Strath Name

Doctoral Thesis

Thesis Title

Author:
John Smith

Examiner:
Dr Man Page

Supervisor:
Dr. James Smith

Phd Student:
Greg

A thesis submitted in fulfillment of the requirements
for the degree of Doctor of Philosophy

in the

Research Group Name
Department or School Name

March 20, 2016

University of
Strathclyde
Engineering

Declaration of Authorship

I, John SMITH, declare that this thesis titled, “Thesis Title” and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date:

STRATH NAME

Abstract

Faculty Name
Department or School Name

Doctor of Philosophy

Thesis Title

by John SMITH

The Thesis Abstract is written here (and usually kept to just this page). The page is kept centered vertically so can expand into the blank space above the title too…

Contents

Chapter 1

Introduction

The Internet of Things or IoT is the concept of a huge network of physical objects connected and communicating to themselves and to the world wide web. Devices can include domestic appliances, buildings, cars. As it becomes a rapidly growing concept with over 50 million devices expected to be connected to the web by 2020[1], the security of the transmissions of these devices is becoming a more and more pressing issue. IoT's main benefits are the remote control of devices and appliances, for the device to have the ability to send information about it's state, such as a vending machine reporting that it has run out of a certain item, and to allow the machines to be more automated and to work with other machines, like a home hub device that can turn on the lights and central heating when an occupant is arriving home, with the lights and heating not being connected to each other but to the central hub.

However IoT will be ultimately be useless if it is unsecure. IoT is an emerging field but there have already been some high profile security disasters. Ranging from relatively less serious problems such as some "hackers" been able to glean important wifi information from your internet connected lights [2] to very concerning, potentially fatal security breaches like someone gaining unauthorised access to your car and assuming control. There have been three examples of this in the last few years with a Jeep Cherokee[3], a Toyota Prius[4] and a Tesla Model S[5] being the cars effected. The hackers were able to control the accelerator, door locks and brakes, among other things. This highlights a very real problem that will only become more important. Too often security is an afterthought but it really needs to be built into products from the offset.

The challenge is to provide a cryptographic solution that is similar in strength of security to solutions that are implemented on servers and computers but on much smaller and less powerful devices. A solution that performs at an acceptable speed, good security with reduced power consumption.

Google and British Gas have recently released Nest and Hive respectively. Nest was released in 2014 and Hive in 2013. These both involve controlling your central heating, among other things, remotely and programming in days when you won't be at home and therefore have no need of heating. However, it was two whole years later upon independent investigators discovering that information, about dates when the heating was on because the occupants were in and off when they were away from home, was being sent unencrypted that British Gas Hive decided to encrypt their products transmissions. With the release of Hive 2 they patched the problems found but they should never have been there. Found in the same investigation, Google had a lesser fault which was sending the post code of the user unencrypted, which has since been patched. It is only when caught or there is a high profile breach that companies take the steps to full secure their customers information [**which**].

This is a new implementation of an old problem, developers and companies don't employ effective security of data, moving this to smaller devices doesn't change this fundamental problem. There are studies about and examples cryptographic systems for microcontrollers, it is not something that cannot be done. However, perhaps it is too difficult or complicated at present for smaller development teams or has two much of a food print in terms of computational resources and time that it gets pushed to the side. Data security is a fundamentally important concept and one that will need to implemented in every application that involves user's private data otherwise there is the potential for loss of money, intellectual property, goods, reputation and health. These problems affect both companies and the consumer.

This project will look at a way to simply secure the private data of a user as it is sent across unsecure networks, the internet. The example used to work with is the secure transmission of a users private home temperature data. If they have a system that monitors the temperature of all the rooms, that data can be used to figure out when they are likely to be home or not. So, using a base platform in the user network that has access to the temperature sensors throughout the house, it takes the sensor data signs then encrypts it and sends it to a remote server.

741

Chapter 2

Project Tools and Planning

helloaslkdhaskjdh klajdsh ak;sdjfh asdf] asdlfj hsalkdfjh asd;flk as df asldjfh laskdjfh s;dpolkfjh as df[SAIHDFASLKDJFH APESGFKA;EODLKHaskjfhAS;KLJLHakjsdh ALKSJDH laksjdh LASKJDH laskjdh ALKSJDH alksjdhlsjH

Chapter 3

Background

more detail description of IoT?

3.1 Cryptography

Cryptography is the practise and study of techniques for secure communication in the presence of attackers. To do so, one can use encryption where by messages are encoded in such a way that only authorised parties or at least parties in possession of the keys can view them. There are two main ways of encryption Symmetric Key encryption and Public Key encryption. In Symmetric Key encryption both parties have the same key the which can encrypt and decrypt messages that are sent between them. The problem is that if Bob wants to send an encrypted message to Alice, he must get the secret key to her. Currently the most secure way for the transmission of secret keys is to hand them over in person, in private. This this project Asymmetric Key encryption and Digital signatures.

3.1.1 Asymmetric Key Encryption

To get round the problem of securing sending secret keys, one can use Asymmetric Key encryption has a secret key and a public key, the public key is generated out of the secret key and are therefore mathematically linked but it is computationally infeasible to calculate the secret key from the public key. This public key can be given out freely and is not a secret. So if Bob sends a message to Alice he encrypts the message with her private key and she can decrypt it with her secret key. This type of key encryption gets past the sharing key problem but it only stops attackers from reading the message. It does not prove the message was sent by a certain person or that the message has not been altered in transit.

3.1.2 Digital Signature

This is a mathematical scheme for proving message authenticity, message integrity and message non-repudation. Similarly to asymmetric key encryption a random private key is created with a corresponding public key. Double check."!! The algorithm takes in a message and a private key and using SHA-512 and? it produces a signature. If Alice signs a message in this way, Bob can use another algorithm to verify the message with the public key and signature.

3.1.3 TweetNaCl

NaCl or “Salt“ is a simple to use high-speed library for authenticated encryption. it provides both Asymmetric and Symmetric encryption, It provides authentication and message integrity with SHA-512. The authors are Daniel J. Bernstein, Tanja Lange and Peter Schwabe but at points it relys on third part implementations for parts. The API is simple, having only a handful of methods but uses high speed, high security primitives. /refhttps://labs.opendns.com/2013/03/06/announcing-sodium-a-new-cryptographic-library/

Unfortunately the library wouldn't work completely on a Arduino, one problem is that there is no /dev/random and therefore no randombytes() which means that it can't create keypairs in the usual way. Also, as mentioned later Arduino can't use the C library as is, it needs to be converted into C++. Another problem is that the library is relatively quite large, the Arduino Due has at it's disposal 512KB flash memory and the full library is 3MB. Fortunately the same creators along with Bernard van Gastel, Wesley Janssen and Sjaak Smetsers made TweetNaCl. Which is a tiny implementation of NaCl, still providing speed and security but with a significantly smaller code size 40KB. It retains the same protections against (from tweetNack-201409..) timing attacks, cache-timing attacks, has to branches depending on secret data and no array indices depending on secret data. In addition it is thread-safe and has no dynamic memory allocation. It is portable and easy to integrate, the library is easily added as it consists of two files, there is no complicated configuration to be set up or any dependencies on external libraries. Because of this compactness it is easier to read and understand it's operation. Although not as fast as NaCl it is still fast enough for most applications. “Most applications can tolerate the 4.2 million cycles that OpenSSL uses on an Ivy Bridge CPU for RSA-2048 decryption, for example, so they can certainly tolerate the 2.5 million cycles that TweetNaCl uses for higher-security decryption (Curve25519).” TweetNaCl is still small after compilation at 11KB thus avoiding instruction cache misses. It is a full library and not a set of isolated functions for a simple NaCl application, only six functions are needed. crypto_box for public-key authenticated encryption; crypto_box_open for verification and decryption; crypto_box_keypair to create a public key; and similarly for signatures crypto_sign, crypto_sign_open, and crypto_sign_keypair. It is open source and the developers encourage it to be used as much as possible.

NaCl will move to Ed25519 signature system, what is that?

TweetNaCl encrypts messages by xor'ing them with the output of Bernsteins Salsa20?? stream cipher

NaCl crypto_stream uses Bernsteins X

Again, TweetNaCl uses SHA-512 as it's hash function with the Ed25519 signature scheme and the code is simplified compared to the NaCl implementation

For asymmetric cryptography TweetNaCl uses Bernsteins' Ed25519 elliptic curve Diffie-Hellman key exchange??

3.2 Types of attacks

3.2.1 Replay Attack

When this attack occurs the attacker replays a valid message. If Bob wants Alice to prove who she and she duly provides some encrypted signature to prove so. Eve can capture that signature, She does not know what the signature is but she knows that it is a signature. She can then connect to Bob and use this message to pretend she is Alice. To prevent this attack, use an identifier that is only valid for one use, this can be session tokens or one-time passwords.

3.2.2 Man in the Middle Attack

In this attack there is an attack between two parties, Bob and Alice, who wish to communicate. The man in the middle, Eve, changes messages as they are in transit and manages to pretend that she is the person that the other thinks they are talking to. An example is if Alice asks for Bob's public key, Eve can capture that public key, replace it with her own and send that and because Alice has no way to prove that it is Bob's key or not she accepts it. So when Alice sends a message that has been encrypted with what she thinks is Bob's key, Eve can take it, decrypt it with her key, change the message then encrypt it with Bob's real public key. Which Bob receives and believes the message is from Alice.

3.2.3 Bit-Flipping Attack

This is where the attacker can change the cipher text in some way that cause a predictable change in the plain text. The attacker does not know exactly what the plain text is. If Alice was to send a message to Bob saying that she owes him £100. If Eve knows the format of the message, she can change the number at the end into £1000.

3.2.4 Stream Cipher Attack

stream cipher attacks, if the same key is used chosen plain text attack

Side channel attack, does NaCl protect a bit against this

3.3 Machine to Machine

M2M refers to the direct communication between two devices using any sort of channel. This is a component of IoT when connected in this way small, low power sensors can transmit their data to another device which can collate, perform analysis or some extra computation or pass the data along again before it reaches a human user. Present day applications monioring the health of machinery, digital billboards or sensory networks. An application may multiple sensors in a network that pass sensor data to multiple nodes which do some computation, such as adjust machines to correct errors, replenish stock or manage a system and possibly sending information, such as state across the Internet to a human user.

3.4 Technologies used

In this project the Arduino was programmed using C++. The C++ was developed in the Arduino IDE. An Arduino Due, Arduino UNO, DS1820S temperature sensor with resistor and two Ethernet Shield R2 boards. On the server side there is an SQL server, PHP scripts that accept that Arduino data and put it in the SQL server. And a Java web application that uses Java Database Connectivity, JBCD, to access the SQL server and outputs dynamic HTML when accessed. The Java code was developed using Eclipse Jee Mars.

3.4.1 PHP

PHP is a server-side scripting language used in web development. It is especially suited server-side applications as it is very portable. PHP code in a file that is requested is executed by the PHP runtime. It can be used with many OS's, web serves and relational database management systems. It is very common for web hosting providers to support PHP use by their clients.

3.4.2 XAMPP Server

XAMPP, Apache, MySQL, PHP and Perl come together to form XAMPP which is free, open source, cross platform, web server stack package for the creation and testing of local server systems that can then be moved as they are onto live systems. This XAMPP used in this project also contains Tomcat.

3.4.3 SQL

Standard Query Language is a special purpose programming language for managing data store in relational databases. SQL organises data into databases and tables and has operations for inserting, updating, deleting, altering and querying data. The vast majority of the databases use SQL.

3.4.4 Apache Tomcat

Tomcat is an open source implementation of the Java servlet, JavaServer Pages, Java Expression Language and Java Websocket technologies. It provides the ability to produce dynamic HTML in a pure Java environment. It accepts web application archives, WAR, files for ease of updating web apps.

3.5 Secure Transmission of Keys

The problem is more acute with symmetric key encryption as it is the same key used to encrypt and decrypt a message, if the key is compromised messages can no longer be trusted. The only thing and most high tech solution is to meet in person and in private with the person you plan to communicate with and exchange keys there. If the communication with be of a diplomatic nature, the bag used to transport the keys can have special legal protection against being opened and this is called a diplomatic bag(pg 52 of book chpt 2). You could send a key over a another secure channel that you trust but there is always the chance that has been compromised and it is not possible to verify that the person on the other side of the secure channel is who you think it is without first having a secure identifier which you can't get solely through internet communication. There is a similar problem with Asymmetric key encryption, a user can freely distribute their public key and they can be pretty sure that the messages they receive are secure but it is not possible to prove who has sent you the message without their public key signature but initially public key transmission is vulnerable to man in the middle attacks.

Chapter 4

Design

4.1 Aims

The basic aims are to design and implement a basic protocol for the transmission data to a server. This protocol will have security such that the messages are protected from modification, spoofing and can't be read in transit. There is to be a server application that allows the user to view the data being sent to the server. Following on, the needs of machine to machine communication will be considered and the protocol will be expanded to include direct communication and secure transmission of keys. Finally, as this will be on a low powered device, the power usage is to be measured to establish performance and power overhead of security.

"prevents rogue devices from transmitting data to others"?!?!

4.2 Microcontroller

The first step in this application is the acquisition of data. For example data, a variable that is monitored for important applications in the real world was chosen, this being temperature. There are a range of temperature sensors out there but one that is compact, cheap, freely available, has a large temperature range, high precision and can derive power directly form the data line, parasite power, is the DS18S20. It returns a 9-BIT byte array so is ready to encrypt straight away.

pretty picture of course

The Arduino Due is a large Arduino, it is the first one with a 32-bit ARM core micrcontroller, has 54 digital I/O pins, 512KB flash memory, 96KB SRAM and a 84MHz. A lot of microcontrollers are 8-bit which means that they are limited in their cryptographic options. With a 32-BIT architecture and a large amount of flash memory it is possible to have a light weight cryptographic library for our application but the Due is still low cost enough that it can be used for an IoT sensor application. Because it is an Arduino it benefits from the large community, wide range of compatible components and large set of open source libraries.

Once the temperature data is on the board there needs to be a way to transmit the data, as the data is being sent over the internet the natural choice is an Ethernet or WiFi shield. Unfortunately although the WiFi shield would be better, as wireless improves and becomes more the norm, the more IoT applications will use it, it is much more expensive than the Ethernet Shield. The first revision of the shield is no longer made so the second revision, R2 will be utilised. IT allows for easy connection for the Arduino to the Internet, it uses the Wiznet W5500 Ethernet chip, supports up to 8 different socket connections at a speed of 10/100Mb and has a MicroSD card slot to store network settings.

4.3 Security

There a lot of security options for desktop programmes and communications like AES, WEP, SSL. In this more ubiquitous field the implementations have been around for a while but for microcontrollers only in recent times have they become powerful enough at a cheap price and therefore popular enough for developers to write or adapt cryptographic libraries suited for microntroller communications. The library in this application had to have acceptable security, really as close as possible to encryption systems in more powerful computers but still have acceptable performance, have a small enough code size to be store on the microcontroller and be easy to use. Of course, it needs to prevent attackers from reading or altering the data and proving who sent it.

some have extra chips to do the encryption

Are 4-8 bit crypto libraries just not as good as 32 bit?

TweetNaCl was chosen for this project. It is a public-domain cryptographic library that is the little brother to NaCl, a high-speed cryptographic library. NaCl aims for absolute performance with good security at the expense of portability whereas the aim of TweetNaCl is portability, small code size and auditability. The developers claim it is first auditable high-security cryptographic library. It is a recently created library, 2014 but it provides public-key cryptography, secret-key cryptography, hashing and string comparison. TweetNaCl is a full cryptography library but has only two files, needs little to no memory, has similar performance to NaCl and is easy to use. There is only one high level method that requires the necessary variables and returns the encrypted or signed message.

4.4 Server

4.4.1 Web Server

The data needs to be sent in some way, how data is passed to and taken from servers is using POST and GET requests. And it is possible to send a GET or POST request to for a file and that file then executes. It was decided the arduino would GET PHP because taking the encrypted and signed data and storing it is a relatively simple job and could be left to a simple php script but the decryption and checking signature and integrity is more complex and extra code needs to be run. There are many different implementations of NaCl in many different programming languages. It was decided to use a Java implementation from the developer Ian Preston in conjunction with Java server pages, JSP, which is a tool to dynamically create web pages using Java code. JSP could be joined with the Java implementation of the C library, NaCl to retrieve the encrypted values from a database and display them. For the database structured query language, SQL, was chosen.

4.4.2 Arduino Server

If it is desired that the temperature be taken from more than one location then there will need to be multiple nodes on the network. Or if there needs to be extra microcontrollers on the network for some other task. One such task is the distribution of public keys, if the server is to update it's public key, say as part of a regular update schedule to provide good security or if a key is known to be comprised then it will need a way to pass on the key. The Due can request a new key or the server can indicate to the Due that a new key has been made a available and the Due will make a request for that key then make a connection to the other Arduinos in the network and send them that key. An extra Arduino Uno was chosen as an extra node for testing. The network will be built using Ethernet and be an extension of the Arduino Due to web server relationship. As the Arduino Due is a client to the web server the other nodes will need to be hosts if information is to be passed between them. The setup described in **??** is desired.

Chapter 5

Implementation

5.1 Overview

There are two files in the server, connect.php and add.php. The Arduino makes a post request to the add.php which effectively just calls it and the first thing the add file does is call connect which has the server details and makes creates a connection. Following that there is a SQL query that inserts the values sent in the post request into the appropriate table entries then close the connection. cooodee snippets

The Java Web app details what the server is do when it gets various types of request, be it get or post...(more on requests?). In this type of application you can dynamically printout all the HTML that will be used to make up the page. The usual HTML, head, body tags are printed at the top and the titles in the table are printed as well. (How it gets the keys?!). The web app uses JDBC to create a driver(idk man) to get the connection to the SQL database. Then using Java language it builds up a SQL query to take out all the values from the database and executes that. This puts all the table entries into a result set and the app cycles through that results set getting the relevant information out. The signed and encrypted hex is encoded as string and some leading zeros are lost in the conversion from byte array to string in the Arduino so these are added now before the string is converted back into a byte array. There is a try catch around the crypto_box_open and crypto_sign_open method so the server doesn't crash if one result set has been broken. Following this is the conversion from hex into integer for the user to read (how does it do it?) and finally the values are written to the browser along with the ending html tags.

The Due is used in two separate ways in this project; first is the client to the java web app and second is a demonstration of secure public key transmission with another Arduino Uno and Ethernet shield.

5.2 IoT Platform

The basic concept of this platform is an Arduino Due that takes the current temperature of the room from a DS1820 temperature sensor. Then that data is signed and encrypted with TweetNaCL before being transmitted, using an Ethernet Shield, across to an SQL server. A web application takes the SQL data decrypts, checks the signature is valid then displays on a website.

graphic here pls

DS1820 is a lost cost temperature sensor that is very accurate, 9 bits of precision and is also low power. It can scavenge power from the data with the arduino and thus does not need it's own power source.

For the prototype, an Ethernet Shield was used as it is much cheaper than a WiFi shield but ultimately completes the same job. The shield is a simple way to connect arduinos to the internet. The shield used was the second revision, R2 and has a w500 ethernet controller which means it needs an different library to the first revision. Fortunately the two libraries share the same API so the the codes are compatible. Simply swap put the older shield and library and replace with the newer one.

5.2.1 Temperature reading

The DS18S20 temperature sensor is wired up with a pull up 4.7Ω resistor, between the orange wires, on the bread board shown in figure 5.1. A pull up resistor is a resistor between the sensor and the positive power supply so that the signal will be a valid logic level if external devices are disconnected or a high impedance is introduced. It is connected to digital pin 9, yellow wire, because it can't be on pins 10, 11, 12, 13 as they will be used by the Ethernet Shield. When looking at the temperature sensor the furtherest left pin is V_{dd} and normally this would be connected to the Arduinos 3.5v or 5v output but the DS18S20 is in parasite power mode which scavenge power off the middle data line, DQ. When the DQ pin is high some of the charge is stored in capacitor DS18S20 to power the device. In parasite power mode both the GND and V_{dd} are connected together, by the blue wire, and then to ground.

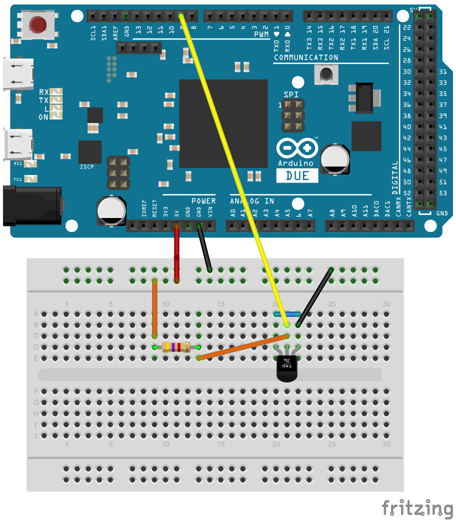

FIGURE 5.1: DS18S20 in parasitic power mode connected to Arduino

The library used to read the DS18S20 is OneWire, which is a proprietary library from Maxim. The command *ds.write(0x44, 1)* starts the internal A-D conversion operation. Once this process is finished the data is copied to the Scratchpad registers. A delay is need to charge the capacitor and to ensure conversion is complete before reading the data. Which is done with *ds.write(0xBE)* and then the data is read out using *ds.read()* then put into an array.

```
                OneWire ds(9);
                  ...
                ds.write(0x44, 1);
                delay(1000);

                present = ds.reset();
                ds.select(addr);
                ds.write(0xBE)

                Serial.print(" Data = ");
                Serial.print(present, HEX);
                Serial.print(" ");
                for ( i = 0; i < 9; i++) {
                    data[i] = ds.read();
                    Serial.print(data[i], HEX);
                    Serial.print(" ");
                }
```

FIGURE 5.2: DS18S20 temperature sensor Arduino Code

5.3 Cryptographic

```
#include <TweetNaCl2.h>
TweetNaCl2 tnacl;

byte arduinosk[crypto_box_SECRETKEYBYTES] = {...};
byte arduinosksign[crypto_sign_SECRETKEYBYTES] = {...};
byte serverpk[crypto_box_PUBLICKEYBYTES] = {...};
byte nonce[crypto_box_NONCEBYTES] = {...};

int const messageLength = crypto_sign_BYTES + 9;
byte message[messageLength] = {...};
unsigned long long signedMessageLength=0;
byte signedCipher[signedMessageLength];
unsigned char
    signedMessage[messageLength+crypto_sign_BYTES];

tnacl.crypto_sign(signedMessage, &signedMessageLength,
    message, messageLength, arduinosksign);
tnacl.crypto_box(signedCipher, signedMessage,
    signedMessageLength, nonce, serverpk, arduinosk);
```

FIGURE 5.3: TweetNaCl Arduino Signature and Encryption Code

The C TweetNaCl has been converted into an Arduino library and therefore needs an instance created called *tnacl*. With this instance the methods needed can be accessed. The keys are preinstalled to avoid the key distribution problem in lines 4-7. The next chunk sets up the messages that are two be passed between the methods. Message is initialised as a byte array of size *messageLength* which can has to be *crypto_sign_BYTES*, 32 bytes, plus the message. The first 32 bytes have to be zero for the signing to work. *signedMessage* and *signedMessageLength* are passed in by reference so after *crypto_sign()* is complete the message with the signature and the size of that array will be in those variables, respectively. The resulting signed message, normally would need the leading 32 bytes to be zero if it is to be encrypted properly but the Java implementation of TweetNaCl on the webserver takes care of that so it is not necessary here. However during testing the adding and removing encryption and signatures, it was implemented. Now that the plain text message has been signed it is to be encrypted. *signedCipher* and *signedMessage* are passed in by reference as well and the length of the cipher has the same size as the signature. The final product is the signed message cipher and that is the array sent.

5.4 Data transmission

Once the data has been encrypted it is to be packaged up and sent across the network.

```
#include<Ethernet2.h> //Ethernet Shield R2 library
#include<SPI.h>

byte mac[] = {
0x90, 0xA2, 0xDA, 0x10, 0x2D, 0xD6 //of the Ethernet Shield
};
char server[] = "192.168.0.6"; //IP of apache web server
EthernetClient client;

IPAddress clientIP(192, 168, 0, 30);

 if(Ethernet.begin(mac)==0){
    Serial.println("Failed to assign IP");
    Ethernet.begin(mac, clientIP);
  }else{
    Serial.println("Assigned IP");
  }

if(client.connect(server,80)){
    String data = "temperatureHex=";
    int contentLength = data.length()+final.length();
    Serial.println("Connected");
    client.println("POST /tempLog/add.php HTTP/1.1");
    client.println("HOST: 192.168.0.6");
    client.println("Content-Type: application/x-www-form-urlencoded");
    client.print("Content-Length: ");
    client.println(contentLength);
    client.println();
    client.print("temperatureHex=");
    client.print(final);
  }else{
     Serial.println("Failed to Connect");
  }

  client.stop();
```

FIGURE 5.4: Ethernet interfacing and transmission Code

Just before the cipher is sent the byte array is converted into a String so that it can be passed around easily as one parameter. This is completed simply be cycling through the byte array and adding each entry together. Care has to be taken when there are hexadecimals that are 0x0F or lower. The leading zero is lost during the conversion to String which means the cipher is incorrect and cannot be decrypted. This is solved by explicitly adding an extra "'0" String if the byte is less than or equal to 0x0F. First the device needs an IP address which is completed through Dynamic Host Configuration, DHCP by the router. The router running DHCP dynamically allocates network configuration parameters such as IP addresses to devices so that they automatically get one that isn't in use. This is completed with the line *Ethernet.begin(mac)* which returns a 1 on successful IP allocation or 0 on failure. If it fails then a static IP can be assigned manually by *Ethernet.Begin(mac, clientIP)*. A connection to the server is attempted using the

IP and the port number, 80 in this case. If it successful the information is sent as a POST request, a POST request is a request method in the HTTP protocol. When a server receives a POST request it knows to take the data and complete an action with it. The request makes it known that it wants add.php to be executed upon receiving the data. Once the data is sent the connection can be closed and other actions performed on the Arduino.

5.5 Server Side

For the prototype, an Apache server, SQL server and Tomcat server were set up using XAMPP on a desktop. The Arduino causes the add.php to be run and the first thing that does is make a connection to the SQL server using SQl server address, username, password and the name of the database. If it can't connect it abandons the task and returns an error message. On successful connect it returns the connection variable.

```php
<?php
    include("connect.php");

    $link=Connection();

  $temp=$_POST["temperatureHex"];

  $query = "INSERT INTO tempLog (tempHex)
    VALUE ('".$temp."')";

    mysql_query($query,$link);
    mysql_close($link);

    header("Location: index.php");
?>
```

FIGURE 5.5: Arduino to SQL interfacing code

The data to be stored is extracted out of the POST request and place in a variable. Then an SQL query is built up before being sent to the SQL server and the connection terminated. The SQL table contains an ID, the time at which the temperature data was received and the data and is created using the command shown in figure 5.6 Notice the corresponding variables, tempLog, the table and tempHex, the data.

```sql
CREATE TABLE `iotplatform`.`tempLog` ( `id` INT(255) NOT
   NULL AUTO_INCREMENT , `timeStamp` TIMESTAMP on update
   CURRENT_TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP ,
   `tempHex` VARCHAR(100) NOT NULL , PRIMARY KEY (`id`))
   ENGINE = InnoDB;
```

FIGURE 5.6: SQL database creation code

The data stored in the database is still encrypted, now a way is needed to decrypt it and display it to the user. A Java web app, using Java server pages, JSP, was created as there are Java implementations of the TweetNaCl library, among other variations, available [**ian**]. Eclipse JEE Mars was used to create a dynamic web project which extends HttpServlet for the creation of dynamic web pages. In this you override at least one of the following methods, doGet, doPost, doPut and doDelete so the depending on the type of request received a different action will occur. This application has the code in the doGet as server will receive a get request when it is accessed by a user.

```
protected void doGet(HttpServletRequest request,
    HttpServletResponse response) throws ServletException,
    IOException {
response.setContentType("text/html");
PrintWriter printWriter = response.getWriter();
printWriter.println("<html>");
}
```

FIGURE 5.7: Prepping the client printer in JSP

The headers and tags you would need and expect in a HTML page can now be print dynamically much like printing to a console. The keys are defined similarly to the code on the Arduino expect the server has it's own secret key and the Arduino's public key.

```
  final String DB_URL="jdbc:mysql://localhost/iotplatform";
  String user = "root";
  String password = "";
  try
        {
          // Register JDBC driver
          Class.forName("com.mysql.jdbc.Driver").newInstance();

           // Open a connection
           Connection conn = DriverManager.getConnection(DB_URL, user, password);

           // Execute SQL query
           Statement stmt = conn.createStatement();
           String sql;
           sql = "SELECT id, timeStamp, tempHex FROM tempLog";
           ResultSet rs = stmt.executeQuery(sql);
           // Extract data from result set
           while(rs.next()){
             //Retrieve by column name
             int id = rs.getInt("id");
             String tempHex = rs.getString("tempHex");
             Timestamp timeStamp = rs.getTimestamp("timeStamp");
  }
}
```

FIGURE 5.8: Accessing the SQL database in JSP

The app is given the server details and access to it before entering a try catch that prints whatever the error message is to the client. Java database connectivity, JDBC, is an API for client access to a database. The app gets a new instance of JDBC and then opens a connection to the server before building up a query to pull out the contents of the table. The variables are put into a result set which much be accessed for each individually variable with the string name as a parameter. The encrypted message is still in it's String format and needs to be converted back into a byte array. During the conversion to a String on the Arduino half of the leading zeros are lost and they are added before the conversion to byte array. The API in the Java implementation of TweetNaCl is the exact same as the C.

```
TweetNaCl.crypto_box_open(signedMessage, cipher, messageLength, nonce, arduinoPublicKey, serverSecretKey);
byte[] javaPlainTextMessage = TweetNaCl.crypto_sign_open(signedCipherArray, ArduinoPublicSignatureKey);
```

FIGURE 5.9: Decryption of message and verification of signature in JSP

The security needs to be taken open in the reverse order as to how it was put on. *TweetNaCl.crypto_box_open* passes the decrypted message out by ref but for *TweetNaCl.crypto_sign_open* the message without the signature is returned. The variable *javaPlainTextMessage* is the temperature in plain hexadecimal but it still needs conversion to integer so it can be read by the average user. The code is taken from the DS18S20 example.

The web app upon being accessed decrypts and checks the signature of each entry, using the keys that it has stored, in the table before converting the raw hex temperature data into more readable integers and displaying in a simple HTML table that can be accessed by the user. When the Arduino has data to send it will make a POST request to a PHP file on the Apache server which takes the data given to it and places it in the SQL server. (security flaw!)

5.6 Public Key Transmission

For the creation of other nodes on the network that are connected onto the Due, an Arduino Uno was set up as a host and example node. As two clients can't directly connect together with the Ethernet protocol and since the Arduino Due is a client to a web server, the Uno must be a host. The Uno will be a server similar to the XAMPP web server expect it isn't directly connected to the internet. additional web server code to facilitate sending of keys to Due?? The Due sends a POST request to the Uno, very similar to the one it sends to the XAMPP web server however the key is under the content header. The method that the Due uses, sending a POST request that executes a php file to extract the data from the POST cannot be used, without difficulty??, as PHP can't be run on an Arduino web server. Instead there is an extra header in the POST, *"Content: key"*.

```
String incomingWord = " ", key = "";
int saveNextWord = 0, takeData = 0;
 if (client.available()) {
    char c = client.read();
    Serial.write(c);
    if(c == ' '){
     key = incomingWord;
     incomingWord=" ";
    }else{
     incomingWord = incomingWord + c;
    }
    if((incomingWord=="Content") && (takeData)){
      saveNextWord = 1;
    }
    if(incomingWord=="POST"){
     takeData=1;
    }
}
```

FIGURE 5.10: Handling POST requests in Arduino

When setting up the Uno server, explicitly define what gateway and subnet the router being used has as the defaults *Ethernet.begin(mac)* uses aren't always correct. The server sits open on a certain port, 8081 in this case, and waits for clients to make a connection. Once they have, the server sends an acknowledgement that it has received a connection and starts reading in the request. There it watches out for the word POST so that it knows to store the content and, of course, for the content so it knows what to store. The request from the client is read out a byte at a time so it is necessary for those bytes to be made into strings so specific keywords can be looked out for. If the byte is not a space then it is part of a word so it is added to the subsequent bytes until another space is reached then it is considered a word and compared against. Once the key is taken it is in String format and needs to be turn back into hex and store as the key to be used in further encryptions. DEmo?

5.7 TweetNaCl Library

The TweetNaCl library as it stands in it's original form is not compatible with Arduinos. The C library compiles without errors but the compiler warns that the TweetNaCl method names are undefined and as a result do not perform their tasks. The method simply returns random numbers, it is possibly that it is trying to access some area of memory and simply returns whatever it finds.(ask greg/james). It is not understand why this is the case but it is a simple case of converting the library into C++ syntax. With a header file that has the main methods used in the project and the #defines and a cpp file with the TweetNaCl code. This is added in the same way to the Arduino IDE and in the code an instance of the class is created and methods are accessed with the dot operator.

The keypair, crypto_sign, crypto_box and equivalent opens were used. These are simple to use, abstracted methods that make this library easy to use. For the encryption the method needs the message to be encrypted which needs to have the first 32 bytes be zero, an empty array that needs to be at least the size of the message with the leading zeros, the length of the message, the nonce, arduino public key and the servers private key. This will reveal the temperature data with the signature. To remove the signature, the crypto_sign_open method needs the server secret signature key and the signed cipher array.

Chapter 6

Strength Of Security

6.1 Sign then Encrypt

Encryption stops unauthorised parties from accessing information, assuming authorised parties are the only party with access to a secret key. Signatures prove the message was not altered in transit, integrity, that the message came from a known sender and that sender cannot deny sending the message. These are two distinct operations and the order of these operations matters. If a user was to encrypt a message then sign and send it, as the signature is added to the message it can be stripped off by an attacker and replaced with their own. If the attacker managed to intercept the transmission of public keys in the first place then they can now impersonated the real recipient. Or they could change the message content through a bit banging attack and because there is no hash of the message to be compared to the receiver can't know if the message integrity has failed. In addition the attack can add any signature to the message, encrypting then signing does not prove that the owner of the signature was aware what was in the message as it was encrypted. In this application the message is signed first so the hash is taken of the plaintext message and it can be proved that the signer was aware as to the contents of the message and then it is encrypted.

can protect against the attacks mentioned earlier because...

6.2 Storing data in plaintext

Storing vital data such as passwords and usernames in plain text in a database is an absolutely horrendous idea. This provides one point of failure that if broken means that every password stored is now useless and the accounts are wide open. In some circumstances that initial break might not be so severe, say if that is for forum site where the worst action that could be taken was writing a mean message but users sometimes use the same password for many different sites. Although in this application the temperature data is being store and not passwords the security implications are still there. This application would have that single point of failure that is so deplorable. It is necessary to provide as many layers of security as possible rather than have one hard layer. That being the security of the database. As such this application stores all the temperature data as it arrives, in it's encrypted form. This way the keys will need to be known by the attacker in order to gain the data.

6.3 Public Key Transmission

If the keys are preinstalled they can be used to sign and encrypt new public keys before they are sent. So that the receiver can use the presinstalled keys to prove integrity and authenticity of the new keys. If there are no preinstalled keys then there is no way to prove that the public key being received is indeed the public key sent. At least for a computer. The most secure and safe method to make sure the keys are the same is to have a human user manually inspected the key before sending and after sending and carefully ascertain that the one sent is correct. If there has been a man in the middle, MITM, and either the key has been altered or replaced entirely then the human user can see the difference.

6.4 Bit Flipping

Bit flipping is the process of changing parts of the encrypted message so that it says something else. This is especially potent when the format of the message is know. As the format of the message is know to us, a test can be set that if it succeeds would change a number. If the message had In this project a test was set up to demonstrate the cryptography caused detected that the message had been altered in transit. After a signed message was encrypted it was altered before being decrypted and the signature checked

As the format of the message is known, key areas of the message were altered and the following output notice

The format of the raw plaintext message is 32 leading zeros and 9 bytes of actual information. Where the first two are the temperature.

when we change the 9th and 8th last hex digits in the resulting cipher text....what happens, decrypts fine but signature fails

Also, encrypt 1 and 9999 to see?

6.5 Length of cipher text

Is it possible to gain information about the message from looking at characteristics of the cipher text? In bad cipher text it can be possible to find out the types of message that are sent as the resulting ciphertext may have change in a certain, predictable. For example if the message is a large number in contrast to a low number, there may be noticeable differences. Unfortunately a vulnerability exists in the cryptography chosen but it is a common weakness and one that does not have much consequence. That is the length of the cipher text corresponds with the length of the message. This is offset somewhat by the addititION of the signature so the cipher text is much larger than one would expect if it was know that 9 bytes of hexadecimal were sent. To fully get around this the message is padded out with zeros so that the whole message is always forty one bytes in length. Which does mean the message has a limit of forty one bytes unless the source code is altered. To test how the cipher text changes when sending different messages four encryptions took place. The messages were YES, NO, 1 and 9999.

sending encrypted the one and 9999 yes and no

To encrypt the hex of the word YES, produces

,0xAE ,0x43 ,0xCD ,0xA5 ,0x8E ,0x54 ,0xE9 ,0x30 ,0x59 ,0xB1 ,0xD5 ,0xA5 ,0xBF ,0x24 ,0x9D ,0xEE ,0x69 ,0xDB ,0x37 105, 219, 55

To encrypt the hex of the word NO, produces

,0x2E ,0xBC ,0xCC ,0x6B ,0x9E ,0xDB ,0x29 ,0x64 ,0xF6 ,0x26 ,0x49 ,0xEF ,0xE9 ,0xFA ,0xBD ,0x9C ,0x7E ,0xD1 126, 209

From this message it is possible to tell the number of bytes sent if you know that the encryption method adds 16 extra bytes but other than that this test shows there isn't a discernable difference

To encrypt a message of all zeros ,0x5B ,0x74 ,0x76 ,0x4C ,0xF4 ,0x19 ,0x65 ,0x37 ,0xC4 ,0x53 ,0xAF ,0xB0 ,0xCE ,0x18 ,0x1 ,0x1 ,0x30 ,0x9E

To encrypt a message of a 1 ,0x49 ,0x1B ,0x1E ,0x52 ,0x10 ,0x38 ,0xD7 ,0x38 ,0x30 ,0x65 ,0x71 ,0xBC ,0xEE ,0x65 ,0x3D ,0x6 ,0x31 ,0x9E

To encrypt a message of 99999 ,0xC ,0xC ,0x29 ,0xE2 ,0x4E ,0x81 ,0x67 ,0x5 ,0x78 ,0x49 ,0x8C ,0xA1 ,0x4F ,0x69 ,0x8 ,0xBB ,0x9 ,0xA7 ,0x5D ,0x63 ,0x4D

6.6 nonces

getting the encryption and decryption to have the same nonce

573

Chapter 7

Results

The basic objectives were to sign and encrypt a message in order to stop an attacker from reading he messages and to protect against them being altered in transit. To send that message to a web server and have that data be accessible by a user.

7.1 Basic Objectives

It can be seen that the DS18S20 temperature successfully found the temperature of the room. That that data was given a signature and then encrypted. It was then packaged up and sent as a POST request across the network to a web server that stored the data in a database. Then the user could access that database and view the temperature data. The message could not be read as it was encrypted nor could it be altered as the signature verification process would fail.

7.2 Power Consumption!

As this is to be a low power system that might run on batteries for a good amount of time, it is important to analyse the power consumption of the device. Although saving power wasn't a top priority concern in the creation application, there is more that could be done to save power such as completely shut down modules on the Arduino Due that are never used. The temperature won't be taken every second so between bursts of activity the Due could be put into a sleep mode and be woken up on an interrupt rather than the busy wait implemented in this prototype.

To start the power was recorded as the temperature was being taken with the DS18S20 temperature sensor, in both parasite power mode and regular power mode. It was found that during both operations the power consumption was the exact same at .123A and .613W. This result was not surprising as the power consumption of the temperature sensor is so small compared to the Due that even though it doesn't consume power in parasite mode, the overall consumption of power is barely affected.

To isolate the the power consumed when the Due was only signing and encrypting, the Ethernet shield was left unplugged. This way the effect of the encryption would be seen without the effects of powering an Ethernet shield throughout. The readings upon start up were 0.123A and 0.613W, when the data was being given a signature the readings were .17A and .970W and when the message was encrypted it rose to .20A and 1W.

In addition the power consumption when the Ethernet shield was attached but had no cable plugged was recorded. The power consumed during the signature and encryption process was the same as above but during DCHP the readings were .199A and 0.990W as it tried to gain an IP address from the router. Once the loop iteration was complete, the code went straight into a busy loop which was a tiny drop in power at .198A and 980mW. A busy wait is a waste of power and resources so at this point the Arduino Due should go into a sleep mode and be woken up by an interrupt when it was to record the temperature again.

The next test was with the shield plugged into the router with the Ethernet cable. The power consumption shot up to 0.265a and 1.30W during DHCP at the start of the sketch and there it remained even during the encryption process. Evidently the shield requires such a significant of the power that by comparison the cryptographic library is as good as unnoticeable. There are versions of the Ethernet Shield with an extra module that allows power over Ethernet. This extra module doesn't add a significant increase to the overall cost to the shield but can alleviate battery problems and extend the life of batteries by extracting power from the Ethernet cable that is connected to the router

The final test was to find out if there was a difference in power consumption depending on the length of message sent. Messages of size 20 and size 250 were sent with no appreciable difference in power consumption found.

The voltage through the experiments was the exact same, a steady 4.96V which makes sense as this is round about what USB 2.0 can give out and is well within normal limits for the Arduino Due.

7.3 Additional Objective

7.3.1 Secure transmission of public keys

Expansion of the network was considered with multiple devices communicating directly. This objective was reached by adding an Arduino Uno with another Ethernet shield as a webserver and the Due could pass on data, such as new server public keys, to it. If more nodes where desired then more Uno webservers could be added and the Due would simply go through the list of servers and send a POST request to each to update their keys.

show lots of pictures?

7.3.2 rekeying on arduino

Chapter 8

Critical Evaluation

8.1 Security Concerns

With the way it is set up at the moment the XAMPP server pages, add.php and connect php, and the JSP application that displays the temperature data are accessible through a browser by anyone on the network. Relems could be set up, IP filtering and a an account system set up. To provide a certain level of access to certain users. The way the arduino accesses the new server public key at the moment is simply by sending a get request to the server for it. But an attacker could easily send the same get request and get a public key. Then this key could be used to send mesages to the arduino causing damage at worst or at least the server and arduino might be out of sync and each are using a different set of keys and be unable to read communications from each other.

The public and private key of the server and the public key of the arduino are stored as plaintext in the web application and it might be possible to read them from the WAR files. This is not as much a problem for the public keys, if caught quickly, as it means we can no longer trust that new messages are from an authorised source but if the secret key is leak then all messages sent from the arduino using that key pair are compromised.

The messages are padded with zeros to avoid leaking out the message length. Perphaps it would be better to include random data rather than a set of never changing zeros. Thus the rare case that an attacker notices that the first 32 bytes don't change or worse that they are all the same would be avoided.

The arduino doesn't have access to u random and can't provide good enough random data and therefore can't make good key pairs. Which means that it can't update it's own secret key, for example if the key has been compromised or it is part of a regular renewal service to key security tight.

8.2 Other SubSection

When the public keys are transmitted for the first time, a human user is required to manually inspect the keys, in private, and ensure they are the same before they can be used for data transmission. This is, unfortunately, unavoidable as there is no way to prove that this key that arrived is the one that was sent. The internet is an unsecure platform and it cannot be trusted. There are many things that could happen as it transit the web.

This prototype has used the Ethernet Shield as it was much cheaper than a WiFi shield. If an implementation similiar to the one described in this report was to be implemented then it would be convient to provide connection over WiFi. The product also risks not being adopted as more and more products have wireless capabilities and users see it as the norm and resent cables. This will be especially important if the device uses batteries to power itself. Ethernet shield? Wifi is better but not cheap.

Chapter 9

Conclusion

9.1 SubSection

security measures will never be fully secure so all we can do is make it not worth the attackers while. At the moment there are so many devices that simply have no protection (web security “hack” example pls), so by having even the minimal amount of protection you can dissuade would be attackers because they have plenty of easier targets. Or make it so that the reward of breaking into your system simply is not worth the effort.

The Arduino’s are by no means perfect. If you are having difficultly in a sketch then move variables around, in and out of setup. There were a few occasions where moving a variable out or into setup() fixed the problem. That is absurd.

Appendix A

Appendix Title Here

Write your Appendix content here.

Bibliography

[1] Matt Warman. *50 billion devices online by 2020*. 2012. URL: http://www.telegraph.co.uk/technology/internet/9051590/50-billion-devices-online-by-2020.html.

[2] Nicole Kobie. *Hacking the Internet of Things: from smart cars to toilets*. 2014. URL: http://www.alphr.com/features/389920/hacking-the-internet-of-things-from-smart-cars-to-toilets.

[3] Andy Greenburg. *Hackers Remotely Kill a Jeep on the Highway—With Me in It*. 2015. URL: http://www.wired.com/2015/07/hackers-remotely-kill-jeep-highway/.

[4] Jill Scharr. *Hackers Hijack Prius with Mac Laptop*. 2013. URL: http://www.tomsguide.com/us/hackers-hijack-prius-with-laptop,review-1797.html.

[5] Jeremy Korzeniewski. *Tesla Model S successfully hacked by Zhejiang University team*. 2014. URL: http://www.autoblog.com/2014/07/18/tesla-model-s-successfully-hacked-by-zhejiang-university-team/.