Joseph Rowley -- jrowley
Lab section: 3 MW 4-6 TA: Jay Roldan
Due: 3/14/2012
Lab Partner: Eric Cao
Other Group: Tyler Smith, Rutherford Le

# Lab 8: Diffie Hellmen Key Exchange Encrpytion System Over Twitter

## Purpose:

In this lab we choose to implement the Diffie Hellmen Key Exchange system using the HC11 and 2 computers running Python scripts. We communicated publicly using Twitter to show the beauty of this simple encryption algorithm. The idea is that we can demonstrate how public key exchange can work in a truly public environment.

## Procedure:

There are several different components involved with this project but first we developed a work flow to document exactly how the program would work. We refined and rewrote this several times and came up with many different possible ways of implementing the key exchange system. Eventually we choose one system and began coding. We split the work into HC11 assembly programming and Python coding. First we got basic serial communications between the computer and HC11 working and then worked on getting the Twitter API working in Python. Then we worked on getting the pulse accumulator to create random numbers and wrote an algorithm to check if that random number was prime. Then we worked on finding a number less than that prime (to be used as the public int) and setup a system for having the user enter their secret number onto the HC11 by the hardware switches. Next code was developed for using the LCD and displaying the numbers the HC11 came up with. Next we wrote code to compute the shared secret key. The final part of the project was implementing the serial communication between the HC11 and the Python script. Decrypting and encrypting messages is pretty simple as it is a simple bitwise XOR operation on the encrypted or plain text with the shared secret number.

## Algorithms and Other Data:

First I will explain the Diffie Hellmen key exchange system and then explain how we implemented those parts on the HC11 and in Python.

### Diffie Hellmen Key Exchange System:

The entire purpose of the Diffie Hellmen key exchange system is to come up with one number (the shared secret) than only two parties know. In this case we will use “Alice” as name of the party connected to the HC11 and “Python Pete” as the computer running independently of the HC11. This shared secret number is computed by a series of operations on numbers that are publicly visible to anyone listening in (such as Eve, the eavesdropper), however only Alice and Pete will be able to compute the shared secret (unless a brute force method is applied).

1. Alice Computes Initial Values and sends them

    a. Alice, the HC11, comes up with a prime number greater than 2. This is the public prime.
    b. Alice computes an integer that is less than prime number. This is the public int.
    c. Alice accepts a personal secret number via switches that is less than the prime number minus 1. This number is only known to Alice (and her operator).
    d. Alice computes the first public key by taking the public integer to the power of her random secret and then taking the modulo of the resultant of that operation with the prime number.
    e. Alice sends the prime number, public int and public key 1 to Pete.
2. Pete Receives & Responds
    a. Python Pete receives the prime number, public int and public key 1 from Alice.
    b. Pete excepts a random secret from the user via console.
    c. Pete then computes the second public key by taking the public integer to the power of his random secret and then taking the modulo of the resultant of that operation with the prime number.
    d. Pete sends public key 2 to Alice.
    e. Pete computes the shared secret key by taking public key 1 to the power of his secret number and taking the modulo of the resultant.
3. Alice Receives Public Key 2 and computes the shared secret
    a. Alice receives public key 2 and computes the shared secret key by taking public key 2 to the power of his secret number and taking the modulo of the resultant.

Now both Python Pete and Alice have a shared key that only they know. They can use this number to encrypt and decrypt text communications now.

**Example Exchange:**

| **Alice** | **Python Pete** |
|---|---|
| Prime = 13 | → Prime = 13 |
| Public Integer = 6 | → Public Integer = 6 |
| Random Secret = 3 | Random Secret = 10 |
| Public Key 1 = $6^{3}$ Mod13 = 8 | Public Key 2 = $6^{10}$ Mod 13 = 4 |
| Shared Secret = $4^{3}$ Mod13 = 12 | Shared Secret = $8^{10}$ Mod13 = 12 |
| Public Key 1 = 8 | Public Key 2 = 4 |
| Shared Secret = 12 | Shared Secret = 12 |

There are several components of the key exchange that the HC11 handles. It has to compute random numbers, determine if that number is prime and then store it if it is.

### Computing Random numbers on the HC11:

To compute random numbers on the HC11 we used the pulse accumulator. The pulse

accumulator iterates through the numbers 0 to 255 every 10 ms or so and the interrupt button is pushed when we want to get a number.

### Determine if a number is Prime on the HC11:

```
for ( i = 2, i < (testnumber/2), i++){
      if ((testnumber % i) == 0) {
            not prime, break loop
      }
}
```

Additionally base cases were added that the prime number must be greater than 11, because if it is less than 11 we may spend a lot of time looking for integers that will also be compatible.

### Finding a General Int on the HC11:

The next task is to find another random number and determine if it is less than the prime. If so then it will be the general int.

### Get Random Secret from User via Switches:

Next we have to get user input from the switches and check to see that it isn't too large to be the secret key.

```
display prime and general int on LCD
//user inputs potential random secret on switches
if (public prime - 1 ≤ input){
      Display error on lcd
      }
else {accept value
      and compute first public key
      }
```

### Computing the Public Key on the HC11

To compute the first public key the HC11 takes the public integer to the power of the random secret and then takes the modulo of the resultant of that operation with the prime number. The HC11 does not have a built in exponent function so to take an exponent we took advantage of the MUL function which allows 8 bit multiplication.

***Evaluating Exponents on the HC11 (Pseudocode)***

```
//initialize vars:
Random Secret = 3 = the power
Public Int = 6
total = public int
powercounter = random secret
power {
       powercounter - 1;
      if (powercounter = 0) then we're done
```

```
else total = total * public int
jmp power
}
```

In the event that the result is greater than 255 we just store this as 255. This is a bad practice and is addressed in the “What went wrong section”.

***Modulo on the HC11 (Pseudocode)***

The final step of computing the public key on the HC11 is taking the modulo of the result of the exponent function and the prime number.
ex) Public Key 1 = 6 3 Mod13 = 8
To perform the modulo operation we use the IDIV instruction which “performs an unsigned integer divide of the 16-bit numerator in D accumulator by the 16-bit denominator in index register X and sets the condition codes accordingly. The quotient is placed in index register X, and the remainder is placed in the D accumulator.” We aren’t interested in the quotient so we just store the remainder which is in accumulator D as our Public Key.

## Computing the Public Key in Python

Computing a public key in Python is trivial and only takes 2 lines of code because Python is a powerful, high level language. The following is an excerpt from the actual code required for the calculation:

```
import math
publicshared2 = pow(publicnum, secretnum) % publicprime
```

## Computing the Shared Secret on the HC11

Computing the shared secret on the HC11 uses the exact same method as described before for creating the first public key, except the operands are switched around.

```
Shared Secret = Public Key^ Random Personal Secret Mod Prime
```

## Computing the Shared Secret in Python

Computing the shared secret in Python is just as easy as calculating Public key. The following is an excerpt from the actual code required for the calculation:

```
import math
sharedsecret = pow(publicshared, secretnum) % publicprime
```

## Decrypting Text on the HC11

To decrypt text on the HC11, we take a char via serial, XOR it and display the ASCII character on the display.

## Encrypting Text in Python

To encrypt and decrypt text in Python we wrote two functions, one for encryption and one for decryption. Each take two parameters, a key value for the shared secret and a string for the text to be encrypted or decrypted. It then returns a string.

```
def encrypt (key, string):
        encryptchar = []
        for ch in string:
                encryptchar.append(ord(ch) ^ sharedsecret)

        encstring = ''
        for c in encryptchar:
                encstring = encstring + str(c) + ' '

        return encstring
```

**Example Output:**

```
Shared Secret    : 12
Plain  Text      : Encrypt me
ASCII Plain Text : 69 110 99 114 121 112 116 (space)109 101
ASCII Cypher     : 73 98 111 126 117 124 120  44     97  105
Cypher Text      :  I  b  o   ~   u   |   x    ,      a   i
```

## What went wrong or what were the challenges?

There were many challenging aspects to this problem. For one we are interfacing the HC11 over serial and communicating with it with a program that we wrote. None of us had extensive experience with the Python, serial communication or the Twitter API however we figured out a lot of it as we went.

One initial challenge we faced was that the computers in the CE lab restricted the installation of Python modules, so that meant that interfacing over serial via Python from the lab computers would not possible. To mitigate the situation we bought 2 USB to Serial adapters however these didn't arrive until the day before the assignment was due. Luckily Daniel, a TA, was able to provide a USB to Serial adapter for us to use.

The first programming issue we had was with the Twitter, specifically *Python Twitter,* a Python wrapper around the Twitter API. We installed the following the dependencies: simplejson, httplib2, python-oauth2 and then installed setuptools. Then we ran the following commands to build, install and test the Python Twitter wraper.

```
python setup.py build
python setup.py install
python setup.py test
```

The build and install ran perfectly however, the test failed numerous times reporting several .json files as nonexistent. After a lot of Googling and reading through forums I discovered that the test was actually inaccurate and after further testing I signed up for a Twitter Dev account and was Tweeting and reading Tweets without problem from Python.

One issue that was never resolved was with uploading code to the HC11 via serial with Python. We were able to connect to the HC11 with the baud rate of 9600bps and then reset it by sending the ASCII character 24 over the serial connection. Then we waited, sent the "D" followed by a newline and then we sent over each line of binary final, line by line. After we

uploaded the binary file the HC11 reported that it successfully had been uploaded. After that we sent over the “E” ASCII code followed by a newline. This should have executed the code however it did not at all and instead the “E” was echoed back out. This error came late in the build process and we could not resolve it. As a backup plan we loaded the binary file on to the HC11 by using Teraterm on a lab computer and executing it on there. Then we manually connected the HC11 to our computers and established a serial connection.

Another issue we have is that the HC11 is only able to perform 8 bit multiplication and that means that we were limited in our exponent function to the same amount. This severally limits the overall strength of our algorithm. That being said having an 8 bit encryption key is in itself ludicrous as brute forcing the possible combinations is exceedingly simple (there are only 256). Additionally, because of the 8 bit maximum size multiplication function we have to truncate all values over 255 and store them as 255. This creates an uneven distribution of keys, as anything over 255 is stored as 255. With this in mind we had to cover the base case for the modulo operation of 256 and store its result as 1 instead of 0.

## Other Information:

## Conclusion:

This lab taught us many valuable lessons. On the technical side I learned a lot about the complexities of serial communication and handling ASCII transmissions. I also learned a lot about Python and the Twitter API. On the HC11 side of things I learned a lot about the complexity of using the HC11 to generate random numbers but also how a simple encryption algorithm can be implemented without too much work. It’s very interesting to see how the HC11 operates and uses the pulse accumulator as seed for a random number generator. Unfortunately this “random number” isn’t very random. One potential idea for future research would be to output a set of “random” values from the pulse accumulator to the serial port and then do analysis of that data to assess truly how random it is. The Ziggurat algorithm or many others could be used to sample the random data and then a distribution of the numbers could be established. This could be compared to other pseudorandom number generators. The whole idea of implementing Diffie Hellmen key exchange over Twitter on the HC11 came up as kind of a joke but as we talked more we realized that it would actually be possible. It showed me that I can do something really interesting and challenging if I have support and motivation from a smart group of people. Also it taught a lot about the importance of the initial planning stages when designing a larger application. I really enjoyed the challenge and can’t wait to do more complex projects.