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 ([8], [10], [26]), 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 [13], [12]. 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[23] 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 [19]. 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 [19] 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(devices), but the closed loop of control can be affected by illegal packet injection, to negatively alter its behaviour; therefore it becomes extremely important to not only protect the data from unauthorised access, but also protect the network itself. As 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.

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** - An attacker attempts to block or overwhelm the target, so that legitimate requests to contact the target will be unsuccessful. In the case of WSN this can occur from RF jamming, or by a packet based overload where the target is forced to continuously process computationally expensive packets for an extended period of time, such as with asymmetric cryptographic operations.

The attack vectors to be tackled by this project will be discussed in section 4.2.1.

2.2.2 Integration

The current implementation exists as a standalone library with several demo applications. Integration of the IoT protocol into a more powerful and 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 hundreds of available Things, subscribe and log events from the sensors, and using user customised rules, run automata to detect if these rules are met and then publish commands to actuators in the network to perform actions within the home.

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 [8, 10]. Whilst this yields certain benefits, such as easy external access and integration with other services [5, 11], 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 for the duration of this project.

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 within the IoT.

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 the beginning where it was the concept of using barcodes, and later RFID[6], 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 notifying a user when the washing machine has finished a load of laundry, turn on a light when the user enters a room or adjust the heating schedule depending on who’s home. 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 IoT protocol

Our previous work[19] demonstrated the need for a new protocol specifically designed for constrained devices in the IoT paradigm, taking into consideration not only the computational constraints, but also the often limited power resources available. As a result, a lightweight and scalable networking protocol, tailored around the IoT architecture, was designed and implemented for TelosB motes running the Contiki OS.

Within the protocol, devices in the IoT architecture were divided up into three categories of Things. Sensors, devices which sense the environment and transmit the data to interested parties at set intervals; actuators, devices which could receive commands and interact with the environment e.g. display data, turn on a light bulb; and controllers, one or more devices which controlled the network of things, connecting to both sensors and actuators to create a closed loop of control e.g. several sensors send temperature readings to a controller, which decides it’s too hot and sends a command to the air conditioner actuator to cool the room.

Whilst the protocol was deemed successful when compared to existing protocols, there are some issues. Firstly, there’s no security, packets are transmitted in plain sight with no encryption. Secondly, in relation to integration, whilst the protocol worked well within the network of Things, it lacked integration into other, more powerful platforms, such as a PC, which would enable it to become far more powerful, dynamic and customisable, as well as allow users to interact with it more easily.

3.1.2 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.

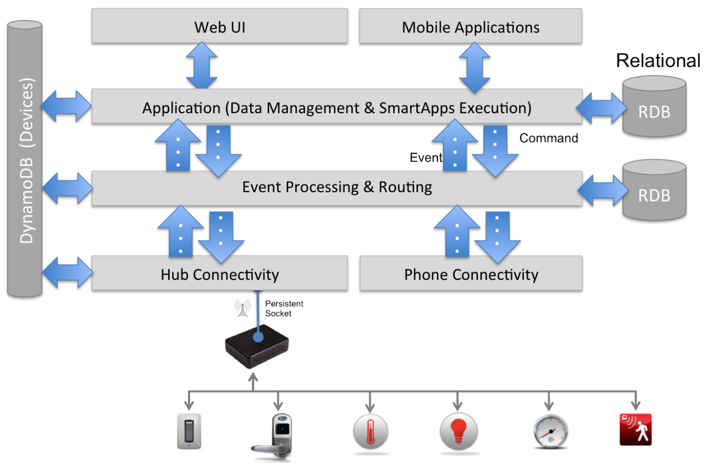

Figure 1: Smart Things - Cloud approach[1]

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 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 [9, 2, 4].

3.1.3 CoAP

CoAP, Constrained Application Protocol[15], is currently a proposed IETF standard to create a request-response style web protocol, specifically designed to directly bridge the gap between the Internet and constrained devices, allowing Things to communicate and respond directly to other devices on the Internet, similar to the typical client-server interactions at present. The protocol is designed as an asynchronous RESTful protocol, to run on top of

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

UDP, using a subset of the standard commands e.g., GET, PUT, DELETE. The reason is that due to the common commands, it allows for an easy transition to standard HTTP, if necessary, whilst reducing the complexity of the protocol.

Unlike the two platforms/protocols previously discussed, this protocol potentially places Things directly on the Internet just like normal HTTP servers. This not only makes these devices much easier to connect, as it's just like connecting to typical servers, but also has similar risks to Smart Things, where devices could be vulnerable and attacked directly, just like a typical server. In relation to this, Symantec recently discovered a new Linux worm[7] targeted directly at IoT devices, thus proving the possible risks of creating such devices.

3.2 Home networking

In the last decade, homes have become filled with a myriad of different computing devices, including PCs, laptops, tablets, phones, all with wireless connectivity; therefore requiring the user to install and manage a small-scale wireless network. Whilst networks have emerged in the home, similar to those found in a typical enterprise setting, the tools to manage them have remained largely untouched, designed for network engineers – not home users. This leaves users to manage features such as access control, firewalls, port forwarding and many others, using primitive tools designed for experienced and qualified network managers. Because of this, networks are often badly managed or their features underutilised.

3.2.1 Homework

In an attempt to fundamentally change the home network, the Homework project[24, 23], a re-invention of the protocols, models and architectures of the domestic setting led by ethnographic studies of use, was created. By reinventing the home network from the ground up around the user, unlike typical face-lift attempts by manufacturers, its aim is to enable the typical home user to manage and be in control of their home network, using tools that induce a minimal overhead on the user.

To do this, Homework utilises an information plane architecture that uses stream database concepts to process a raw event stream, generate derived events and allows monitoring applications to publish and subscribe to events in the stream [27]. Atop of this, a policy management engine, enables users to manage access and bandwidth control, using a user friendly and informative web interface, utilising the stream database to monitor the policies and display network statistics in a user-understandable way e.g. displaying a user's devices with progress bars displaying the percent bandwidth utilised.

Underneath, Homework uses Glasgow automata[17], concise complex processing engines which run within the context of the steam database, subscribing to events and generating new ones based on its programmatic definition. Automata are created using the Glasgow automata programming language, an imperative programming language enabling programmers to create complex query processing tasks on the fly, as opposed to predefined query languages.

As more networked devices enter the home, including the flood of devices soon to be brought by the Internet of Things, managing a home with multiple disconnected and heterogeneously filled networks will become an even more important and difficult task for the home user to perform. Thus, the task of intelligently merging these networks together and creation of an all in one network and information management system needs to be undertaken, ensuring that new attack vectors aren't exposed by the thoughtless integration of new devices. Using the Homework information plane architecture, this concept appears possible; even with the event streams filled with hundreds, if not thousands of events, generated by devices, including Things and typical computing devices, Homework appears ready to handle it[17]. Most importantly, the end result must abstract away the complexities, so that a typical home user can manage and maintain control over the network, utilising more complex and varying policies than before in order to manage the closed loop of control required for an IoT.

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, therefore 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 as the key), and block-based ciphers, where data is segmented into blocks and operated on by a set of functions which modify it based on the key, often with the previous cipher block providing input into the next block.

In the case of WSN, block-based ciphers have seen exclusive use. However, one issue with a standard 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 one.

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 others' 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 its value repeats, 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[2] 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 [22]. 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

[2]An unfortunate collision between abbreviations for media access control and message authentication code, the latter will only be referred to in this report.

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.[3]

Another improvement, is the use of the synchronised counter (also used as the IV) to prevent replay attacks. As each packet is received, the counter is incremented accordingly, thus if a packet arrives with a counter less than the one stored locally, it is dropped as one can determine it must be a replayed packet.

3.3.3 ContikiSec

Similar to TinySec[16] and MiniSec[22], ContikiSec [14] is a symmetric 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 symmetric 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]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 benefit of the proposed reduced packet-size state-based counter.

3.4.1 USB interface connectivity

A simple method of distributing keys would be to require 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, as they now require additional components, but it can also increase the physical size and complexity of their design and in using them. This also might not be possible due to the type and location of the device, preventing it from being moved e.g. it's difficult to move a fridge or cooker to program it.

3.4.2 Message-in-a-bottle

In an attempt to distribute symmetric 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 [18]. To distribute the session key to a new device, the user places the keying device and the new device into a Faraday cage[4]. 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 five seconds the exchange completes and the user removes the device, which will then construct the key based on what it received inside the cage, enabling it to connect to the network.

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 (using, 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 also 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 session 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.

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

The following section describes this in more detail, as well as some previous work related to WSN and asymmetric cryptography.

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. 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 a solution to the man-in-the-middle problem, as well as two different implementations of asymmetric cryptography algorithms for WSN.

3.5.1 Certificates and Public Key Infrastructures

By itself, asymmetric cryptography only allows two participants to create a secure link over an insecure channel, preventing eavesdroppers. It does not provide authentication for either participant, therefore, 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 known 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 can't have 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[28]. 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. There is also 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 its own private key, therefore any other device with the CA's public key can read it and must trust the CA.

The security protocol itself is a simple challenge-response protocol. First, the third party decides to connect to the sensor network, sends its 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 operations 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, such as TinyPK, implementations only use a subset of the operations on the sensor node in order to reduce the long temporal overhead associated with them; however, this also reduces the effectiveness of the security and forces workarounds to be created, as TinyPK demonstrated.

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, not only consuming less storage but also reducing lengthy public key packet transmissions. This is achieved whilst maintaining the equivalent security as RSA[25] e.g. a 1024-bit RSA key size is equal in security to a 160-bit ECC key size. These benefits make it far more suitable for use on WSNs when compared to RSA, thus removing the need to offload work to other, more power, devices[28].

TinyECC [21] 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[28], 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 a short period when securely bootstrapping keys for symmetric cryptography, which would then be used after completion.

3.6 Summary of issues with previous efforts

Little work has been done to present a fully deployable and dynamic secure WSN that is suitable for the home. Typical wireless sensor network security has considered and used symmetric cryptography[16, 22, 14][5] 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 over previously existing methods[21]; therefore allowing pairs of nodes in a network to dynamically authenticate each other using certificates and share secret keys over a secure channel using each other's public 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 1.

	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 of TinySec and TinyECC, on the Mica2(8Mhz) and TelosB (8Mhz)mote respectively[6]. [16, 21]

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[28].

For the latter half of the project, existing attempts at integrating the IoT into the home still present some problems. As previously discussed in prior work[19], many Things are simply embedded with a WI-FI transceiver enabling them to talk the Internet individually, creating a disjoint network of Things all managed and interacting independently, creating a significant burden for the user. To make matters worse, Symantec recently discovered a new Linux worm[7] that appears to be targeted at Internet of Things devices, of which many home users are unaware that they are vulnerable. Other attempts, such as Smart Things,

[5]MiniSec is the most secure symmetric implementation available at the time of writing for TinyOS. Another implementation called SenSec, referenced by several other later papers, was unavailable both in paper-form and its implementation.

[6]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. MiniSec doesn't provide timed results of the implementation, only energy consumed.

present a cloud approach whereby there are also considerable risks related to security, robustness and privacy.

In contrast to these other attempts, a local approach, where the connectivity, device management and policies are managed within the home might be a better approach, ensuring the network can only be accessed from one point externally, therefore reducing the overhead for managing and updating software to combat new vulnerabilities. Furthering this, combining the IoT network with the traditional home network filled with PCs, tablets and mobile devices, to create a single network of devices, where the user can control, manage and configure in one place, would create a more seamless and user friendly experience. As discussed in section 3.2, the Homework project attempts to reinvent the home network, enabling novice users to have full control over their network, without the need of network expertise, expensive hardware or the cloud[7] and by combining this with the IoT protocol, it could create an effective and powerful all-in-one home information platform and router.

4 Proposed Approach

After surveying the previous work completed in the various related fields, it's now possible to propose a feasible approach that can be undertaken in order to achieve the aims of this project. This section will discuss the two independent but related aims and the proposed approach to solve each one, in order to create a secure IoT platform integrated into the existing Homework platform.

4.1 Solving the Problem

The aims of this project are two-fold; firstly, secure the currently existing IoT protocol developed in our previous work against eavesdroppers and unsanctioned participants; secondly, integrate the IoT controller role into the Homework information plane architecture, enabling the IoT network to be managed and controlled by Homework and in effect, the user through policies.

4.2 Security Architecture

As previously discussed, there have been many attempts to create a secure WSN, both utilising symmetric and asymmetric cryptography; however, there have been very few attempts to demonstrate their use in tandem, to provide a dynamic secure network that allows nodes to enter and leave the network, without prior knowledge or pre-installed secrets. Some attempts have been made to circumvent the need for using asymmetric cryptography for key

[7]Homework project demonstrated its implementation on an EEPC 1000H netbook with an Atom 1.6GHz CPU and 2GB RAM.

distribution, due to its prohibitive cost in time and space on constrained devices; however, these attempts were not only complex [28], but also infeasible as networks scale (bigger networks and devices)[18]. With the innovations made with ECC, asymmetric cryptography has become a far more feasible approach for secure key distribution. Therefore, for this project, asymmetric cryptography will be used to initially bootstrap the session keys for later use in symmetric cryptography, enabling new devices to securely join the network without prior knowledge of it.

4.2.1 Attack vectors

As described in section 2.2.1, there are a variety of possible attack vectors that can be employed by attackers; for the purpose of this project, only the first two, eavesdropping and unsanctioned participants, will be fully considered and mitigated against. Node capture is relatively difficult to protect against due to the nature of the placement of Things within the environment, thus it will be left up to the user to ensure devices within the network are kept safe, just as they would with other property. Similarly, denial-of-service is difficult to protect against due to the nature of wireless networking where RF jamming can occur; however, some attempt will be made to reduce the effect of packet-based DoS attacks, which aim to overload the target by initiating complex processing tasks such as asymmetric cryptography tasks.

4.2.2 Asymmetric Cryptography

For the asymmetric cryptography part of the security protocol, TinyECC[21] will be used to initially authenticate new nodes and bootstrap the session keys that will then be used for symmetric cryptography. Once the key exchange is complete, the TinyECC code can be removed from RAM, as it won't be needed again unless the node disconnects for the network or is rebooted. Section 4.2.4 discusses the key exchange process in more detail.

Whilst TinyECC makes asymmetric cryptography feasible on constrained devices, it's still extremely costly, taking several seconds to compute operations. Therefore, if asymmetric key exchange packets were to always be processed, it could open up the device to DoS attacks, whereby an attacker attempts to overload the node by continuously sending these packets to the target. To attempt to reduce the impact of this, the user will be required to press a button on the device that will cause the device to silently enter discovery mode for a short time period. Whilst in discovery mode, the device will accept the costly asymmetric key exchange packets(unsigning the certificate); then, once the device has exited discovery mode, it will drop all asymmetric key exchange packets it receives, reducing the computational overhead and thus limiting the effect.

In order to authenticate the devices and ensure a man-in-the-middle attack cannot occur in the network upon initial contact, a public key infrastructure will need to be set up. This will consist of a global certificate authority, which will sign the public keys of all

Things, including the Homework router[8]. The CA will return its public key along with the newly created public key certificate for that device, to be burned onto the device prior to deployment. Using these certificates, a device can prove its identity and authenticity to any other node in the network which contains the CA's public key. During the authentication process the user will be able to view the authenticated devices available to connect to in the network and select the appropriate one, from within an interface on the router. On the device, the user will be able to observe that it shows the correct status (via LEDs) and connects to their router, not to an attackers device, thus giving them demonstrative identification of which device the router is connecting to, as well as the success and security of the system.

4.2.3 Symmetric Cryptography

For the symmetric part of the security protocol, MiniSec[22] will be used to ensure that eavesdroppers can't see what data is traversing the network and stop unsanctioned participants from performing replay attacks or injecting bad data into the network. In order for this to be achieved, both encryption and authentication needs to be used. MiniSec provides this combined encryption and authentication in one pass over the packet using the OCB mode of operation. For each node connected to the controller, a unique session key will be used. This will help reduce the cost of a single node being captured[9], as only the connection between the node and the router would be compromised, as opposed to the case in which the whole network would be compromised if a network-wide session key were to be used, such as in TinySec[16]. However, if a node is captured, it would still be possible to illegally alter the device or retrieve its public/private keys, therefore the risk of node capture can't be fully mitigated against.

For a normal packet, using MiniSec to encrypt and authenticate the payload will add an additional 3 bytes of overhead. This consists of removing the group (grp) and CRC fields as they are no longer necessary, replacing them with a source address field and message authentication code (MAC)[10]. The MAC field incorporates both integrity (replacing CRC), and verification, which is needed to ensure a packet is authentic and hasn't been altered by an attacker. Additionally, as mentioned previously in section 3.3.2, MiniSec provides replay prevention through the use of its IV counter embedded in the length field.

One limitation of the protocol is the length of the IV counter, once it repeats, semantic security is lost and a eavesdropper could retrieve some information. To counter this issue, a new session key can be issued at any time before the counter wrap-around and transmitted over the still secure channel; this will ensure semantic security is maintained and an expensive asymmetric re-key isn't necessary. Issuing new session keys is also good practice, ensuring active attackers won't have enough time to break any given key.

[8]Whilst this may seem infeasible if commercially deployed, it's possible to create a hierarchy of trusted authorities which can be dynamically accessed when authenticating devices.

[9]This won't limit packet forwarding over larger networks as headers will still be in the clear.

[10]Unfortunate collision between abbreviations for media access control and message authentication code, the latter will only be referred to in this report.

As previously mentioned, the current implementation of MiniSec uses the now unrecommended security algorithm, Skipjack. Lenstra and Verheul recommended use of Skipjack until 2012 based on its 80 bit key length[20]. For the purposes of this project Skipjack will be used as a proof-of-concept, and if time is made available, an attempt to integrate a more secure algorithm, such as AES, will be made.

4.2.4 Security protocol

The security protocol is split up into three main parts, as shown in figure 3. The bootstrapping phase, in which the devices are all independently (in time and space) issued unique public key certificates and the corresponding private keys, as well as the public key of the CA. The asymmetric phase, in which a new device is brought into the network and the router(controller) initiates contact with the device, they both authenticate each other and then the router sends the device the session key. The symmetric phase, in which the two devices now communicate using symmetric cryptography with the previously transmitted session key. The basic communication steps are as followed:

1. Bootstrap devices with public key certificate and privates keys (+ CA public key) prior to deployment.
2. Initiate discovery mode on Thing and router(controller).
3. Router and Thing authenticate one another. (A nonce is used to prevent replays.)
4. User is presented with a list of authenticated devices to connect to and selects the one they want to connect to.
5. Devices now exchange session key using secure channel (via authenticated public keys).
6. Asymmetric cryptography structures free'd and symmetric cryptography started. Thing then ACKs router over symmetric channel to signify handshake has complete.
7. IoT communications commence.

4.3 Implementation of IoT Protocol on TinyOS

Currently, their exists implementations of our IoT protocol for both the Contiki[11] and Arduino[12] platforms. However most of the security research carried out on WSNs has been implemented only for TinyOS, bar ContikiSec. There is an implementation of ECC for Contiki, but it is not feature complete and the project appears to have stalled[3]. Therefore, it will be necessary to port the protocol over to TinyOS. Due to the event based structure of the existing code base, this should be a trivial task.

[11]Contiki implementation available at: `https://github.com/fergul/KNoT`

[12]Arduino implementation available at: `https://github.com/fergul/KNoT_panstamp`

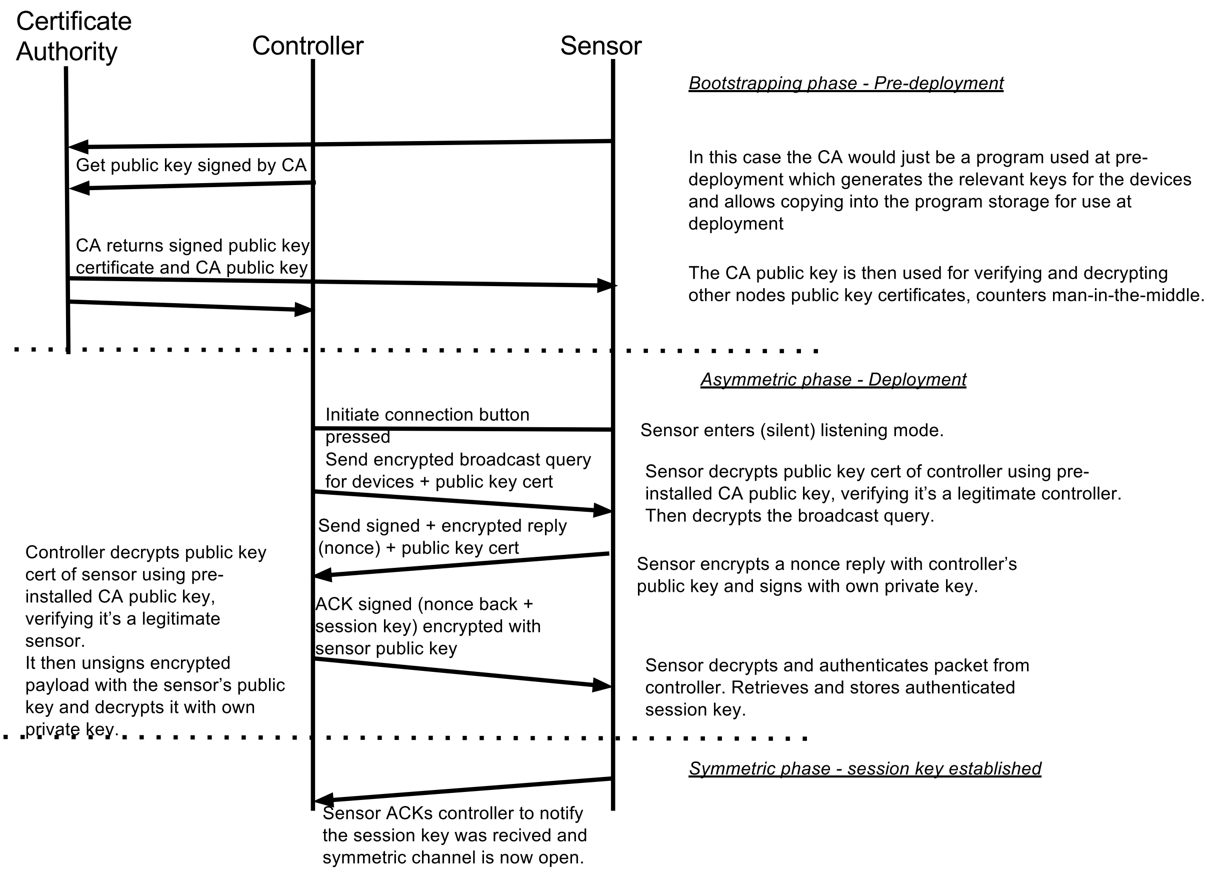
Certificate Authority
Controller
Sensor
Bootstrapping phase - Pre-deployment
Get public key signed by CA
In this case the CA would just be a program used at pre-deployment which generates the relevant keys for the devices and allows copying into the program storage for use at deployment
CA returns signed public key certificate and CA public key
The CA public key is then used for verifying and decrypting other nodes public key certificates, counters man-in-the-middle.
Asymmetric phase - Deployment
Initiate connection button pressed
Sensor enters (silent) listening mode.
Send encrypted broadcast query for devices + public key cert
Sensor decrypts public key cert of controller using pre-installed CA public key, verifying it's a legitimate controller. Then decrypts the broadcast query.
Send signed + encrypted reply (nonce) + public key cert
Controller decrypts public key cert of sensor using pre-installed CA public key, verifying it's a legitimate sensor. It then unsigns encrypted payload with the sensor's public key and decrypts it with own private key.
Sensor encrypts a nonce reply with controller's public key and signs with own private key.
ACK signed (nonce back + session key) encrypted with sensor public key
Sensor decrypts and authenticates packet from controller. Retrieves and stores authenticated session key.
Symmetric phase - session key established
Sensor ACKs controller to notify the session key was recived and symmetric channel is now open.

Figure 2: Security sequence diagram

4.3.1 Hardware

Similar to the previous work[19], this project will also use the TelosB motes as the target hardware platform. These devices have an 8MHz CPU, 10KB RAM and 48KB ROM, plus a variety of basic sensors.

Whilst speed and security have been discussed in-depth in relation to constrained devices, in order for this approach to be deemed feasible, another key concern of the proposed solution that must be considered is the resource consumption. The total resources of the combined components must not exceed the available resources for the hardware available i.e. TeloB mote. Based on the individual compile sizes of each module, shown in table 2, the proposed approach appears to be feasible and well below the limits of the device.

Components	**ROM Size**	**RAM Size**
MiniSec	1.8KB	300B
TinyECC	15.65KB	3KB
IoT	2.4KB	994B
Total	19.9KB	4.3KB

Table 2: Estimated total implementation size based on individual compiled sizes

4.4 Integration of IoT with Smart Home Router

The latter half of the project will be focused on implementing a proxy on the host machine, which will not only enable the Homework platform to communicate with the IoT network, but also allow it to manage and control the network of Things using its event processing capabilities.

TinyOS provides comprehensive tools for communicating between a device running TinyOS and a host machine, available in Java, C and Python. The Homework platform also provides Java and C bindings, enabling other applications to communicate with the database, publishing and subscribing to events in the event stream. Additionally, using the Glasgow automata programming language, relevant automaton will need to be created to demonstrate some basic closed loops interactions enabled by the use of the Homework platform e.g. an automata subscribes to a motion event, and publishes a turn on light event when a motion event occurs; this would then pass through the proxy to the IoT and switch the light on.

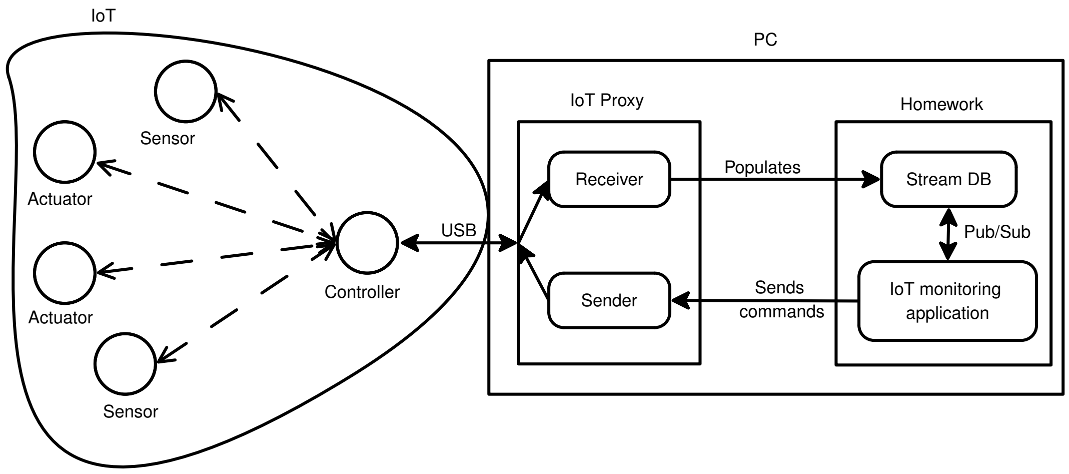

Figure 3: Controller - Sensor message diagram

5 Work Plan

This section will discuss the plan of work to be completed, highlight the target deliverables and discuss some possible risks the project could face in the following semester. The semester is fifteen weeks long, starting on Jan 13th and ending April 25th, with several intermediate deliverables to break up the workload.

5.1 Tasks & Deliverables

Within the project there are several tasks to be completed which go towards the two main aims, shown below. For each task, there are some sub-tasks and an overall deliverable. These tasks are also shown along with their durations and deadlines in a Gantt chart in figure 4 in section 5.3.

1. Port Secure IoT protocol to TinyOS. *(Deliverable 1)*
2. Extend IoT protocol to include security protocols. *(Deliverable 2)*
   (a) Implement asymmetric security protocol.
   (b) Implement symmetric security protocol.
3. Homework integration. *(Deliverable 3)*
   (a) Create host proxy for Secure IoT Protocol to connect to controller device and Homework DB.
   (b) Create monitoring application for Homework and test with demo policies.
4. Project report. *(Deliverable 4)*

Testing will be an ongoing process throughout the duration of the project in order to ensure the protocol implementation is working correctly and is omitted from the Gantt chart. For the latter part of the project, the monitoring application will function as a test of the entire system's functionality, including the protocol and integration.

5.2 Risks

Whilst the project has been well researched, planned and plenty of time allocated to it, there are still some risks that need to be considered and managed. Below are some of the key risks, their possible consequences and the techniques used to try and mitigate them.

- Support - Much of this project relies on third party libraries or code, which after researching and testing briefly, is deemed to be well supported and sufficient for the needs of the project. However, as the project develops, problems or bugs may arise with other's work, leading to unforeseen issues or delays.

- Architecture difficulties - TinyOS's architecture is based entirely on a split-phase event style architecture, however the previous implementations have been developed for imperative style systems. Therefore, they may be some difficulty in porting over certain aspects into an event style, which may stall/delay the project.
- Flawed security - As with all security protocols and algorithms, careful planning, verification and peer-review is absolutely necessary. The security protocol (certificate system) designed was strongly influenced by previous work and existing protocols in use today, such as the Chip and Pin system[13] and TinyPK. However, whilst precautions have been made, it is still possible an attacker may discover an unseen vulnerability in the system.
- Project overruns - Due to a misjudgement in difficulty or scale, the project could overrun, leading to only a subset of the original tasks being completed and possibly not achieving the project aims.

5.3 Gantt Chart

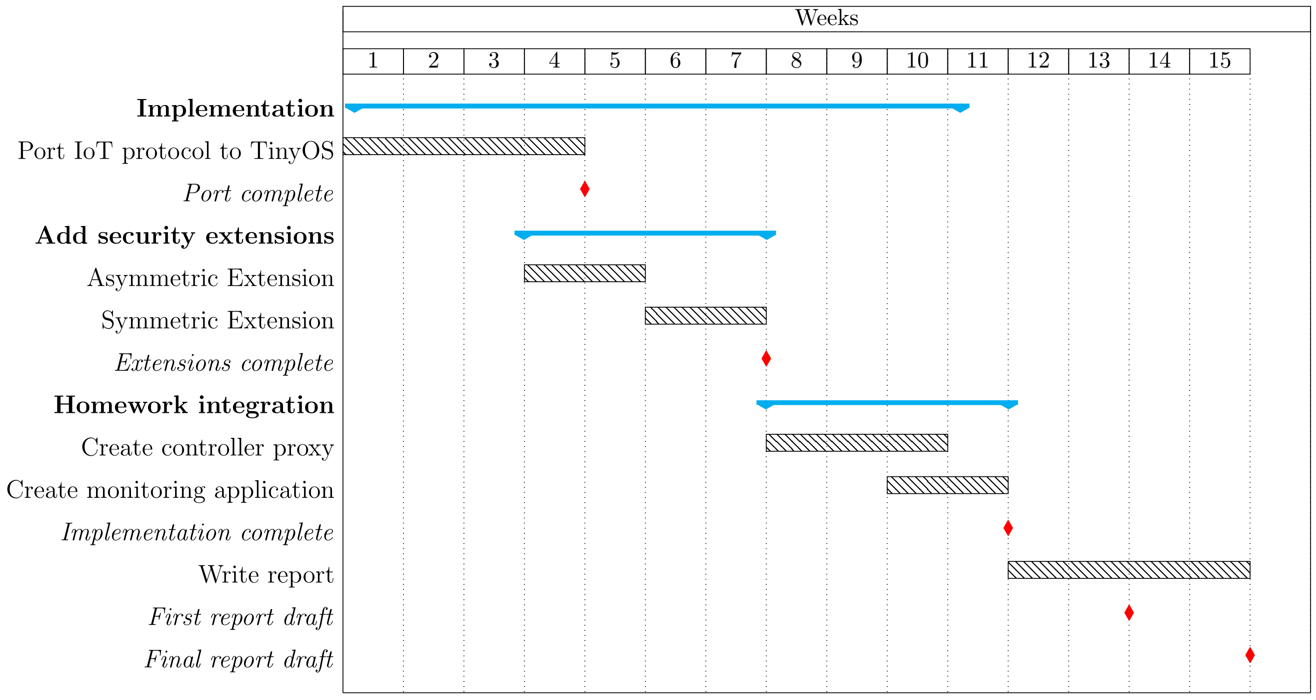

Figure 4: Work plan timeline

[13]As discussed in Security and Cryptography and provided in lecture notes.

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] ECC implementation for Contiki. http://score.ucsc.lk/projects/contikiecc. Accessed: 21/03/2013.

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

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

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

[7] Linux worm targeted at Internet of Things. http://www.symantec.com/connect/blogs/linux-worm-targeting-hidden-devices. Accessed: 12/12/2013.

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

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

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

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

[12] 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.

[13] 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.

[14] 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.

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

[16] 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.

[17] A. Koliousis and J. Sventek. Debs grand challenge: Glasgow automata illustrated. In *6th ACM International Conference on Distributed Event-Based Systems*, July 2012.

[18] 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.

[19] 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.

[20] ArjenK. Lenstra and EricR. Verheul. Selecting cryptographic key sizes. In Hideki Imai and Yuliang Zheng, editors, *Public Key Cryptography*, volume 1751 of *Lecture Notes in Computer Science*, pages 446–465. Springer Berlin Heidelberg, 2000.

[21] 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.

[22] 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.

[23] R. Mortier, T. Rodden, T. Lodge, D. McAuley, C. Rotsos, A.W. Moore, A. Koliousis, and J. Sventek. Control and understanding: Owning your home network. In *Communication Systems and Networks (COMSNETS), 2012 Fourth International Conference on*, pages 1–10, 2012.

[24] Richard Mortier, Tom Rodden, Peter Tolmie, Tom Lodge, Robert Spencer, Andy crabtree, Joe Sventek, and Alexandros Koliousis. Homework: Putting interaction into the infrastructure. In *Proceedings of the 25th Annual ACM Symposium on User Interface Software and Technology*, UIST '12, pages 197–206, New York, NY, USA, 2012. ACM.

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

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

[27] J. Sventek, A. Koliousis, O. Sharma, N. Dulay, D. Pediaditakis, M. Sloman, T. Rodden, T. Lodge, B. Bedwell, K. Glover, and R. Mortier. An information plane architecture

supporting home network management. In *Integrated Network Management (IM), 2011 IFIP/IEEE International Symposium on*, pages 1–8, 2011.

[28] 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.