AUTHENICATION AND ENCRYPTION

Background

This paper is the final part of the “methods for securing IoT systems”. The first part was centered around “Trust Anchors”. This final part is concerned with the finer detail of authentication and encryption.

In our previous paper “Trust Anchors” we discussed what they are and why they are needed. We established that they are the essential foundation to any trusted communication between a number of parties. We also established that they can ONLY be implemented in specifically designed hardware, not general purpose computing used in IoT, mobile devices, laptops and desktop computers.

Once we have two trust anchors (or one for each point on our communication channel we can begin communication. However, we must make the assumption that all communications we conduct are not secure and are recorded and spied upon. Are we being paranoid? Maybe we are, but that does not mean that it’s not real.

Specifically, the assumptions we must make are:

(1) The communication channel is not secure.
(2) There is interference on the communication channel.
(3) The communication channel is not instantaneous.
What do we mean by these assumptions?

Insecure communication channels

When we sit in a cafe and discuss intimate details with our partner we run the risk of someone overhearing our conversation. We can minimise this by talking in low voices and sitting close to each other. We also can observe the people around us and modify our conversation depending upon our observations.
However, if we wanted to have intimate conversations across a crowded cafe, but we could not sit in close proximity, we run the risk that everyone will hear. This is analogous to sending data across the Internet. Everyone, everywhere has the ability to listen to your Internet traffic. We call this an Insecure communication channel. When we can sit close to one-another and talk in low voices to avoid eavesdroppers, we call this a one-to-one or one-to-many secure communications channel.

Communications Interference

Back in our crowded cafe we are having a detailed conversation with a business partner. We sit close and listen intently to the conversation. The human brain is exceptional at tuning out unwanted interference from other conversations and figuring out what our business partner is saying. However, if someone in the cafe starts playing loud music very close to us, even the Human brain cannot decipher the conversation, because it is swamped by noise. When we send information across the Internet, our data can be affected and altered either intentionally or unintentionally. Take the case of a Skype video call. These are usually very clear and usable, but with large volumes of interference (either intentional or unintentional) the video will brake-up. We call this channel interference.

Communications delay

In our cafe we talk to our partner in real time. As our lips move their eyes detect this and associate the sounds with the expressions and movements of our lips. If we watch a video on the

Internet where the audio and video are out of sync it is very difficult to watch (we either listen to the audio only or watch the video only).

If we wanted to talk to our partner from a distance (one town to another) in the days before the telephone, we would use a letter. This form of communication is fine to convey emotions, instructions and descriptions. But because it is not real- time it is easy to copy, alter, forge and generally tamper with. So it is with the Internet. Data sent in the form of email, twitter, snapchat, facebook etc are not real-time and thus can be intercepted, altered and forwarded.

A phenomenon called “doxing” has started to appear. This is where someone sends a tweet, or message to the Internet for others to comment upon. However, before the message appears, a malicious hacker – called the “Man-in-the-middle”, intercepts the message, alters it and then sends the message on. It thus it appears that the original person has sent a tweet or message that they did not send. Many people call this “fake news”, and it is becoming a real issue to reliably know what the original sender intended to send.

Authentication

In the time before electronic communications, important documents were sealed with a Wax seal. This performed two functions: (1) anti-tamper detection and (2) authentication.

The Wax seal provided a method to detect if a tamper had occurred. The device that provided the seal was a carefully guarded device – either a ring or a stamp. In the case of the Crown Royal Seals it was an offense punishable by death to copy a Royal Seal. In the case of individuals, a secret Ring was used.

Any attempt to open the document broke the wax seal, and it was obvious that the contents may have been tampered with. Authentication was performed because if the seal was not broken the contents were guaranteed to be the wishes of the sender.

In 2017 we provide authentication by using a digital seal or signature. This could be a one-time-only used piece of information that both parties know that has been mathematically appended to the message.

Note that it is not just a case of sharing a piece of information between both parties, then appending that information to the message, because in the days of digital computers the message can be separated from the information, altered and then re-appended.

We need a method of integrating the message with the information.

SECURE HASH ALGORITHM STANDARD (SHA)

One method for integrating the message with the information is called a SHA. This is a mathematical process where it is relatively easy for someone to compute the signature from the message, but it is exceptionally difficult to reverse-engineer the message from the signature.

Researchers over the years have looked for mathematical functions that can perform this and have standardized a method into the Secure Hash Standard. This is approved by the US (FIPS 180), UK and many governments world- wide as a guaranteed standard for digital signatures.

In order to use the SHA standard there are some subtleties we must understand. We must take account of the assumptions we made above. Specifically messages in the MAN-IN-THE-MIDDLE-ATTACK, can be intercepted, altered and resent to the receiver. In this instance it is no good sending the message and a SHA of the message, because the MAN-IN-THE-MIDDLE can simply copy the message alter it, sign it using the SHA algorithm and send it on.

In this instance the receiver gets a message and a correct signature for that message but has no way of knowing if the message has been altered.

So in order to send a correct authenticated message that both the sender and receiver can rely upon we need to perform the following steps:

SENDER
(1) take the message and append some shared information only known to the sender and receiver.
(2) compute the SHA of the message and shared information.
(3) send the message + SHA (NOTE DO NOT SEND THE SHARED INFORMATION!)

RECIEVER
(1) receive the message and SHA
(2) append to the message the shared information
(3) compute the SHA of the received message and shared information
(4) compare the received SHA and the computed SHA
(5) if the 2 SHA's are the same the message is authentic.

This will work because, when the MAN-IN-THE-MIDDLE tries to alter the message, compute the SHA and sent it on, they cannot append the shared information, which they do not know.

It is a subtlety to the use of the SHA standard, but if ignored invalidates the process.

The SHA standard is very useful and is widely used in digital communications. It has a useful property of compression. It allows a large quantity of data to be reduced to a very small signature in a unique fashion. For example, this document could be digitally signed by performing the SHA on the document. We could reduce this document to a 256bit signature. This signature would be a unique reduction and a small change in the underlying document will map to a different signature.

However, for a small quantity of data less than the 256bit signature SHA is not effective. For this type of data we can use a different technique.

SYMMECTRICAL AUTHENTICATION AND AES256

Symmetrical authentication relies on the shared information and a mathematical process. The idea is to develop a mathematical process that combines the data and the shared information in such a way that the original data and shared information cannot be separated from the result of the mathematical process. Given the shared information the original data can be separated from the result by the reverse mathematical process.

Let's call our message the **Plain Text** and the shared information the **Key**. We can call the mathematical process of taking the **Plain Text** and the **Key** and producing a result, the **Encrypt** process and we can call the result of the process the **Cipher Text**. Finally, the reverse mathematical process is called **Decrypt**. A typical Symmetrical algorithm is AES256 (American Encryption Standard) and this is the default "gold" standard, for world-wide communications.

So we now have our process of symmetrical authentication as the following:

Plain Text + Key → Encrypt = Cipher Text

and the reverse is:

Cipher Text + Key → Decrypt = Plain Text

So in our authentication process we perform the following:

SENDER

(1) Encrypt message using **Key** and **Plain Text → Cipher Text**

(2) Send **Plain Text** and **Cipher Text**

RECIEVER

(1) Receive **Plain Text** and **Cipher Text**

(2) Decrypt **Cipher Text** with **Key → Plain Text**

(3) Compare received message with decrypted **Plain Text**

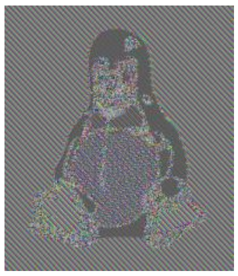
(4) If they are the same – the message has been authenticated

This works because our Man-In-The-Middle cannot generate the Plain Text from the message because they do not know the key. If we use a symmetrical algorithm such as AES256 and our key length is 256bits, we can be very certain that our message is authentic.

The big problem with the above is that the man-in-the-middle could copy the message and resend it later. For example, if a set of messages were to turn a traffic light Red, Green or Amber and these were copied it would be trivial for the man-in-the-middle to replace the message sent with another one. The **Plain Text** and **Cipher Text** would be correct, just in the wrong time sequence.

SALT AND TIME

So the problem with our Symmetrical Algorithm is that given some **Plain Text** and a **Key** our AES algorithm produces EXACTLY the same **Cipher Text**. This is a problem because we can associate the **Cipher Text** with the **Plain Text** with great ease.



These are two images of Tux – the first is a normal jpeg. The second has been encoded using AES256 with a key. Note how the image can be discerned, because the symmetrical algorithm produces exactly the same result for the same input. So in an image with large areas of the same color pixels these will compute to the same (but encrypted) pixels.

Generally to disrupt the association between **Cipher Text** with the **Plain Text** we use some additional information in the message generally called **Salt** - (I guess because it adds “flavour” to the symmetrical algorithm).

One version of the **Salt** is to use a technique where we add a number that is in a sequence. This number increments after each message is sent and repeats after a significant number of messages. We call the number a **nonce** – after **n**umber **once**. Another version of the **Salt** is to use time. This works because the man-in-the-middle cannot capture the messages and send them at different times – because the date and time is in the message. Time is very useful in the authentication of our messages, because it gives a time limit on the amount of analysing an adversary can perform on our message, before it becomes invalid. However, we do need time, everywhere. Fortunately, most networks have time and GPS is available world-wide.

There are a number of different methods for disrupting the **Cipher Text** and **Plain Text** association. These are centered around a chain of blocks of the encryption process, where the result of the previous encryption is used as the **Salt**
for the next encryption. The obvious disadvantage for some of these is that if one of the encryptions is broken in transit then subsequent decryption may be unreadable. However, there are methods to alleviate this problem, but it is beyond the scope of this article.

BRUTE FORCE ATTACKS

One problem using a published algorithm like AES is the possibility of a Brute Force Attack – especially by Quantum Computers. If the key length is trivial at 64bit, the average laptop would be able to try every key with the **Cipher Text** until it finds the key. This might take an hour, but it is worth it because the man-in-the-middle now has the key and can decrypt any messages.

Even with 128bit key lengths, powerful laptops can potentially break the key in a few hours – to a few days. If a WiFi password is constantly being used the it is trivial to gather large quantities of data and try to brute-force-attack them for the key. Even if no Plain Text is being sent (because the data is encrypted as well as authenticated) the attacker knows the format of the IP header and thus can look for known data structures.

It is suggested that a Quantum Computer of 256 Qbits could brute force attack 256bit encryption instantaneously. If this were true then we need other methods for securing our data, but no one has yet published any work on this. However, this is not surprising, as it would render all commercial encryption on the Internet as useless.

Any key under 56bits of length for a standard algorithm is generally regarded as trivial to crack using brute force.

Today we generally use AES at 128 or 256 bits because it gives a wide factor for brute force attacks because it takes account for data encrypted to be safe for the next 5-10 years, after which the data will become superseded.

So what is the answer:
(1) use longer key lengths (256 bit for example)
(2) use a custom unpublished algorithm
(3) only use the **Key** once
(4) compress the underlying data before encryption
(5) use binary only transfers with no standard binary to ASCII mapping
(6) use quantum encryption

For most users, companies and governments AES256 is the “Gold” standard that should be used. However, some users have specific requirements that make using AES256 difficult. For example, export restrictions, short data lengths, private communication systems using non-standard equipment, are all valid reasons for not using a published standard.

Other algorithms are available – for example Twofish, designed by Bruce Schniener et al, for example is available as a book with a “C” code implementation. This was one of the finalists of the AES competition, and lost out to Rinjial which became the AES256 standard. All of the finalists generally conform to the Fiestal method of encryption. Horst Fiestal developed this method in the late 70's as a secure method for encrypting and decrypting data. What is important is that he did not specify what mathematical function should be performed on the data, but merely the order and the structure of the computations. This enables a whole family of Ciphers that are incompatible with each other, but similar in operation and thus security.

BENEFITS OF CUSTOM ALGORITHMS

There is one main benefit from a custom algorithm – they are resistant to Brute Force Attacks (BFA). This is because in a traditional BFA the algorithm is repeatedly operated on the sure data with a different key until the data is revealed. If the algorithm is a custom version, then there is no way of knowing how the **Plain Text** maps to the **Cipher Text** and thus no way of performing a BFA. Even for short key lengths of 64 bits the custom algorithm is deemed completely safe.

DISADVANTAGES OF CUSTOM ALGORITHMS

There are 2 main disadvantages of a custom algorithm. The first is that the algorithm has not had the benefit of public scrutiny and therefore might have weaknesses that are not known to the creator. However, if the algorithm is based upon the work by Horst Fiestal then the cryptanalysis of the Horst cipher has been well documented and whilst individual implementations may be stronger than others, there is no doubt that after a certain number of "rounds" of the function they all become impossible to analyse. Over 12 is deemed safe and beyond 16 rounds there is no published work on 'cracking' this type of cypher.

The second main disadvantage, and the most problematic, is in the implementation. AES256 is available on every computer worldwide, it is well published and uses a 256bit key length to provide security. Software versions of AES256 can be analysed for the underlying structure, but will reveal nothing new over the well published details of the algorithm. However, in a private algorithm, ANY software implementation provides the possibility to reverse engineer the details of the structure of the Cipher. This leads to the algorithm becoming public and opens the possibility of a Brute Force Attack.

So we have 2 possibilities for using a custom algorithm, use a long key length for our custom algorithm, and take the risk that it will be found out. Or only implement the algorithm in hardware so that it could never be analysed.

USING A KEY ONLY ONCE

This technique is often called a “one- time pad” because it comes from the early days of cryptography. It works this way: A user writes down on a pad of paper a random key on each page. They then copy the pad of paper so they have 2 unique copies. They give one pad to the person they need to communicate with (we will gloss over how the pad gets there!). When they want to communicate they use the code on the top of the pad for their communication. After the communication has finished (or on the stroke of midnight) the tear off the top sheet and use the next key. This was used very successfully in WWII by the Germans with the Enigma Machines.

The one- time pad is very effective as it only gives the man-in-the-middle the chance to decode one message at a time with one key. To decode the message using a brute force attack is very processor intensive and thus to do this for every message is usually not worth the effort, because the information is “stale” by the time it is decoded.

DATA COMPRESSION

In a brute force attack the encrypted data is repeatedly tried with a different key in order to decode it. How do we know if the data we have decoded is correct? The answer is obvious if the original file was a word document containing English characters. However, if the underlying data was a file of random numbers, there is no way of knowing which “decryption” was the correct one.

This is called stenography and it is concerned with hiding data within other data. If we compress our word document using a compression algorithm only we know then a brute force attack is even more difficult.

Bruce Scheiner suggests always “zipping” a document before encryption, as it makes a BFA almost impossible – good advice!

QUANTUM ATTACK

Quantum computers are very good at finding the minimum energy in a field of energy. For example, Noise Cancellation is a classic Least Mean Squared minimum energy calculation problem. In a traditional Noise Canceller - for example, a un-born baby heart monitor, where the microphone picks up the mothers heart and the baby's heart, the LSA algorithm is used to adjust a filter to remove the mothers heartbeat. This is used on high-end headphones and in some cars for example. The problem with the LSA algorithm is that in the field of “hills” that is the sound map from the microphone there can be many local minimum points, and it is difficult for the algorithm to know the global minimum. This manifests its self as a sub-optimal solution where there is a bit more noise or hiss on the headphones, or the baby’s heartbeat is not as clear as it could be. Quantum computers can look at the field as a whole and find the global minimum instantaneously – which is the optimal solution.

So programming a Quantum Computer is one of mapping the energy field to the Computer so that it can find the minimum energy state. However, this is very complex and people have been looking at ways of making this mapping automated. Quantum Computers belong to the larger set of Lerner-Classifier systems, where the system (Quantum Computer, Neural Network, Genetic Algorithm etc) is placed in Learning mode and given a series of inputs and known outputs. Once the system has been given enough “examples” it is placed in Classifying mode, and then given an unknown input will successfully classify it to a known output.

In cryptography the energy field might be an encrypted message, and the minimum energy might be the key or the **Plain Text.** The problem of finding the minimum energy and hence the key is then one of mapping the encrypted data to the quantum computer in such a way that it can solve for the minimum. If there is no known **Plain Text,** then there is no mapping of encrypted data and the quantum computer cannot be used to search for a minimum. Even in the“learning - classifying” mode of a quantum computer, if the **Plain Text** is not known then the Quantum computer cannot be trained.

So in order for a successful Quantum Computer attack, a **Plain Text** attack is performed where a lots of **Plain Text** and its equivalent **Cypher Text** is used - (IP Header attack is a good example).

QUANTUM ENCRYPTION

So if we are worried about a brute force attack on our encryption system, could we use Quantum Encryption? Well possibly and this is an area of intense activity at the moment. The D-Wave quantum computer uses a process where the “programming” of the computer involves setting a series of parameters that control how the Quantum Computer behaves. Given this set of parameters the Quantum Computer will produce an answer. The encryption process would be to choose the parameters for the Plain Text data set and record the answer. The decryption process would be to choose the parameters for the Cipher Text and record the answer.

Well, it’s not quite that simple. The computer works because of the uncertainty surrounding the state of particles in a quantum state inside the computer. Its output is a set of answers with varying probability, but never one single answer with 100%. This is great for problems that are trying to find the minimum and can live with a degree of uncertainty (noise cancellers for example), or for problems where there is more than one answer (like which stocks to purchase on the stock market to make most money). It is not good for cryptography, because encryption and decryption rely upon an exact answer - even 1 bit of uncertainty must lead to a completely different answer.

However, I am sure that over the next few years there will be much research into these problems and solutions offered. Real Quantum Computing and in particular Quantum Cryptography is still in its infancy and currently requires room-sized hardware.

Experts have predicted that Quantum Computer Farms will offer immense computing power on-demand over the Internet, and your average mobile device will have access to this cloud computing resource.

SUMMARY

In this article we outlined the reasons for Authentication and Encryption, their uses and limitations. These terms are widely used but seldom understood, and how they are implemented is even less well practiced.

Most software programmers' experience of authentication and encryption in the use of standard libraries available in Windows, Linux and iOS. They seldom understand the fundamental aspects of the tools they are using and hence the large quantity of software products sold everyday, have security flaws.

Security patches for all major operating systems are a matter of daily life and everyone has witnessed the messages that appear on their screens warning users to update their computer to counter a software vulnerability.

The problem stems from the desire to make "cool" products quickly and get them out to users – for profit, before adequate security testing and vetting can be performed. Although this is, at most, an inconvenience for users and they accept the updates and the anti-virus software as a part of their daily lives, there is a more pernicious threat lurking – IoT.

The Internet of Things is happening now, but instead of a vulnerability in your desktop computer requiring a security patch we have the vulnerability in IoT devices controlling the brakes and steering on your car and just about anything complex with some level of automation.

A software update being applied to the ABS braking controller on your vehicle just as the user is approaching a busy junction is a problem waiting to happen.

When the number of fatalities has risen to an unacceptable level in Autonomous Vehicles, Insurance Companies, Governments and Users might insist that industry priorities security over "cool".

--o--