Securing and Integrating the IoT with a Smart Home Router

Fergus W. Leahy

School of Computing Science
Sir Alwyn Williams Building
University of Glasgow
G12 8QQ

Masters project proposal

16/12/2013

Contents

1 Introduction

> *"The most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it."*
>
> – Mark Weiser, *The Computer for the Twenty-First Century, 1991*

The modern home is becoming increasingly filled with a variety of *connected* devices (laptops, tablets, phones, set-top boxes etc.), providing a myriad of different services to users within the home. On top of this, with the introduction of smart phones and wearable devices, we too are starting to carry around our own personal network of devices everywhere we go, brushing past many others in our daily lives at home, work and on the street. Although all connected to the Internet, these devices are often encapsulated within their own environment and ecosystem, unable to interconnect, creating a fractured and often complex user experience.

Making matters more interesting, the Internet of Things paradigm is once again becoming a field of great interest due to the advent of cheap, low-power wireless embedded devices [1]. However, not much consideration has been made for how these Things should be integrated into the existing home network, with many approaches opting to simply bridge the device to the cloud ([6], [8], [20]), with obvious concerns for security, privacy and up-time.

As these devices enter our homes and pockets, bringing with them their own ecosystems, the user is faced with the increasingly difficult burden of managing them and their ecosystems [11], [10]. Due to the sheer number and diversity of these devices, many of which will provide overlapping services and functionality, problems arise with respect to how these devices cooperate, as well as how to ensure that the user's network and information stays secure against new and unanticipated threats.

In order for these multiple layered networks of devices to truly fade away into the fabric of our everyday lives, a platform and relevant protocols need to be engineered to not only support this heterogeneous network securely, but also aid the user in managing both the network and the privacy of their information.

The Homework home router platform was created to address these issues. Rather than assume every user is a network administrator, the project investigated the needs and abilities of the average user in order to propose the future of home networking, re-inventing the protocols, models and architectures to truly suit the home environment. This re-invention of the home router allows a user to easily install, manage and use their home network, without the need of a Cisco qualification.

In regards to the Internet of Things development, previous work demonstrated that it was in need of a suitable protocol in order to meet the specific needs of a network of Things [16]. Thus, a new protocol was designed and implemented, which could not only run on the most constrained battery-powered devices (8MHz), but it could also efficiently scale to support hundreds of Things within the same network.

2 Statement of Problem

Whilst there have been many attempts, both past and recent, to create the perfect Internet of Things network for the home, all still exhibit several issues. This section will first present a typical user scenario to provide a context for this problem, then discuss several of the issues found in existing work, both our previous work and others', which provide the motivation for this project.

2.1 A Scenario

In order to establish the context of the problem, a typical user scenario has been created to demonstrate a typical novice user purchasing a new Thing and attempting to integrate it into their network securely with minimal knowledge and effort.

Bob buys a new Thing, a motion sensor, from a shop and brings it home. He wants to connect it to his currently existing home network. To do so, Bob turns on the Thing and presses the connect button on both the Thing and his home router. Using his tablet/PC Bob is able to view the newly available Things in the network that he can connect to, in which he can see his new Thing. Bob is able to view various information about the Thing, including it's type, manufacturer, functions etc. Bob selects the Thing he wants to connect to the network and within a minute the new thing is now part of the network and he can then customise how he wants to use it in the network.

2.2 The Problem

The Internet of Things protocol created in [16] proved to be a successful proof-of-concept; however, in order for it to be considered for deployment and integration into existing homes, two main issues need to first be addressed.

2.2.1 Security

The initial design of the IoT protocol didn't consider security due to time constraints; however, because not only can such rich and sensitive data can be gathered about the user and their home from Things, but the closed loop of control can be affected by injection; therefore it becomes extremely important to not only protect the data from unauthorised access, but also protect the network itself. Presented below is a summary of the typical attacks targeted towards WSNs, and in this case an IoT.

1. **Eavesdropping** - An attacker observes the unencrypted traffic between the various devices in the network, and uses this information for illegitimate purposes. This needs to be prevented due to the richness, coverage and sensitivity of the data captured

within an IoT, which could be used to perform serious physical attacks, spying, as well as have other serious consequences. Therefore, it's necessary to encrypt all traffic where possible, so that only legitimate participants may observe the information traversing the network.

2. **Unsanctioned participants** - An attacker infiltrates the network masquerading as a legitimate participant, enabling it to spoof traffic, such as events and commands, as well as observe other traffic. Such an attack would enable an attacker to spy on and manipulate the network of Things, which could lead to various damaging consequences, ranging from energy waste, to more serious home security issues such security locks being disabled enabling other possible crimes.

3. **Node capture** - Due to the placement of Things, sometimes outwith the protective confines of a home or building, they can become vulnerable to an attacker capturing them. Once captured, it may be possible to extract encryption keys, illegally modify the functionality and also replace nodes with maliciously modified ones.

4. **Denial of service** -

Demonstrated in the scenario, a user may want to purchase any kind of compatible Thing from a shop and then want bring it into their home network with minimal effort. The Thing itself would have no prior knowledge of the network and needs a secure mechanism to enter the network without compromising the rest of the network or exposing its own data to unauthorised users.

2.2.2 Integration

The current implementation exists as a standalone library with several demo applications. Integration of the IoT protocol into a user-friendly platform is necessary to harness the full power of a network of Things. The integrated platform would then be able to discover and connect to available Things, subscribe and log events from the sensors, and using user customised rules, use automata to detect if these rules are met and then perform actions by publishing commands to actuators in the network.

2.3 Intranet of Things vs Internet of Things

As described earlier in section 1, many previous deployments of Internet of Things networks have taken a cloud first approach [6, 8]. Whilst this yields certain benefits, such as easy external access and integration with other services [4, 9], it also poses several questions regards data security, privacy and up-time. For this project, the focus will be on developing a home first platform, in which all Things communication will be kept local, with no cloud processing involved; thus a more suitable name, the Intranet of Things, will be used.

2.4 Project Outcomes

Outcomes of the project:

- An extended IoT protocol with sufficient security to prevent eavesdropping and unsanctioned devices.
- An Extended home information platform (Homework), with the IoT controller role implemented, enabling capturing of IoT events from sensors and creation of commands for actuators.
- Use of Homework's automata to implement closed loop control of Things in the home, subscribing to sensor events, processing rules and publishing to actuators to perform actions.

2.5 Structure

The rest of this proposal is structured in the following way. Section 3 surveys previous work on the field of wireless sensor networks security, home networks and Internet of Things systems and protocols. Section 4 takes what has been learnt from the literature survey and proposes a solution to solve the problem described above. Lastly, section 5 discusses the work plan outlined for the implementation and evaluation part of the project, for the second half of the year.

3 Literature Review

This section discusses previous work on Internet of Things protocols, WSN security and home networking which provides the motivation and possibly aids in solving the previously mentioned issues.

3.1 State-of-the-art IoT Protocols

Over the last 20 years, the Internet of Things has developed and evolved continuously, from 20 years ago where it was the concept of using barcodes, and later RFID[5], to track items in a warehouse, to today where our home appliances are filled with "smart" electronics, able to sense, react and notify the user of events, such as a washing machine finishing a load of laundry. Whilst the Internet of Things has been around for quite some time, under various guises, it's yet to become commercially successful and see wide-spread deployment; many believe that 2013 is the year that the Internet of Things will truly take off and become ubiquitous in our daily lives[1].

The rest of this section will discuss various different attempts made to create Internet of Things networks in the home.

3.1.1 Smart Things

Launched in 2012, the SmartThings platform aims to allow the user to turn any ordinary object in the home into a "Smart" object by giving it the ability to connect to the Internet. The platform consists of a central hub connected to the Internet, containing a low-power Zigbee radio, from which it connects to an array of SmartThings accessories as well as many pre-existing third party Zigbee devices. Some of these accessories include motion detectors, moisture sensors, vibration detectors, power-plug switches as well as many others. The user can then view and interact with their Smart Things through a cloud service via an Internet-connected PC or a smartphone, allowing them to either control the devices directly or set-up rules/schedules for devices.

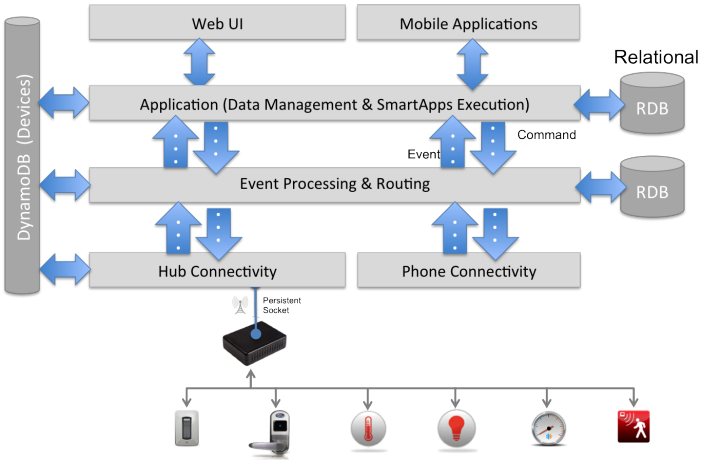

Figure 1: Smart Things - Cloud approach[1]

As shown in figure 1, Smart Things takes a cloud approach, offloading all processing to the cloud, leaving the hub to just act as a gateway and translator between the Things and the cloud. By utilising the cloud in this way, it enables the hub to be a very simple and low-power device, reducing both the acquisition and running costs for the user. It also allows the user to access their network of Things from anywhere, at any time, which would otherwise be difficult to do in a local approach.

However, the cloud approach also has several drawbacks. By offloading the entire processing needs of the network, as shown above, the network of Things becomes vulnerable to failure if either the network fails or is unresponsive, or if the cloud service fails or goes down for maintenance. Another issue is the privacy of the user's data; how safe is the data in the cloud and how secure are the services they make available for users to interact with

[1]Image from Smart Things developer support: `https://support.smartthings.com/entries/21603009-Device-Types-Capabilities-Attributes`

their devices? There have been many examples of online services that have been attacked, leaking user data and/or shutting down for extended periods of time [7, 2, 3].

Talk about integration of two home networks.

3.1.2 CoRE CoAP HIP-DEX

3.1.3 KNoT

3.2 Home networking

Discuss Homework router and cache db

3.3 WSN Security - Symmetric Cryptography

Substantial research has been carried out to secure WSNs, to ensure data within the network can't be compromised, enabling attackers to eavesdrop and/or masquerade as sanctioned participants, and introduce bogus data that can affect the control loop activity, or leak sensitive data.

Symmetric key cryptography is a class of cryptographic algorithms which enable two participants to exchange data in secret over a public channel using a shared key, known by both participants. Both encryption and decryption use the same key. One of the key benefits to symmetric algorithms is that it's relatively inexpensive, both in time and space, to encrypt and decrypt data, which makes this ideal for use in a WSN where resources are extremely constrained.

Symmetric algorithms fall into two main categories, stream based ciphers, where data is encrypted by combining it with a pseudorandom data stream (one time pad), and block based ciphers, where data is segmented into blocks and operated on by a set of functions, often with the previous cipher block feeding into the next input.

In the case of WSN, block based ciphers have seen exclusive use. However, one issue with a block based cipher is that if two identical blocks are encrypted, both will return equal encrypted blocks, thus a passive adversary could extract partial information about the plaintext sent (semantic security). To solve this and maintain semantic security, an additional security primitive, an initialisation vector, can be used to ensure that this cannot occur. An initialisation vector provides a unique value which is used in the encryption of the first block, ensuring no two identical blocks return the same cipher text block; after the first block, the initialisation vector becomes the previous ciphertext block.

The rest of this section discusses several attempts to secure WSNs using symmetric key cryptography for both the TinyOS and Contiki WSN operating systems, mentioning both

the benefits and some drawbacks apparent in each.

3.3.1 TinySec

TinySec is a fully functional symmetric security link layer component created for the wireless sensor network operating system, TinyOS. It was the first fully implemented solution for WSNs and was created to address the security worries of running a WSN and transmitting private sensor data in the clear. Unlike conventional security protocol implementations which can afford significant time and space overheads, such as 16-32 bytes for security per packet, WSN typically run on extremely constrained devices with packet sizes of just 30 bytes, making those implementations impossible/extremely expensive to run.

To resolve this, TinySec took a balanced approach making a compromise between the level of security, packet overhead and resource requirements. The end result proved that it's possible to secure a WSN efficiently entirely in software, without the need for additional hardware.

Communication between nodes, not just nodes-to-base-station, in WSNs is often quite important, allowing nodes to not only redirect other's traffic along routes but also consolidate duplicate packets from multiple nodes about the same event, saving the overall network from wasting power receiving and transmitting the extra packets; tinySec chose to engineer in security at the link layer, allowing these mechanisms to perform without alteration. The security goals of TinySec aimed to enable access control, whereby only authorised participants may participate in the network, with unauthorised messages easy to spot and reject; ensure message integrity, so that authorised messages can't be illegally altered by a man-in-the-middle without the receiver noticing; and ensure confidentiality, to ensure information is kept secret from unauthorised eavesdroppers.

The TinySec implementation uses Cipher Block Chaining with an initialisation vector (IV), together these achieve semantic security, therefore ensuring that encrypting the same plain text twice returns a different cipher text each time. So that the receiving end knows how to begin decryption of the data, the IV must be sent in the clear along with the encrypted data. When using an IV, its length needs to be taken into consideration because repeats will occur when the number wraps, causing a security vulnerability. On unconstrained devices an IV is usually 8 or 16 bytes, however due to the packet size limitations of the wireless sensors used, a 8 byte IV counter was chosen. In the IV, 6 bytes are made up of pre-existing fields to conserve space and ensure globally unique IVs in the network, with only 2 bytes acting as the counter, thus lowering the size and duration of protection e.g. if two nodes send the same data and both happen to have the same counter value, but differ in source address (src), then the IVs will differ, therefore security is preserved.

For ensuring authenticity and integrity of messages, TinySec uses Cipher Block Chaining Message Authentication Codes (CBC-MAC) of 4 bytes in length. Similar to a CRC, CBC-MAC runs over the data and produces a 4 byte MAC which is appended to the packet. If a message was to be altered, the attacker has a 1 in 2^{32} of blindly forging a valid MAC. In

a WSN with a limited send rate of 19.2Kb/s it would take over 20 months to send enough packets to possibly succeed in forging a MAC. In the case of attack, a receive heuristic can be used to detect multiple failed MAC transmissions at a nearby node, triggering an alert to the rest of the network.

Whilst TinySec can secure a WSN against eavesdropping and forged messages, it has two significant drawbacks. Firstly, in regards to key distribution, encryption and authentication keys need to be loaded onto the nodes prior to deployment. This can cause issues when the shared keys need to be changed, such as when they are compromised, as all nodes in the network will need the new key. This can be especially difficult post-deployment, simply due to the number of nodes and often embedded and/or difficult-to-reach locations. Secondly, if a node in the network is compromised, because the authentication key is a network-wide shared key, the illegitimate node can not only eavesdrop on all traffic but can also pretend to be any other node in the network.

3.3.2 MiniSec

MiniSec was created to tackle several problems apparent in the then current WSN security protocols, TinySec and Zigbee [18]. The pre-cursor to MiniSec, TinySec, received much attention and use due to its power and resource efficient security implementation, but because of limited authentication and lack of replay prevention, the overall security provided was deemed insufficient to protect a WSN. A commercial alternative, Zigbee, exhibits significantly higher security, but does so at the cost of higher energy consumption. MiniSec was designed to find the middle-ground between the two, increasing security whilst remaining energy efficient.

In contrast to TinySec, for its encryption mode of operation, MiniSec uses Offset Code Book. Unlike cipher block chaining (CBC) which requires two passes to provide both encryption and authentication, OCB provides both in only one pass over the data. This one pass also performs faster that CBC's two passes and only requires one key for operation, thus making it more appropriate for a constrained device in terms of power and storage.

MiniSec also differs from TinySec by reducing the size of the IV counter sent in a packet, yet managing to increase the size of the IV so that it wraps less often. This is achieved by storing some state about the IV counter locally and only transmitting the last n bits of it to the receiving node. This also requires some logic on both sides to manage the counter in the event of packet loss larger than the range of values stored in the last n bits sent, i.e. when $> 2^n$ are lost. For MiniSec, the authors chose n = 3. Because of this, only 3 extra bits needed to be sent with a packet, instead of the 2 extra bytes in TinySec. However, this overhead could be removed altogether. The maximum default packet size in TinyOS is 29 bytes, therefore the 3 most significant bits in the packet-size byte aren't used and can instead be used to store the IV counter. Therefore the need for the extra byte to store those bits is eliminated.[2]

[2]The publicly available source code for MiniSec (`https://sparrow.ece.cmu.edu/group/minisec.html`) does not make use of this technique, instead it appends an additional byte, thus reducing the

Another improvement, is the use of a synchronised counter as a nonce to prevent replay attacks. In every packet a monotonically increasing nonce is encrypted with the payload, when the recipient receives a packet with a nonce lower than its counter, it ignores the packet, therefore replayed packets are dropped.

3.3.3 ContikiSec

Similar to TinySec[14] and MiniSec[18], ContikiSec [12] is an asymmetric cryptographic security network layer, however, it's built for the other significant WSN OS, Contiki, instead of TinyOS. The paper presents two main contributions; first, an extensive evaluation of several block-ciphers and modes-of-operation, comparing ROM/RAM sizes and timings; second, a modular asymmetric encryption network layer for Contiki, offering modes for encryption, authentication or encryption + authentication.

After comparing several block ciphers (AES, Skipjack, RC5, Triple-DES, Twofish and XTEA), AES was chosen for use in ContikiSec due to its good trade-off between resource consumption and security. This is in contrast to both TinySec and MiniSec, which chose to use Skipjack. At the respective times of publication of TinySec and MiniSec, Skipjack was deemed sufficiently secure by NIST until 2008. Similar to MiniSec, ContikiSec also uses offset codebook as its mode of operation, combining both encryption and authentication into one pass.

3.4 Key Distribution Problem

Whilst previous work has demonstrated it is feasible to secure a wireless sensor network against eavesdropping, unauthorised participants and replay attacks, the issue of key distribution still remains. In the typical academic scenario, this may be a non-issue due to the expertise of the users. However, as described in the scenario in section 2, the typical home user will only have a very basic knowledge, if any, of how to use and operate the devices, let alone configure and distribute network encryption keys on all of their Things. Therefore, it becomes necessary to investigate a method by which the keys for symmetric cryptography can be distributed, with minimal effort and knowledge required from the user.

3.4.1 USB interface connectivity

A simple method of distributing keys could be to force the user to plug the Thing into their PC or router, which would then automatically program the correct key into the device with minimal user interaction. Whilst this seems attractive at first, the main issue with this method is that it requires every Thing to have a USB port, plus the appropriate circuitry and logic to perform the programming. This not only increases the cost for the Things,

benefit of the proposed reduced packet-size state-based counter.

as they now require additional components, but it can also increase the physical size and complexity of using them.

3.4.2 Message-in-a-bottle

In an attempt to distribute session keys in a user friendly way, and without the use of physical interfaces or asymmetric cryptography, Message-in-a-bottle utilises several physical techniques to carry out a wireless keying mechanism resilient to typical WSN attack vectors [15]. To achieve this, a Faraday cage and RF jamming is used in combination with a wireless session key distribution protocol.

To distribute the session key to a new uninitialised device, a user places the keying device and the new device entering the network into a Faraday cage[3]. The keying device then wirelessly transmits keying information to the node inside the cage. Meanwhile outside the cage, a keying beacon jams the wireless frequency of the shielded devices, in an attempt to ensure no leakage happens due to imperfections in the cage. After the keying session has complete, signalled by the keying beacon blinking its blue LED, the new node can be taken out of the cage. The keying beacon then communicates with the device, which will then construct the key based on what it received inside the cage, enabling it to connect to the network. If successful the keying beacon will blink its green LED.

Message-in-a-bottle presents a novel method of performing key distribution, however, whilst it reduces the cost of each device due to the absence of a physical connector, it still requires additional hardware (Faraday cage) to perform the key distribution protocol; this not only increases the burden of managing the network (storing and finding the cage), but also still presents an additional non-trivial task for the user to perform, especially in the case of a new network filled with tens of devices. The paper presents an additional method for distributing keys to multiple devices simultaneously, but this requires a large Faraday cage (pot size), further increasing the burden of managing the network. Another issue to consider is that whilst some devices may be small enough to fit in the cage, many others, such as household appliances, will not; therefore, a more appropriate key distribution protocol is necessary for an IoT network.

3.4.3 Asymmetric Cryptography

Another possible alternative for distributing the symmetric shared keys is to use asymmetric cryptography to establish an expensive, albeit short-term, secure channel between the two participants over an insecure channel and then exchange the key over this channel. The following section describes this in more detail, as well as some previous work related to WSN and asymmetric cryptography.

[3]The authors used a steel pipe sealed at both ends.

3.5 WSN Security - Asymmetric Cryptography

Asymmetric cryptography, also know as public-key cryptography, is a class of cryptographic algorithms which enable two or more participants to establish a secure channel over an insecure channel with no prior knowledge or shared secrets beforehand. Participants generate a pair of keys before starting to communicate, known as their public and private keys. The private key must be kept secret, whereas the public key can be distributed freely.

To securely send data to another participant, both must exchange one another's public keys; this can be done over an insecure channel. The newly received public key can then be used to encrypt a packet to send to the other participant. Once a packet has been encrypted with a participant's public key, only that participant can decrypt it using their private key and read the secret inside; an eavesdropper would only have the public key, which can't be used to decrypt the packet, hence the term asymmetric.

Alternatively, the private key can also be used to sign some data, which other participants can authenticate using the public key. This assures the recipient that the data was sent by the signee, the only one with the private key; however, because anyone with the public key can view the data, it is therefore not secret.

Whilst asymmetric cryptography seems to solve the problems previously discussed in symmetric cryptography, it also has some issues of its own. Firstly, in terms of size and speed, asymmetric algorithms tend to be several orders of magnitude slower than symmetric algorithms, due to the more complex algorithms used to provide the asymmetry, therefore use of them continuously, especially on a WSN where processing power is severely limited, can reduce the throughput and efficiency of the device. Secondly, whilst it's possible to distribute public keys in the open, a man-in-the-middle attack can compromise the security, in which a third party could intercept the public keys and switch them out for their own; after which, the participants would believe they are communicating securely, but are instead passing traffic through the third party.

The rest of this section will discuss two different implementations of asymmetric cryptography algorithms for WSN, as well as a possible solution to the authentication problem.

3.5.1 Certificates and Public Key Infrastructures

By itself, asymmetric cryptography only allows two participants to create a secure link over an unsecured channel, preventing eavesdroppers. It does not allow authentication of either participant, meaning that an active eavesdropper could perform a man-in-the-middle attack.

To counter this, a public key infrastructure can be used to pass the authentication up to trusted body. This trusted body is know as a certificate authority. The CA's job is to issue public key certificates for agents that want to be trusted by others. An agent can obtain a PKC by presenting the CA with its details and public key, which the CA approves

and signs the pair using its own private key; it then returns the signed certificate to the agent along with the CA's public key. Now, when the agent carries out an asymmetric key exchange with another party, it can send its certificate to the other party, which can then verify it using its own copy of the CA's public key. This allows two parties to now authenticate each other's public key, ensuring that an active eavesdropper hasn't performed a man-in-the-middle attack and switched out the keys for their own. However, now both parties must trust the CA and obtain a personal PKC and CA private key from it; if the CA were to be compromised, all trust is lost.

3.5.2 TinyPK

TinyPK is an implementation of public-key protocols using the Rivest Shamir Adelman (RSA) algorithm for use on TinyOS WSNs[21]. The purpose of the paper was to demonstrate that it's viable to selectively use public key protocols to securely distribute shared session keys for later use in symmetric cryptography.

In the context of the paper, there are two entities, a sensor network and a third party that wishes to connect to the network. Also apparent is a public key infrastructure, which contains a trusted certificate authority. This CA assists with authenticating all communicating members. Prior to deployment, public and private key pairs are generated for all members (static keys). The CA signs each member's public key with its (CA) private key and then gives back the signed public key and the its own public key to the respective owner. This enables any member to authenticate a public key received from another member, assuring it that a man-in-the-middle attack can't have occurred.

One of the issues TinyPK tries to solve, is the high time cost associated with performing private key functions on the motes, stating that they could take tens of minutes to perform. To avoid this, it is assumed that the third party that wishes to connect to the WSN has more resources at hand and can compute the private functions within reasonable time bounds. This in effect means that any signing required on the sensor side needs to either be avoided or be precomputed by the CA using its private key. An example in this context, is a mini certificate used to identify a sensor, containing meta-data about the sensor such as ID, date of construction and type. Because the sensor doesn't have the computational power to sign it, the CA can sign it using the CA's private key, therefore any other device with the public key can read it and trust the CA.

The security protocol is itself a simple challenge-response protocol. First, the third party decides to connect to the sensor network, sends it's CA signed public key and sets a challenge in the form of a nonce to protect against replay attacks. If the receiving sensor is legitimate it will contain the CA's public key which can be used to authenticate the public key, using this the device can then un-sign the challenge nonce. Now that the sensor can prove the third party is legitimate, it can then choose to reply to it using the valid nonce it received along with the session key. This time the packet is encrypted using the third party's received public key, ensuring no-one else can eavesdrop on it. Upon reception the third party verifies the nonce and then stores the session key for future use.

Because of the high cost of private key on the wireless sensor nodes, it's not possible to authenticate a node prior to passing over the session key, therefore creating a possible weakness. However, to try and resolve this, TinyPK uses an additional Diffie-Hellman challenge over the secure session channel to verify the sensor node's authenticity. This further complicates the process and also allows an unauthenticated sensor node to communicate with the third party prior to the authentication process, possibly making the third party vulnerable, especially if the session key is used network-wide.

3.5.3 TinyECC

As previously discussed, public key cryptography (PKC) using traditional algorithms, such as RSA, has had a limited deployment in WSN due to its high computational cost and implementation size. In cases where it has been used, implementations, such as TinyPK, only using a subset of the operations on the sensor node in order to reduce the overhead, but also reducing the effectiveness of it.

As an alternative to RSA and other typical algorithms, elliptic curve cryptography (ECC) shows promise for using PKC on WSNs, due to its lower computational overhead, enabling both public and private operations to be carried out on a sensor node, as well as it reduced key size and compact signatures, all the whilst maintaining the same security as RSA[19] e.g. a 1024-bit RSA key size is equal in security to a 160-bit ECC key size. These benefits make it far for suitable for use on WSNs when compared to RSA, thus removing the need to offload work to other, more power, devices[21].

TinyECC [17] is an implementation of ECC for the TinyOS WSN OS, featuring a full set of public and private key operations, unlike TinyPK. It also has various optimisation switches, allowing developers to balance implementation size against performance. TinyECC was tested on a variety of TinyOS-compatible constrained sensor platforms, including MICAz, TelosB, Tmote Sky and Imote2, extensively proving that it's feasible for a wide range of WSN platforms.

As mentioned in TinyPK, private key operations for signing blocks of data took in the order of tens of minutes[21], whereas, with the use of ECC, TinyECC achieves the same operation in 1.6s when all optimisations are used. Similarly, TinyECC also achieves encryption speeds of 3.3s/pkt and decryption speeds of 2.1s/pkt. Whilst these rates are several orders of magnitude slower than symmetric algorithms, they are only normally used for securely bootstrapping keys for symmetric cryptography, which would then be used instead.

3.6 Other Works

3.6.1 MQTT

3.6.2 IETF Work

[13, 20]

3.7 Problems with previous work

Typical wireless sensor network security has considered and used symmetric cryptography[14, 18, 12] for securing networks in the wild, however key distribution has been bootstrapped (burned to ROM) prior to deployment. This would not only increase the difficulty of bringing a new Thing into a home network, but would also require each Thing have additional hardware in order to interface with the key distributor e.g. a PC or router.

Alternatively, asymmetric cryptography has also been proved viable on typical WSN platforms thanks to elliptic curve cryptography which provides significant performance benefits of previously existing methods[17]; therefore allowing pairs of nodes in a network to dynamically authenticate each other and create shared secret keys. Whilst there is some improvement in performance with the use of ECC over RSA, it still considerably slower and consumes far more resources than symmetric cryptography, as shown in the table below.

	Symmetric(TinySec)	Asymmetric(TinyECC)
ROM size	3KB	15.65KB
RAM size	300B	1.8KB
Encrypt	<2ms/pkt	3.2s/pkt
Decrypt	<2ms/pkt	2.1s/pkt

Table 1: Cryptographic implementation sizes and performances on the Mica2(8Mhz) and TelosB (8Mhz)mote respectively[4]. [14, 17]

However, by using public key cryptography to initially bootstrap the symmetric cryptography with session keys, the computational impact of using PKC needs to only be dealt with once at boot-time, whilst allowing for dynamic session key assignment. This is similar to TinyPK, however, because of the innovation of ECC, it's no longer necessary to perform a reduced and more complicated authentication process[21].

Discuss issue of segmentation of home network and IoT network Little has been done to present a fully deployable and dynamic secure WSN that is suitable for the home.

[4]Based on TinySec's implementation (8MHz CPU) where encrypt/decrypt operations take 0.38ms to perform over 64 bit(8 byte) blocks, with TinyOS's default packet size set to 29 bytes.

4 Proposed Approach

4.1 Solving the Problem

forward secrecy Demonstrative identification.

4.2 Security Architecture

4.2.1 Symmetric Key Cryptography

4.2.2 Asymmetric Key Cryptography

4.3 Implementation of IoT Protocol on TinyOS

4.4 Integration of IoT with Smart Home Router

5 Work Plan

- Extend IoT protocol to include security
- Port Secure IoT protocol on TinyOS
- Create host proxy for Secure IoT Protocol
- Create automata for Homework

References

[1] 2013: The year of the Internet of Things. http://www.technologyreview.com/view/509546/2013-the-year-of-the-internet-of-things/. Accessed: 21/03/2013.

[2] Amazon Cloud outage. http://aws.amazon.com/message/680587/. Accessed: 21/03/2013.

[3] Google Mail outage. http://news.cnet.com/8301-1023_3-57415281-93/gmail-users-experience-outage/. Accessed: 21/03/2013.

[4] If This Then That service. https://ifttt.com/. Accessed: 21/03/2013.

[5] Internet of Things first coined. http://www.rfidjournal.com/article/view/4986. Accessed: 21/03/2013.

[6] Smart Things IoT platform. http://smartthings.com/. Accessed: 21/03/2013.

[7] Sony Playstation Network Breach. http://blog.us.playstation.com/2011/04/26/update-on-playstation-network-and-qriocity/. Accessed: 21/03/2013.

[8] Twine "Internet of Things" Thing. http://supermechanical.com/. Accessed: 21/03/2013.

[9] Xively, Internet of Things public cloud. https://xively.com/. Accessed: 21/03/2013.

[10] Anthony Brown, Richard Mortier, and Tom Rodden. Multinet: Reducing interaction overhead in domestic wireless networks. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '13, pages 1569–1578, New York, NY, USA, 2013. ACM.

[11] Patrick Brundell, Andrew Crabtre, Richard Mortier, Tom Rodden, Paul Tennent, and Peter Tolmie. W-must'11 best papers-the network from above and below. *SIGCOMM-Computer Communication Review*, 41(4):519, 2011.

[12] Lander Casado and Philippas Tsigas. Contikisec: A secure network layer for wireless sensor networks under the contiki operating system. In Audun Jsang, Torleiv Maseng, and SveinJohan Knapskog, editors, *Identity and Privacy in the Internet Age*, volume 5838 of *Lecture Notes in Computer Science*, pages 133–147. Springer Berlin Heidelberg, 2009.

[13] Castellani. CoAP to HTTP mapping. https://datatracker.ietf.org/doc/draft-castellani-core-http-mapping/. Accessed: 21/03/2013.

[14] Chris Karlof, Naveen Sastry, and David Wagner. Tinysec: A link layer security architecture for wireless sensor networks. In *Proceedings of the 2Nd International Conference on Embedded Networked Sensor Systems*, SenSys '04, pages 162–175, New York, NY, USA, 2004. ACM.

[15] Cynthia Kuo, Mark Luk, Rohit Negi, and Adrian Perrig. Message-in-a-bottle: User-friendly and secure key deployment for sensor nodes. In *Proceedings of the 5th International Conference on Embedded Networked Sensor Systems*, SenSys '07, pages 233–246, New York, NY, USA, 2007. ACM.

[16] F.W. Leahy. A lightweight protocol for constrained devices for use in the Internet of Things paradigm. Technical report, University of Glasgow, 2013. 4th Year Dissertation.

[17] An Liu and Peng Ning. Tinyecc: A configurable library for elliptic curve cryptography in wireless sensor networks. In *Information Processing in Sensor Networks, 2008. IPSN '08. International Conference on*, pages 245–256, 2008.

[18] Mark Luk, Ghita Mezzour, Adrian Perrig, and Virgil Gligor. Minisec: a secure sensor network communication architecture. In *Information Processing in Sensor Networks, 2007. IPSN 2007. 6th International Symposium on*, pages 479–488. IEEE, 2007.

[19] Certicom Research. Standards for efficient cryptography sec 1: Elliptic curve cryptography. `http://www.secg.org/download/aid-385/sec1_final.pdf`, September 2000.

[20] Z. Shelby. IETF, Constrained RESTful Environments - Resource Discovery, 2012. RFC6690, 1.2.1.

[21] Ronald Watro, Derrick Kong, Sue-fen Cuti, Charles Gardiner, Charles Lynn, and Peter Kruus. Tinypk: Securing sensor networks with public key technology. In *Proceedings of the 2Nd ACM Workshop on Security of Ad Hoc and Sensor Networks*, SASN '04, pages 59–64, New York, NY, USA, 2004. ACM.