But First just some random websites I came across while looking at IoT security:

<https://www.saaspass.com/about/internet-of-things-authentication-security.html>

This maybe what I saw that said to forget about passwords everywhere.

<https://www.saaspass.com/>

Jeremy I believe talked about Diceware - and here is a website.

<https://www.rempe.us/diceware/#diceware>

Some Definitions:

**Encryption** is the most effective way to achieve data security. To read an **encrypted** file, you must have access to a secret key or password that enables you to decrypt it. Unencrypted data is called plain text ; **encrypted** data is referred to as cipher text.

Crypto - short for cryptography.

Cryptography - the art of writing or solving codes.

Cipher - a secret or disguised way of writing; a code.

Block Ciphers - In cryptography, a **block cipher** is a deterministic algorithm operating on fixed-length groups of bits, called **blocks**, with an unvarying transformation that is specified by a symmetric key.

<https://en.wikipedia.org/wiki/Block_cipher>

Stream Ciphers - A **stream cipher** is a symmetric key **cipher** where plaintext digits are combined with a pseudorandom**cipher** digit **stream** (keystream). In a **stream cipher**each plaintext digit is encrypted one at a time with the corresponding digit of the keystream, to give a digit of the **ciphertext stream**.

<https://en.wikipedia.org/wiki/Stream_cipher>

HASH function -A **hash function** is any **function** that can be used to map data of arbitrary size to data of fixed size. The values returned by a **hash function** are called **hash**values, **hash** codes, **hash** sums, or simply **hashes**

<https://en.wikipedia.org/wiki/Hash_function>

CRC - A **cyclic redundancy check** (**CRC**) is an error-detecting code commonly used in digital networks and storage devices to detect accidental changes to raw data. Blocks of data entering these systems get a short check value attached, based on the remainder of a polynomial division of their contents.

<https://en.wikipedia.org/wiki/Cyclic_redundancy_check>

Several Types of encryption, some better than others. These are some that I found that have Arduino Libraries:

Vigenere Cipher - The **Vigenère cipher** is a method of encrypting alphabetic text by using a series of different Caesar **ciphers** based on the letters of a keyword. It is a simple form of polyalphabetic substitution. The **Vigenère** (French pronunciation: ​[viʒnɛːʁ])**cipher** has been reinvented many times.

<https://en.wikipedia.org/wiki/Vigen%C3%A8re_cipher>

Diffie Hellman is an algorithm used to establish a shared secret between two parties. It is primarily used as a method of exchanging **cryptography** keys for use in symmetric **encryption** algorithms like AES. The algorithm in itself is very simple.

<https://en.wikipedia.org/wiki/Diffie%E2%80%93Hellman_key_exchange>

AES128 - <https://en.wikipedia.org/wiki/Advanced_Encryption_Standard>

SPECK Cipher - **Speck** is a family of lightweight [block ciphers](https://en.wikipedia.org/wiki/Block_cipher) publicly released by the National Security Agency (NSA) in June 2013. Speck has been optimized for performance in software implementations, while its sister algorithm, [Simon](https://en.wikipedia.org/wiki/Simon_(cipher)), has been optimized for hardware implementations. Speck is an add-rotate-xor (ARX) cipher.

Speck supports a variety of block and key sizes. A block is always two words, but the words may be 16, 24, 32, 48 or 64 bits in size. The corresponding key is 2, 3 or 4 words. The round function consists of two rotations, adding the right word to the left word, xoring the key into the left word, then and xoring the left word to the right word.

<https://en.wikipedia.org/wiki/Speck_(cipher)>

SHA - In cryptography, **SHA-1** (**Secure Hash Algorithm 1**) is a cryptographic hash function designed by the United States National Security Agency and is a U.S. Federal Information Processing Standard published by the United States NIST. SHA-1 produces a 160-bit (20-byte) hash value known as a message digest. A SHA-1 hash value is typically rendered as a hexadecimal number, 40 digits long. SHA-1 is no longer considered secure against well-funded opponents. In 2005, cryptanalysts found attacks on SHA-1 suggesting that the algorithm might not be secure enough for ongoing use, and since 2010 many organizations have recommended its replacement by SHA-2 or SHA-3.

<https://en.wikipedia.org/wiki/SHA-1>

(IF SHA-1 is no longer secure why am I listing it, well I did find a library for the Arduino that uses it, and I wanted to post that it may not be secure)

DES - The **Data Encryption Standard** (**DES**) is a symmetric-key block cipher published by the National Institute of Standards and Technology (NIST). **DES** is an implementation of a Feistel Cipher. It uses 16 round Feistel structure. The block size is 64-bit.

<http://www.tutorialspoint.com/cryptography/data_encryption_standard.htm>

Present Cipher - **PRESENT** is a lightweight block cipher, developed by the Orange Labs (France), Ruhr University Bochum (Germany) and the Technical University of Denmark in 2007. PRESENT is designed by Andrey Bogdanov, Lars R. Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew J. B. Robshaw, Yannick Seurin, and C. Vikkelsoe. The algorithm is notable for its compact size (about 2.5 times smaller than AES).

<https://en.wikipedia.org/wiki/PRESENT_(cipher)>

**Grain** is a stream cipher submitted to eSTREAM in 2004 by Martin Hell, Thomas Johansson and Willi Meier. It has been selected for the final eSTREAM portfolio for Profile 2 by the eSTREAM project. Grain is designed primarily for restricted hardware environments. It accepts an 80-bit key and a 64-bit IV. The specifications do not recommended a maximum length of output per (key, iv) pair. A number of potential weaknesses in the cipher have been identified and corrected in Grain 128a which is now the recommended cipher to use for hardware environments providing both 128bit security and authentication.

<https://en.wikipedia.org/wiki/Grain_(cipher)>

Mickey - In cryptography, **Mutual Irregular Clocking KEYstream generator (MICKEY)** is a stream cipher algorithm developed by Steve Babbage and Matthew Dodd. The cipher is designed to be used in hardware platforms with limited resources, and was one of the three ciphers accepted into Profile 2 of the eSTREAM portfolio. The algorithm is not patented and is free for any use.

<https://en.wikipedia.org/wiki/MICKEY>

**Trivium** is a synchronous stream cipher designed to provide a flexible trade-off between speed and gate count in hardware, and reasonably efficient software implementation.

Trivium was submitted to the Profile II (hardware) of the eSTREAM competition by its authors, Christophe De Cannière and Bart Preneel, and has been selected as part of the portfolio for low area hardware ciphers (Profile 2) by the eSTREAM project. It is not patented and has been specified as an International Standard under ISO/IEC 29192-3.[[1]](https://en.wikipedia.org/wiki/Trivium_(cipher)#cite_note-1)

It generates up to 264 bits of output from an 80-bit key and an 80-bit IV. It is the simplest eSTREAM entrant; while it shows remarkable resistance to cryptanalysis for its simplicity and performance, recent attacks leave the security margin looking rather slim.

<https://en.wikipedia.org/wiki/Trivium_(cipher)>

RC4/Spritz - In cryptography, **RC4** (Rivest Cipher 4 also known as **ARC4** or **ARCFOUR** meaning Alleged RC4, see below) is a stream cipher. While remarkable for its simplicity and speed in software, multiple vulnerabilities have been discovered in RC4, rendering it insecure. It is especially vulnerable when the beginning of the output keystream is not discarded, or when nonrandom or related keys are used. Particularly problematic uses of RC4 have led to very insecure protocols such as WEP.

As of 2015, there is speculation that some state cryptologic agencies may possess the capability to break RC4 when used in the TLS protocol.IETF has published RFC 7465 to prohibit the use of RC4 in TLS;Mozilla and Microsoft have issued similar recommendations.

In 2014, Ronald Rivest gave a talk and published a paperon an updated redesign called Spritz

<https://en.wikipedia.org/wiki/RC4#Spritz>

Now on to the Arduino:

Ok, honestly I’m not sure what we are suppose to do with this - but I think it provides a set of tools so you can “authenticate” a message sent between two arduinos. It also looks like it will make randomized passwords, do some encryption and some decryption as well. And it doesn’t seem like it is all that slow on the tiny 8-bit arduino.

I’ve only played with the example sketches, and only enough to know that I don’t really know what they are doing.

<https://hackaday.io/project/8244-arduino-spritz-cipher-library>

<https://github.com/abderraouf-adjal/ArduinoSpritzCipher>

(Appears this may not be secure, see the wiki page, and <https://www.schneier.com/blog/archives/2014/10/spritz_a_new_rc.html> )

New AES Library forum posting on Arduino.cc

<http://forum.arduino.cc/index.php?topic=88890.0>

Davy Landman’s AES Library on Github:

<https://github.com/DavyLandman/AESLib>

ShaifulFlame - Arduino Vigenere Cipher

<https://github.com/ShaifulFlame/ArduinoVigenereCipher>

it have the ability to create a 95 random table key from 4 character which are important due to the fact that the table key need to be always changed periodically, thus programmer can send small key to generate large key set

Arpitchauhan Cryptographic protocols Arduino and PC

<https://github.com/arpitchauhan/cryptographic-protocols-arduino-and-PC>

RSA, AES - Diffie-Hellman Algorithm

ACrypto - collection of open-source code which implements various cryptographic operations on the Arduino family of experimentation boards.

AES128/Block Cipher/CBC Mode/ECB Mode

<https://github.com/kristjanvj/ACrypto>

Arduino Crypto - Uses SPECK Encryption

<https://github.com/Chien-Ning/ArduinoCrypto>

Cryptosuite Library - required by some of the projects listed above.

Newer version - <https://github.com/Phoul/cryptosuite>

Old version - <https://github.com/Cathedrow/Cryptosuite>

Lightweight Cryptography

Block Ciphers - AES/DES/Present

Stream Ciphers - Grain/Mickey/Trivium

(This library doesn’t contain any examples, it appears that you build it to suite your own needs and hardware)

<https://github.com/efDidymos/LightweightCryptography>

Crypto Arduino Library - another early crypto library (last updated it looks like 7 years ago) Probably not secure using SHA1 -

<https://github.com/leoserra/crypto-arduino-library>

Arduino Rokkit Hash - Arduino port of Paul Hsieh’s SuperFastHash - Great alternative for CRC/hashing applications.

<https://github.com/mrbio/Arduino-rokkit-hash>

And more useful libraries on github:

<https://rweather.github.io/arduinolibs/index.html>

The two libraries I would like to explore a little more the Arduino Vigenere Cipher

<https://github.com/ShaifulFlame/ArduinoVigenereCipher>

And The Cryptographic Protocols Arduino and PC library

Specifically I’d like to look at the digital signature in this library.

<https://github.com/arpitchauhan/cryptographic-protocols-arduino-and-PC>

Time permitting I’d like to look at the SPECK library as well.

So I finally had some time today to look at my choices for this project (July 12, 2016) - and it turns out that the cryptographic-protocols-arduino-and-PC library compiles with a lot of errors, if I had more time I’d look into why, but time is still short.

I also looked at the SPECK library - and honestly I just don’t understand it. It seems to work only with the values that are already coded in the library - making any changes seems to break it. And I didn’t see anything that explained what was needed to make it work.

So that leaves us with the Arduino Vigenere Cipher - I think this will work quite well for what we are doing. Vigenere Ciphers according to WIKI are secure - but I don’t know if this one is as. More testing would need to be made to see if people could work out how it works. For now thou, it seems pretty good.

The library (which really isn’t a library, it’s a sketch, with function calls to make it work) uses what they call a MasterKey - it’s 4 alphanumeric/symbol digits, it also uses a 4 digit seed value (With I am calling a public key - or a random key). This seed value appears to work with alphanumerics and with a digit that is larger than 4 digits.

The way I plan on using this is somewhat like a key exchange with public/random key and private keys.

I’ve already put a link to ShaifulFlame’s project on github - but here it is again:

<https://github.com/ShaifulFlame/ArduinoVigenereCipher>

I forked the project, and uploaded it to our Automation Technology Club Github, created a new branch for our changes, and that is here:

<https://github.com/automation-technology-club/ArduinoVigenereCipher/tree/ourchanges>

One thing I did - was to create a new varable called “PassKey” - this will change again, as I realized it’s going to be more of a randomKey.

So the idea is, we will have an app on our phone/tablet that will be capable of generating the Vigenere Cipher, it will also generate a random 4 digit key. It will encrypt whatever data we want to send to the Arduino, and send the random/public key in the clear.

Remember, the Arduino just doesn’t have the power to run a HTTPS website, so everything you send to it is in the clear.

The Arduino will use the public Key and decrypt the message, and hopefully perform some action.

Both the app and the Arduino will have the Master/Private Key hard coded - this code change thou as needs change.

For the Demo I will use an Arduino UNO with a ethernet shield running a slightly modified version of the web server that has been demonstrated before.

I will have a 2nd Arduino hooked up to generate the key, and be using a web-browser (Firefox or Chrome) to have the 1st Arduino do something.

In the case of the Demo I will setup a couple of LEDs to be turned on or off.

We will also send it information it doesn’t know what to do with, or a bad key.

So, tomorrow afternoon, I will set out to get the web server setup and working.

July 13, 2016 - I have a working Demo, memory is getting tight on the web server side of things, but is still manageable.

What I have does work, it randomizes the “message” each time you want to do something - but I ran into what I see as a serious issue, and one that I’m not quite sure how to work around.

The whole idea was to make it harder for people to turn on or off, open/close, etc

If someone is watching the stream of data to the website, yes the message is encrypted, no one will be able to tell what you are doing. But here is the problem, they can just copy the encrypted text and send it back to server.

I’m thinking for this to be even more secure and better, maybe a couple of RTCs and make a CRC check based on Date/Time - or even just Date. The CRC would be the 1st thing to check if it fails, then your done, no reason to decode the rest of the message.

I think that is something I might work on for next weeks meeting, time permitting.

At this point however, while it appears to be more secure, it’s probably no more secure than sending in the clear.

July 20 - 21, 2016.

New code using a RTC (real time clock) has been added to github. The new code adds something as a time stamp to the encrypted string. The time stamp itself is not encrypted, but is sent in a way that just by looking you may not know what it is. I think if you watch the stream for any length of time you could figure out what the addition is.

A ARDUINO MEGA IS REQUIRED FOR THE RTC webserver code

The new string looks something like this:

ilox{\V\_372043

^^^^^^^

This is the public key and encrypted message

372043 - is the date stamp.

You might look at this and go hum…

It’s two parts - 1st part is hours and minutes (37) or 13:24 - now that you know what time 37 represents, you can see how I got it - just add the two numbers together. I given that there are 24 hours to a day (0 to 23), and 60 minutes to a hour (0 to 59) it would be very hard to decide what 37 was unless you knew either the hour or the minute the message was sent.

The 2nd part is the month, day, year-2043, this one might be a little easier to get, since it will be the same number for 24 hours. 2043 - works out to 7/20/2016 (7+20+2016).

Today is 7/21/2016 - so today’s number should be 2044.

So, now that you understand how it works, it would be easy to fake a valid time/date but you’d still need a good message.

Potential Problem: month/day/year is pretty simple to figure out - it is sent in clear. And a better way to do this maybe to add the time with the month day year (in the example 372043 would then become 2080). I don’t know if this would solve it or not.

Another possible way to solve it, is have the 372043 encrypted and sent as part of the message.

Some other problems that I noticed:

Some of the special characters the web server doesn’t like (and tokenizes them) - this is a problem. And at this time I don’t have a good fix. My possible solution would be to have the program that generates the key check to see if any of the problem characters are in the strings, if so regenerate a encrypted message before it is sent.

Things that I know are causing problems (space) “

This list probably will get bigger over time.

One idea I had was to try to make the cypher have a smaller table, but that ended up causing the messages to not be decrypted correctly.