to about once per line, or 0.0015. The bit error rate grew very quickly in until the signal was unnoticeable at 42cm.

Error correction will be able to extend the range of the device. If each transaction sends 500 bytes of data, then for 99% successful transactions at 42cm, a bit error rate of (.01/500) = 2e-5. Simulink simulations were performed by forcing random bit errors and the basic convolutional encoder was able to improve the bit error rate from 0.0015 to less than 1e-7.

The convolutional encoder chosen is shown in Figure 4. The output of the convolutional encoder follows the state chart in Figure 5. This convolutional encoder has a constraint length of three, because three input values are used to calculate the output. The rate of this code is ½, because there are two coded values for every input value[5].

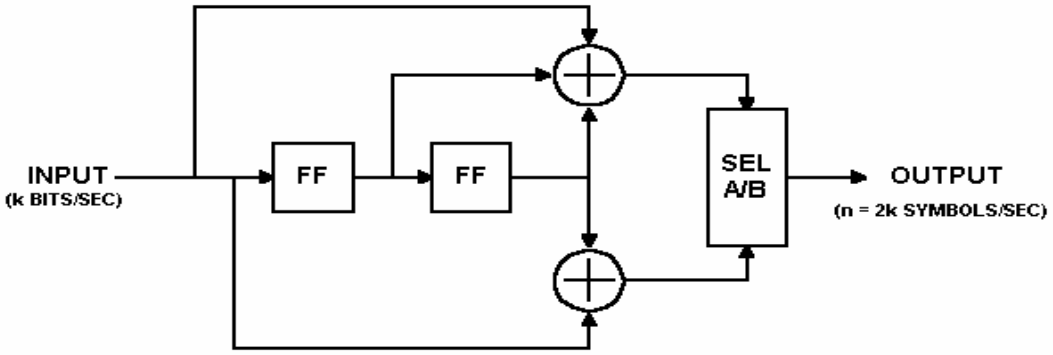


Figure 4. Graphical representation of the convolutional encoder. FF is a delay by one and the cross is an XOR. The output is in the form ABABAB...(picture from ref. 5)

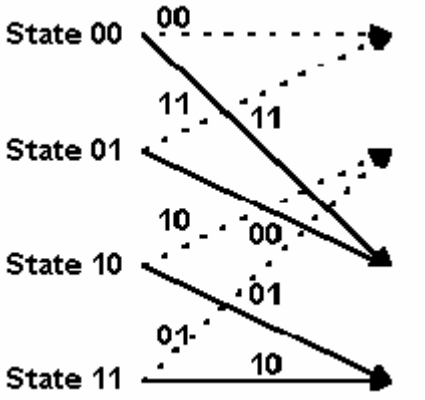


Figure 5: State diagram for convolutional encoder. Dashed lines represent a 0 input and solid is a 1 input. The coded values are marked along the lines. (diagram from ref. 6)

[5] http://home.netcom.com/~chip.f/viterbi/algrthms.html

The convolutional codes are decoded using the Viterbi decoder. The Viterbi decoder builds three matrices. The first matrix to be built is referred to as the state metric[6]. The state metric contains the number of errors needed to be at a particular state for each pair of input bits. For example, assume the coded data is '10.' The state metric for the previous input bits is given as state00=1 and state01=2. There are two ways to get to state 00, from state 01 and state 00. The data needed to take those paths is 00 and 11 respectively. Each path would give one error, so the path with the minimum error is from state 00, which has the minimum starting error. The resulting error metric for state 00 would be 1+1=2. This procedure is repeated for all states and all data pairs.

The second matrix, referred to as the survival matrix, keeps track of the previous state which gave the current state the lowest number of bit errors. In the previous example, the survival matrix would contain 0 for state 00, because the path with the minimum error came from the 0^{th} state. The third matrix goes backwards through the state metric and the survival matrix to build the path that has the least number of bit errors. This matrix has just one row and is called the traceback matrix. Knowing the states for each pair of inputs, the original data can be built.

The Viterbi decoder was implemented with C code found on the internet. This code was only written for a specific code length, and is not practical for long strings of data. Therefore this code is adjusted and optimized for this project. To build an accurate decoder, a traceback length of 15 is required[7]. The survival path and state metric of the next 16 bit pairs are used to decode the first pair. Noting that the state metric and survival metric only depend on the current and previous inputs, these matrices do not change when more data is added to the end. Therefore, the state metric and survival matrix can be saved after decoding each bit pair and used for the next pair of bits. When the next pair of bits are decoded, the state metric and survival matrix are just shifted and the new values are added to the end of the matrices. The traceback array is recalculated for every pair of bits, but that is a minor part of the code in terms of number of cycles.

Link/Protocol Layer

Data Flow Graph

The figure below shows our flow graph that was modified from our project proposal. The main difference is that the transmitter does not send the data from EVM through IR sensors, but rather from PC to IR. The receiver side receives through IR and through EVM. Let us suppose a situation where a transaction occurs by a transfer that is initiated by PTD to a POS. PTD will be a transmitter in this case, hence POS will be a receiver. The user will input a command, through GUI in our demo implemented in Java. Then the Java GUI will call on the PC side Java code, which will then call a PC side C code through the use of Java Network Interface (JNI). Then the PC side C code and EVM C code will talk to each other using the HPI connection. The transfer of data will occur from PC side Java through UART communication which sends data through to IR sensor and transmits to the IR sensor in the receiver side. Similarly, when POS needs to send

[6] http://home.netcom.com/~chip.f/viterbi/algrthms2.html
[7] http://home.netcom.com/~chip.f/viterbi/algrthms2.html

data to PTD, the same procedure will just reverse, making POS a transmitter and the PTD a receiver.

Link/Protocol Details

Our model uses the IrFM™ (Infrared Financial Messaging) protocol when the devices form a synchronized link. We considered the following two devices, POS (Point-of-Sale) for market vendors and PTD (Personal Trusted Device)[8] for customers' handheld devices. The scenario we considered is that a customer walks up to the POS device when purchasing a product. The POS is linked to the existing financial database (i.e.: Visa).

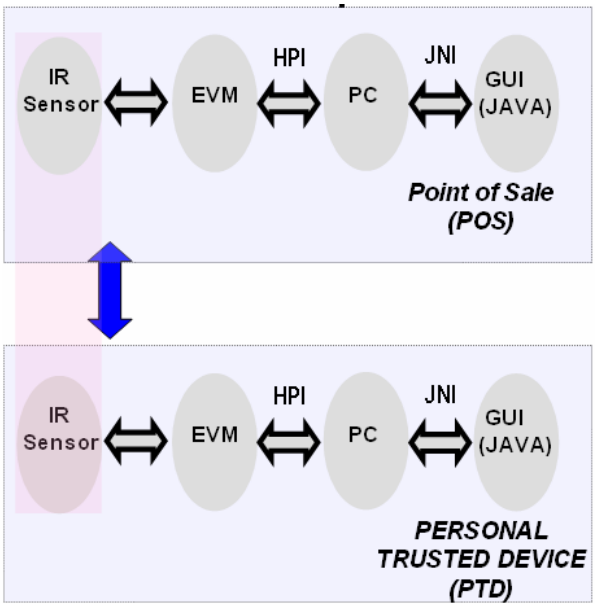


Figure 6 *Old Proposal "Point and Pay" flow graph*

Our current model of implementing this IR transaction has been modified from our initial proposal in certain extent. The above figure is our initial flow graph used in the proposal stage of this project. For example, we did not follow too strictly on the IrDA standard's connection modes. In the standards, the system first goes into a Normal Disconnect Mode (Refer to the diagram below). This mode requires the PTD to start polling for a response by the POS device.[9] As soon as the POS acknowledges back to the PTD with a response, the system enters the Normal Disconnect Mode. In order to allow the PTD to recognize all the devices in its range, it sends out multiple packets numbered uniquely (usually around 6 – 8 packets). Depending on which packet the POS has responded to, the corresponding POS will be assigned the corresponding packet number, so that the PTD can uniquely identify each POS devices in its range. This will allow the customer to correctly identify his/her vendor and prevent any fraud. If the correct vendor POS did not reply back due to packet collisions/loss, the user will let the PTD send out

[8] "Point and Pay Profile" Infrared Data Association(IrDA®), Version 1.0, December 9, 2002
[9] "Microchip MCP2155, IrDA® Standard Protocol Stack Controller Supporting DCE Applications", © 2001 Microchip Technology Inc.

another 6-8 packets to see if the device can get the acknowledgement back from the POS the user wants to start the transaction with. Then the PTD will send out its XID without numbering this packet.

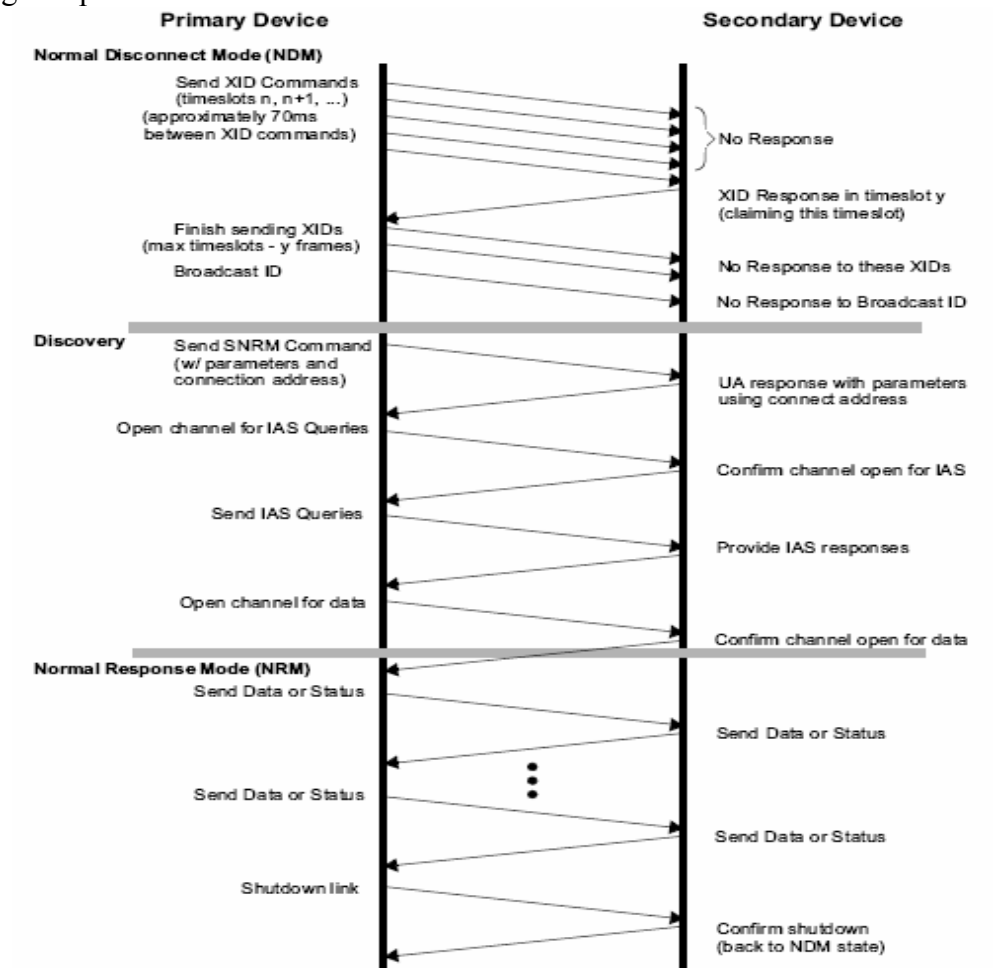


Figure 7 - *Microchip MCP2155 IrDA® Standard Protocol Stack Controller Supporting DCA Applications* [9]

However, in our current model of this project, we have ignored many of the unnecessary multiple POS problems because we assumed the customer is walking up to the vendor and pointing the PTD to the vendor POS. The user initiates a transaction by starting the GUI ran by the Java applications on the PTD. Then using the JNI (Java Network Interface), the PC side Java code calls on the PC side C code. This C code with the C code on the EVM enables the transfer of a byte long predefined RQPOS, the request POS Info from PC to EVM through HPI connection. Then this RQPOS request message gets encoded on the EVM. The encoded message then is sent back to the PC side C code through the HPI connection. After that the Java code calls on the C function to get the data received from the EVM to send it through the serial connection to the IR sensors to the EVM of the receiver. This has to be done in this manner because our serial transmit functionality has been implemented in Java.

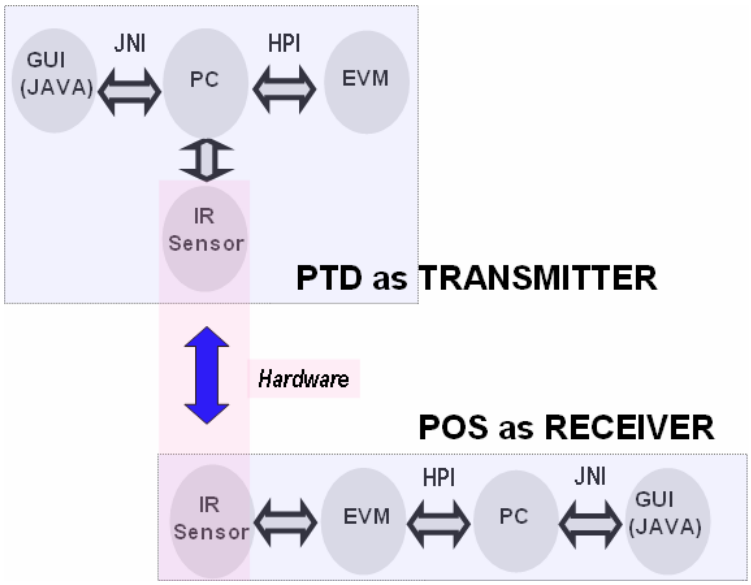


Figure 8 *Current Flow Graph*

On the receive side as soon as it gets a message for the first time from some PTD, it checks for the RQPOS. If the incoming message is the same as its predefined RQPOS message, the POS sends out its POS information, such as the vendor name, address and number. It does this by encoding the POS info into a packet which is a buffer of length 512 bytes. This buffer is made this long so that we can test transfers of long data at times. Then the packet is transferred over to the PC C code then to Java, through serial to IR sensor back to the EVM side of the PTD. However, I had to put some delays in between the receive and the send on the receiver side. This was mainly due to the fact that the system needed time to get into the proper states for the receiver.

In the proposal, we stated that before the system moves onto the Normal Connect Mode, which is the state in which the two devices communicate sensitive information, the PTD will send out its public key to the POS. Then the POS will send its public key information to the PTD with the encryption (using the public key received from the PTD). Then the PTD will use its internal private key to decrypt the message, to get the POS's public key. From this point onwards, the devices will keep encrypting their messages using other's public key, and decrypt the received messages using their own private keys. (Please read the report on the Application Layer for further encryption/decryption details.)
Using the same procedure as sending out the RQPOS info and sending back the POS info, the public key exchanged worked the exact same was for both the PTD and the POS.

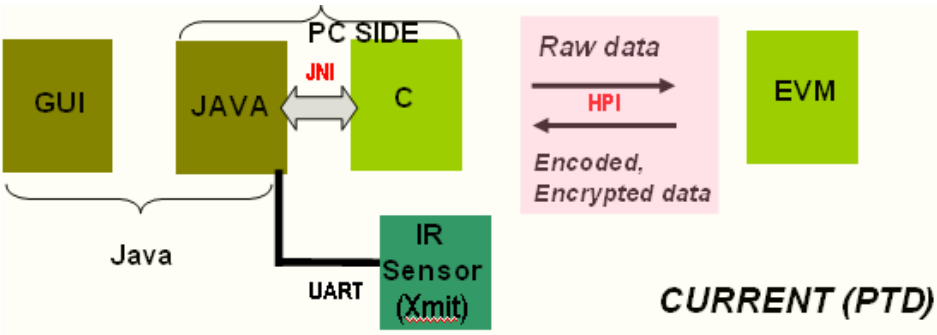


Fig 9 *Current PTD flow graph)*

In the proposal, we also stated that after exchanging the public keys, the system enters the Normal Connect Mode.[9] In this state, the two devices will communicate freely, but encrypted, to transmit and receive sensitive information such as the user ID (i.e.: Name, SSN, phone number), credit card number, total price purchased. Also, the user will be prompted for confirmation on the purchase items. When all the transactions are completed, the user will acknowledge a close link command to terminate the communication and transaction.

Again following the proposal tightly, with the exchanged public keys, the devices encrypt/decrypt messages using both the public and private keys. From this point onwards, sensitive data transfer is encrypted before transmitting. The PTD encrypts the credit card information transferred form the PC side onto the EVM. Then the encrypted data is encoded before being transferred back to the PC side. Then the encrypted and encoded data gets sent over the IR channel to the POS side. The POS will firstly decode the data. Then it will decrypt the message and send the raw data to the PC side of the POS.

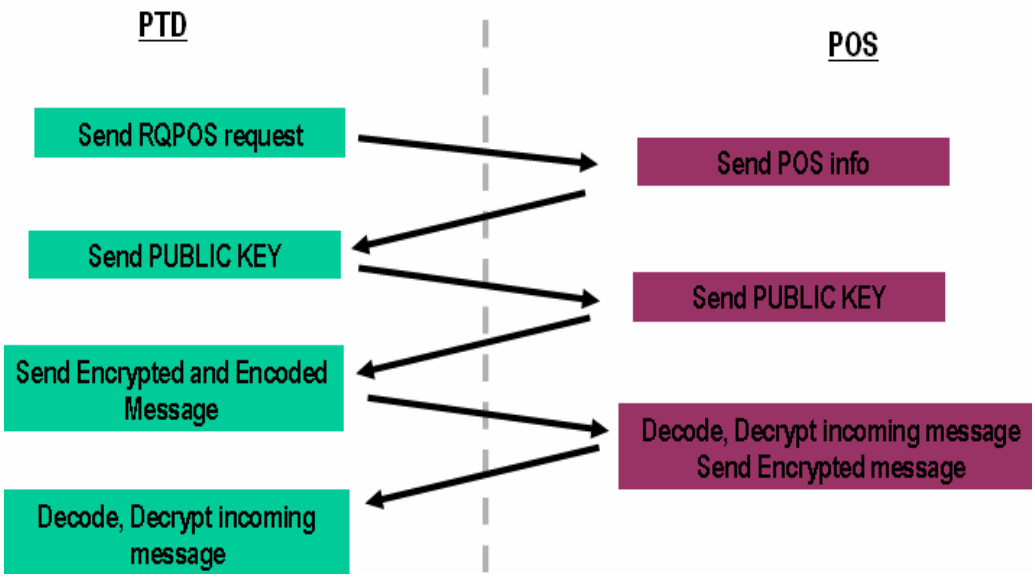


Fig 10 *Current Protocol Flow Graph*

One of the main problems we faced in implementing this project in this layer was that there was a problem of deadlock because we changed the flow graph from

EVM⇔EVM IR transfers to PC⇔EVM. This is because in the old implementation, where we had the EVM to EVM transfer through IR, the PC always waited for the EVM. Therefore, this did not cause any problem. However, in our new model, where we have multiple transfers going back and forth between the PC and EVM, since the data has to be transferred to the serial to IR from PC, and not from EVM, this can cause race conditions. This is because if EVM requests transfer before the PC is ready, then the system goes into a deadlock.

There were two possible solutions to this problem. Firstly, if the windows event handler, the interrupt handler, is modified to put the semaphores appropriately, it could be solved. However this would require a lot of time and because we came across this problem rather late in schedule, we decided to solve this problem using the second way. The second possible solution to this problem was to use delays right before the request transfer function call in the EVM code. This enables the PC to have more time to be ready first.

Encryption and Decryption

Security – Introduction and Previous work

Our wireless communication requires an absolutely secure channel, because the transaction contains the important financial information and sensitive personal information. The IrFM standard 1.0 provides the set of rules for the encryption requirements, but they do not specify the algorithms for the encryption.[10] However, they do provide the ability to choose different algorithms for each session to ensure the compatibility of the devices. The IrFM devices are commercially in service in Korea, called Moneta. Although the technical details behind their devices are not documented anywhere, we were able to figure out that most of the commercial IrFM devices in Korea use SEED[10]algorithm which uses 128-bit symmetric key and operates on 128-bit data blocks. The algorithm is developed by KISA (Korea Information Security Agency) and it is the encryption standard widely used in Korea. The key advantage of using a symmetric key is that it only requires small number of computations. However, this comes at a cost of trading some security. The symmetric key systems are safe based on the assumption that the key can be transmitted safely, but we cannot always assure such transmit path, thus we want to consider using asymmetric key system. We considered few possible encryption algorithms.

[10] SEED Algorithm Specification, http://www.kisa.or.kr/seed/

Asymmetric Key Encryption / Decryption

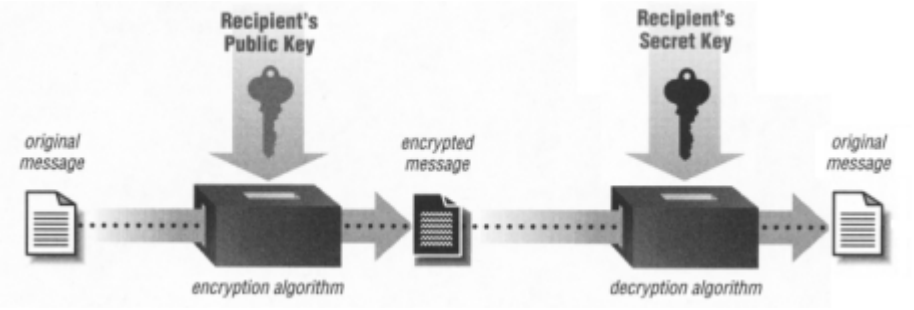


Figure 11: Asymmetric Key Cryptography System
http://www.neomailbox.com/ssl_session_encryption.html

RSA (Rivest-Shamir-Adleman) Data Security has been dominating the public key cryptography for decades. The proven security of RSA algorithm attracted many applications that require secure connection. For example, SSH, one of the most widely used remote computer access client and server, uses RSA public key encryption to ensure that user's passwords cannot be sniffed by the hackers. However, RSA is very expensive in terms of computation, because of the nature of the algorithm. In order to implement the RSA, we need prime factoring and many mod operations, which tend to be one of the most expensive operations in common the processors. Our computation power is restricted to small hand held devices with low power DSP processor or embedded processor. In addition, RSA algorithm requires a lot of memory to work efficiently, which we do not have in our small platforms. For these reasons, we find the RSA encryption to be too expensive to implement for our project.

Realizing the computation limits, we looked for a fast and small algorithms designed for embedded platforms. A group from previous year used Tiny Encryption Algorithm (TEA) for their project, because it is very cheap in terms of computation and memory usage. However, as the group pointed out, TEA has some redundancy so called, "equivalent key" that reduces the effective encryption bits. This was not a big problem for the previous group, but since we are dealing with financial transactions, the weakness of the encryption can be a significant threat to the creditability of the system.

In order to achieve both goals, cheap computation cost and strong encryption, we looked into elliptic curve cryptography algorithm[11]. Elliptic curve cryptography is relatively new compared to RSA. This algorithm uses the mathematical properties of elliptic curve to provide secure and powerful cryptography system. The algorithm requires the understanding of abstract algebra, group theory and linear algebra. For this reason, not many people can fully understand this algorithm. The encryption and decryption algorithm solves the Elliptic Curve Discrete Logarithmic Problem (ECDLP) to process the data.

Elliptic Curve Cryptography devices require less storage, less power, less memory, and less bandwidth than other systems. This allows you to implement cryptography in platforms that are constrained, such as wireless devices, handheld computers, smart cards, and thin-clients. It also provides a big win in situations where efficiency is important. For example, the current key-size recommendation for legacy public schemes is 2048 bits. A

[11] http://www.certicom.com/

vastly smaller 224-bit ECC key offers the same level of security. This advantage only increases with security level—for example, a 3072 bit legacy key and a 256 bit ECC key are equivalent—something that will be important as stronger security systems become mandated and devices get smaller. [from the certicom website]

Regardless of all of the advantages, the difficulty of the algorithm implementation forces many developers to choose RSA, however, when the elliptic curve is implemented correctly, it offers a factor of about ten in speed, which is very attractive in embedded application. In order to overcome the excess computation requirement of RSA, many companies, such as Microsoft, Sun Microsystems and Motorola, began to invest in elliptic curve cryptography in recent years. Our group decided to implement the Elliptic Curve Cryptography at the end.

Algorithm – Elliptic Curve Cryptography [from the certicom website]

The basic idea of the algorithm is to solve an Elliptic Curve Discrete Logarithm Problem to embed a message into an elliptic curve and decrypt message from the curve. Here is a brief description of the general algorithm. (Figures and examples are from the certicom website)

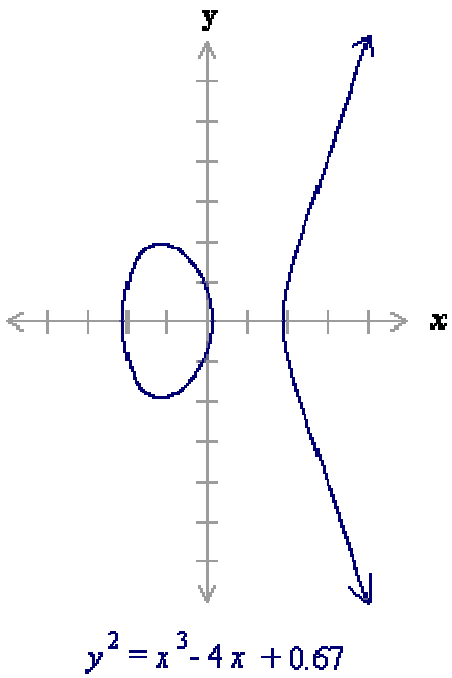


Figure 12: Example from Certicom Website

An elliptic curve over real numbers are defined as the set of points (x,y) which satisfy an elliptic curve equation of the form:

$y^2 = x^3 + ax + b$

where x, y, a and b are real numbers.

Each choice of the numbers a and b yields a different elliptic curve. For example [certicom], a = -4 and b = 0.67 gives the elliptic curve with equation $y^2 = x^3 - 4x + 0.67$; the graph of this curve is shown on left.

If $x^3 + ax + b$ contains no repeated factors, or equivalently if $4a^3 + 27b^2$ is not 0, then the elliptic curve $y^2 = x^3 + ax + b$ can be used to form a group. An elliptic curve group over real numbers consists of the points on the corresponding elliptic curve, together with a special point O called the point at infinity[12].

Calculations over the real numbers are slow because most of the hardware does not have support for the specific bit lengths used in the cryptography systems. It is also inaccurate due to round-off error. Our application requires fast and precise arithmetic operations; thus elliptic curve groups over the finite fields F2m is used as computation basis for the project. Elements of the field F2m are m-bit binary strings which are represented as bitmaps of unsigned integers. The rules for arithmetic in F2m can be

[12] http://www.certicom.com/index.php?action=ecc,ecc_tutorial

defined by either polynomial representation or by optimal normal basis representation. [Certicom] We defined F2m by optimal normal basis representation in our final version. At first we tried to implement using polynomial representation, but we had some trouble with the EVM. Look at the implementation issues section for the details. The computations can be done efficiently since we are using bitmap representation of the fields.

In order to use binary representation, we need to pick the elements a and b within F2m field with the condition that b is not 0. Due to the characteristics of the elliptic curve, the equation needs to be modified to work with binary representation as follows.

$y^2 + xy = x^3 + ax^2 + b$

All points (x,y) which satisfy the equation over the field F2m can be found on the elliptic curve that can be represented with the above equation. There are finitely many points that satisfy the conditions on the corresponding elliptic curve.

To help understand the algorithm, here is a small example from the Certicom website[13]. Consider the field $F_2$4, defined by using polynomial representation with the irreducible polynomial $f(x) = x^4 + x + 1$.

The element g = (0010) is a generator for the field . The powers of g are:

$g^0 = (0001)$ $g^1 = (0010)$ $g^2 = (0100)$ $g^3 = (1000)$ $g^4 = (0011)$ $g^5 = (0110)$
$g^6 = (1100)$ $g^7 = (1011)$ $g^8 = (0101)$ $g^9 = (1010)$ $g^{10} = (0111)$ $g^{11} = (1110)$
$g^{12} = (1111)$ $g^{13} = (1101)$ $g^{14} = (1001)$ $g^{15} = (0001)$

In a true cryptographic application, the parameter m must be large enough to preclude the efficient generation of such a table otherwise the cryptosystem can be broken. In today's practice, m = 160 is a suitable choice. The table allows the use of generator notation (g^e) rather than bit string notation, as used in the following example. Also, using generator notation allows multiplication without reference to the irreducible polynomial

$f(x) = x^4 + x + 1.$

Consider the elliptic curve $y^2 + xy = x^3 + g^4x^2 + 1$. Here $a = g^4$ and $b = g^0 = 1$. The point (g^5, g^3) satisfies this equation over F_2m :

$y^2 + xy = x^3 + g^4x^2 + 1$

$(g^3)^2 + g^5g^3 = (g^5)^3 + g^4g^{10} + 1$
$g^6 + g^8 = g^{15} + g^{14} + 1$

$(1100) + (0101) = (0001) + (1001) + (0001)$
$(1001) = (1001)$

[13] http://www.certicom.com/index.php?action=ecc,ecc_tutorial

The fifteen points which satisfy this equation are:

$(1, g^{13})$ (g^3, g^{13}) (g^5, g^{11}) (g^6, g^{14}) (g^9, g^{13}) (g^{10}, g^8) $(g^{12}, g^{12})(1, g^6)$ (g^3, g^8) (g^5, g^3) (g^6, g^8) (g^9, g^{10}) (g^{10}, g) $(g^{12}, 0)$ $(0, 1)$

These points are graphed below:

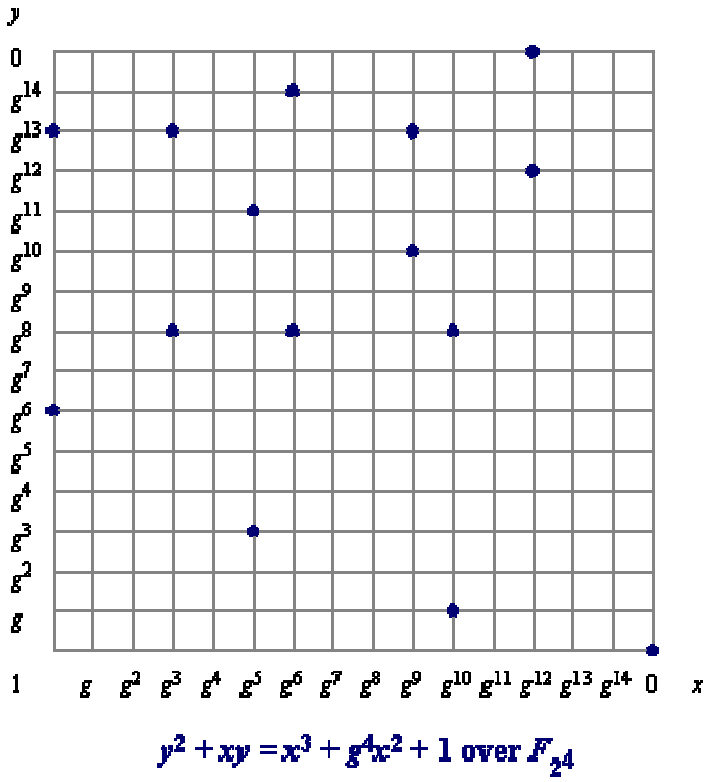


Figure 13: Example from www.certicom.com

Now we understand how the mathematical operations can be performed on the elliptic curve, we can finally define our problem. The discrete logarithm problem in the multiplicative group Zp^* can be defined as finding a number k such that r = qk mod p, given elements r and q of the group, and a prime p. Then the elliptic curve discrete logarithm problem on the elliptic curve groups is defined as find a number that Pk = Q. We will use this property of discrete logarithm problem to achieve a strong encryption algorithm as described in the next section.

Algorithm – Elliptic Curve ElGamal Protocol

The actual public key protocol we implemented is based on the ElGamal. "The ElGamal algorithm is an asymmetric key encryption algorithm for public key cryptography which is based on Diffie-Hellman key agreement. It was described by Taher Elgamal in 1984. The ElGamal algorithm is used in the free GNU Privacy Guard software, recent versions of PGP, and other cryptosystems. ElGamal can be defined over any cyclic group G. Its security depends upon the difficulty of a certain problem in G

related to computing discrete logarithms"[14]. Given the fact that the Elliptic Curve Discrete Logarithm Problems are very hard to solve, we are assured that ECElGamal provides a strong encryption scheme.

The ElGamal protocol can be break down into three major parts, the key generation, encryption and decryption. Here is a description of each step.

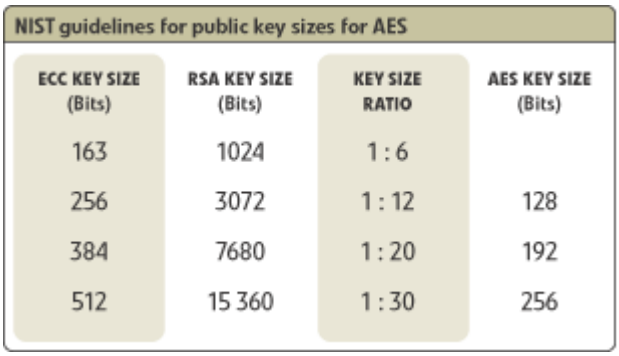
Key Generation
PTD generates an efficient description of a cyclic group G of order q with generator g. PTD chooses a random x from {0, … , q-1} PTD computes $h = gx$ PTD publishes h, along with the description of G,q,g, as its public key. PTD retains x as its secret key.
Encryption
In order to encrypt a message m for PTD under its public key (G,q,g,h), POS converts m into an element of G. POS chooses a random y from {0, … , q-1} , then calculates $c1 = g^y$ and $c2 = m \cdot h^y$ POS sends the ciphertext $(c1,c2)$ to PTD.
Decryption
In order to decrypt a ciphertext $(c1,c2)$ with POS secret key x, POS computes $c_2(c_1^x)^{-1}$ as the plaintext message. The decryption algorithm produces the intended message, since $c_2(c_1^x)^{-1} = \frac{m \cdot h^y}{g^{xy}} = \frac{m \cdot g^{xy}}{g^{xy}} = m$

Elliptic Curve Cryptography - Implementation Issues and Results

We used "Implementing Elliptic Curve Cryptography" by Michael Rosing for the reference. In this book, he provided basic codelets that can be used for the encryption and the decryption using C. However, his C codes were written in old style that was not ANSI C compatible. In addition, he did not consider porting the code for DSP hardware such as TI EVM we used in our project. Therefore, his code did not run on the EVM at the beginning due to various precision problems caused by casting between data types. We were never able to figure out how to make the polynomial based representation of F2m work on the EVM. There was a strange behavior when we used long type variables on the EVM and then converted into unsigned int. After almost five days of struggle, we finally made the code to compile and run without errors, but we had to use the optimal normal basis to represent the field. When we first planned our project, we were aiming for 131-bit encryption; however, after we implemented our first prototype on the EVM, we realized that 131-bit encryption would require too much computation and memory. After few tries, we had to lower the number of bits used for the encryption to 69-bits. However, 69-bits of encryption would be still strong compared to most of the current algorithms used in the products. Since the elliptic curve cryptography provides the encryption

[14] http://en.wikipedia.org/wiki/Elgamal

strength of factor of 6 compared to RSA, our 69-bit implementation would be approximately equal to 414-bits of RSA.



NIST guidelines for public key sizes for AES

ECC KEY SIZE (Bits)	RSA KEY SIZE (Bits)	KEY SIZE RATIO	AES KEY SIZE (Bits)
163	1024	1 : 6	
256	3072	1 : 12	128
384	7680	1 : 20	192
512	15 360	1 : 30	256

Supplied by NIST to ANSI X9F1

Table: http://www.certicom.com/index.php?action=ecc,ecc_faq

This table from the Certicom website provides the corresponding encryption strength at different bits used for different algorithms. Using this fact, we can justify our decision on reducing the number of bits used for the encryption. After reducing the number of bits, we were able to run the actual encryption and decryption on the EVM board.

However, the initial performance turned out to be too slow to be used for an actual system. The details of profile data can be found in the performance evaluation section. Our initial encryption took about 150 million cycles for just a single 69-bits of data. After realizing our code was not put on the on-chip memory, we made modifications so that all the computationally expensive portions of the codes get allocated in the fast on-chip memory. Then the cycle time went down to approximately 40 million cycles. This was a big improvement but it was still not very satisfactory. We were expecting about a factor of 15 improvements. Thus, we turned on the optimizing level to O3, then did the profiling again. This time, the cycle count was reduced to approximately 8.5 million.

The actual data that is being transferred between the POS and PTD is usually about 256 bytes. (512 bytes after encoding) Since we can break up the message into 8 bytes (64bits) of basic blocks and encrypt them, we need to run the encryption routines 32 times. This means that each message will take 272 million cycles. Since the EVM CPU runs at 133Mhz, this means the encryption takes about 2.05 seconds. This number is reasonable, although this does not meet our goal from the beginning.

Limitations and Reality

Our project did not consider a problem of exchanging the public key with malicious POS. We did not have enough time to solve this. Therefore, even with a strong encryption, our system still can be vulnerable to fraud transaction if there is a fake POS that pretends to be an authorized merchant. However, the current credit card system also has the same problem inherently. We rely on the central authority such as VISA to ensure us that there will be no POS that cheats on our system. In addition, the infrared device must be close to POS and directed at the sensor of POS, so the user should be able to see the actual target transaction system, which reduces the threat of fraud. This location verification is a natural advantage of infrared system over other financial system. Also, most of the cell phones and PDAs nowadays can access the internet. Since each POS device will have a unique ID embedded into the system when manufactured, the hand held device could request verification of the POS through the web to ensure the authenticity. The ID can be encrypted in such a way that only the central authority can decrypt the ID to prevent duplication of POS.

Speed and Profiling

The final program size is too large to place everything on chip. Only the most time critical and computationally intensive sections of the program are on chip. These include the start bit correlator, and the decimator from the receiver, the Viterbi decoder from error correction, and encryption, decryption from the security section. The speed was also recoded with a higher level of compiler optimization, but the higher levels were not able to work with the receiver. All of our data section of the memory could fit on chip. Unlike other groups working with images, our project does not use any memory intensive algorithms.

The receiver functions are required to finish within 0.0208 ms before the receive interrupt is called. The encryption and decryption routines take most of our computation cycles. This is due to algorithmic expense inherent in a public key cryptography system. Even though the Elliptic Curve Cryptography claims to be fast and efficient compared to other public key systems, it is still expensive. After optimizations, the speed of encryption and decryption was brought down to a reasonable range. Since our encoded packet size is about 512 bytes, our encryption worked on a 8 byte block over 256 byte per packet. This takes about 2.04 seconds.

Table 1: Profiling key functions with O1

Function	Speed (cycles)	Speed (ms)
Correlate with start bytes	485	0.00365
Receive data byte	1,574	0.01183
Viterbi decoder (per byte)	16,200	0.1218
Encrypt Initialize	8,000,000	60.15
Encrypt Message (69 bits)	40,000,000	300.75
Decrypt Message (69 bits)	18,000,000	135.3

Table 2: Profiling key functions with O2

Function	Speed (cycles)	Speed (ms)
Correlate with start bytes	329	0.00247
Receive data byte	-----	------
Viterbi decoder (per byte)	9,400	0.07070
Encrypt Initialize	7,200,000	54.14
Encrypt Message (69 bits)	8,000,000	60.15
Decrypt Message (69 bits)	3,500,000	26.32

Table 3: Memory Usage

	On-Chip	Off-Chip (SBSRAM)
Program Code	~48KB	~28KB
Data	ALL	NONE

Demo / GUI

Our project demo was done to show that the combination of all the parts was working correctly. In order to show this, we implemented a graphical user interface. Our initial proposal for this was to have a user-level implementation for the PTD and POS. However in our final demonstration, we just had a GUI for the PTD and printed out results on the POS EVM and PC side.

The Demo was done like the way it was described in the Link/Protocol layer. The user initiated the transaction by starting the PTD GUI program. This initiated the transfer of POS Info Request form PTD to POS. When POS received this message, the POS sent back its information including its vendor name and address. Then the PTD displayed the info in the prints. After that the PTD sent its public key to the POS, and upon receiving the public key, POS sent its own public key back to the PTD. Once the keys were exchanged, the PTD encrypted its credit card information and sent it to the POS. The POS was able to print out the PTD's credit card number.

Although we did not show full demo of a purchase of an item due to the lack of time at the end of the project (because we spent a lot of time debugging numerous unexpected bugs), the current demo was sufficient to show key implementations in a transaction procedures such as encoding/decoding, encryption/decryption and link/protocol layers. The demonstration of item purchases would be same procedure as was showed in the demo but with different packets sent back and forth. We also showed some tests of error cases and how the system accounts for them. For example, when we were sending encoded data across from POS to PTD, we purposely made error in the packet to demonstrate that the error correction works. As expected, the PTD side still printed "Giant Eagle" and its address correctly. Also, we showed that the POS does not send out its information if the predefined Request POS Info message does not match the first incoming message, as we defined the Request POS Info to be 100 in both PTD and POS.

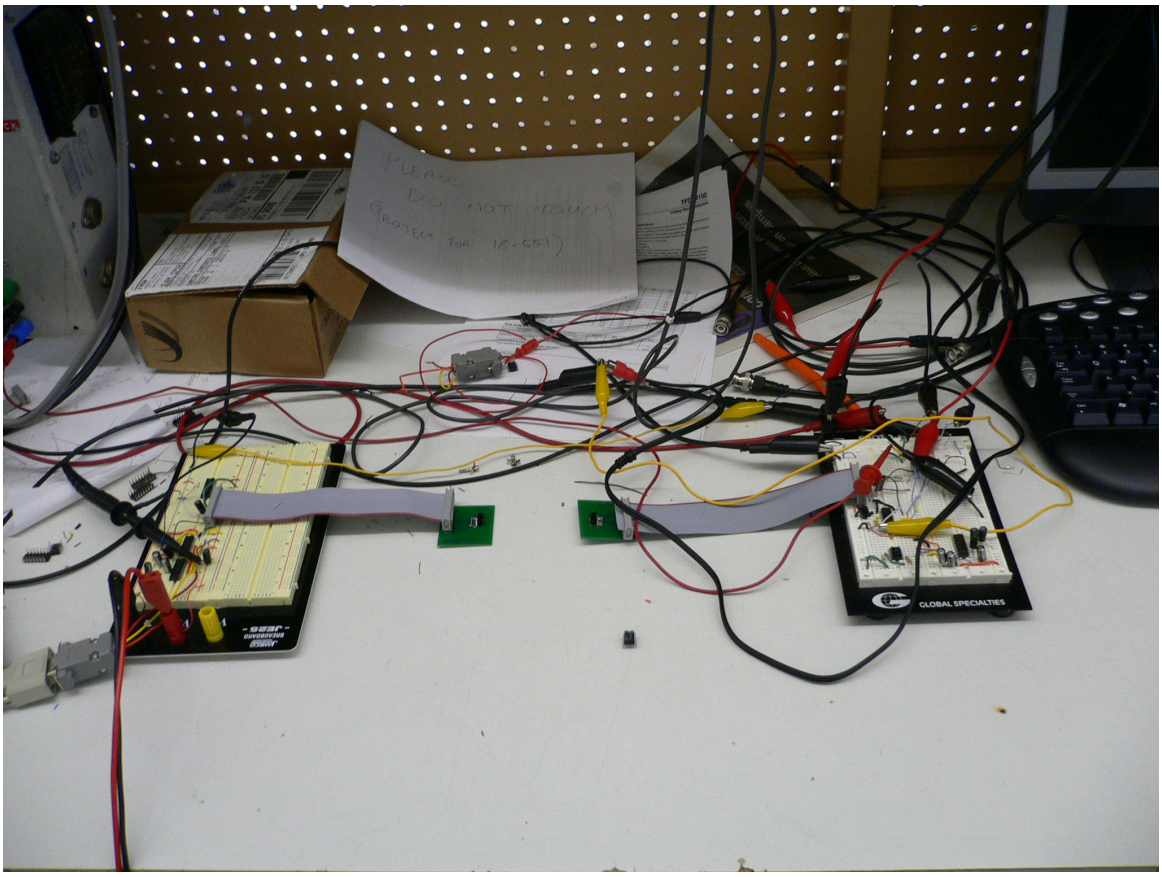
Difference from other Projects in 18-551

This project is very similar to The Wireless Intercom System project from last year. The two projects both use wireless communications and solve problems with synchronization, encryption, and encoding. This project is different because the BER is lower so that large encoding and spreading processes are not needed.

This project is also similar to Software Defined GSM Receiver. The main difference is that they focused on a receiver for a complicated standard, while we built a communications system on our own defined standard.

This project is unique because it implements wireless communications in the real world rather than simulations. Secondly, this project include a stronger encryption method than has been done before in 551.

Conclusion



The final project results included most of the important features of the system proposed. We successfully created the physical hardware (shown above), sending and receiving packets, convolutional encoder, Viterbi decoder and the Elliptic Curve Cryptography. However, due to the time constraints caused by various problems, including physical hardware blowing up five minutes before the demo, we did not have satisfactory demo and the protocols as we proposed at the beginning. We also have to admit that our system is still vulnerable to fraud transactions. This issue could have been handled through some protocols, but we did not have time to think carefully about this.

Regardless of all the troubles we had, we believe our project was successful. Our project involved many difficult, but realistic, problems such as dealing with noises, drifts in the signals, encryption and decryption. We had so many unexpected problems towards the end of our project and we learned how to schedule the work with considerations for such situations. After all, it was a very distinct experience to sleep under the desk in the lab. We all enjoyed working on the project and we all wish good luck for up-coming 551 students.