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 **3**

2 Statement of Problem **4**

2.1 A Scenario 4

2.2 The Problem 4

2.2.1 Security 4

2.2.2 Integration 5

2.3 Intranet of Things vs Internet of Things 5

2.4 Project Outcomes 5

2.5 Structure 5

3 Literature Review **6**

3.1 State-of-the-art IoT Protocols 6

3.1.1 Smart Things 6

3.1.2 CoRE CoAP HIP-DEX 7

3.1.3 KNoT 7

3.2 Home networking 7

3.3 WSN Security - Symmetric Cryptography 7

3.3.1 TinySec 8

3.3.2 MiniSec 9

3.3.3 ContikiSec 10

3.4 Key Distribution Problem 10

3.4.1 USB interface connectivity 10

3.4.2 Message in a Bottle 11

3.4.3 Asymmetric Cryptography 11

3.5 WSN Security - Asymmetric Cryptography 11

3.5.1 TinyECC 12
3.5.2 Certificates and Public Key Infrastructures 12
3.6 Other Works 12
3.6.1 MQTT 12
3.6.2 IETF Work 12
3.7 Problems with previous work 12

4 Proposed Approach **13**
4.1 Solving the Problem 13
4.2 Security Architecture 13
4.2.1 Symmetric Key Cryptography 13
4.2.2 Asymmetric Key Cryptography 13
4.3 Implementation of IoT Protocol on TinyOS 13
4.4 Integration of IoT with Smart Home Router 13

5 Work Plan **13**

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 advent use of smart phones and introduction of 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], [22]), 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 all of them and the ecosystems [11], [10]. Due to the sheer number and diversity of these devices, many of which will provide overlapping services and functionality, problems arise in how to ensure these devices not only play nicely together but also ensuring 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 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 a suitable protocol in order to meet the specific needs of a network of Things [17]. Thus, a new protocol was designed and implemented, which could not only run on even 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, many 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 [17] 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 such rich and sensitive data can be gathered about the user and their home from Thing's, it becomes extremely important to not only protect the data from unauthorised persons, but also protect the network itself. The IoT protocol needs to be sufficiently secured to prevent eavesdropping of the transmitted data and injection of false events by perpetrators masquerading as sanctioned participants in the network.

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, see [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 possible aid 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 the washing machine finishing a load. 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 the 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 the 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.

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

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

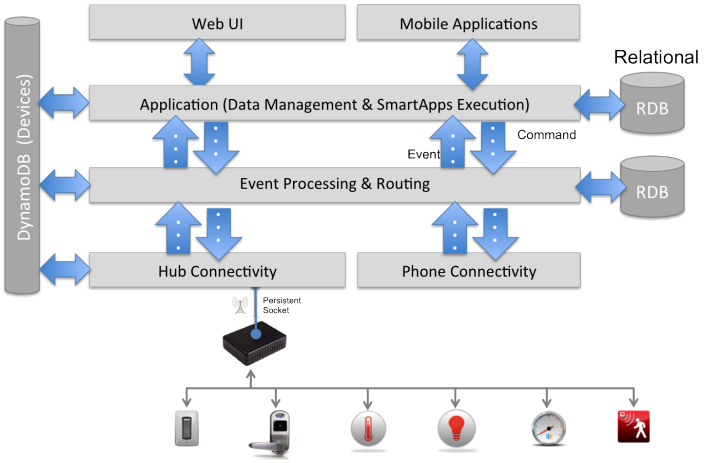

Figure 1: Smart Things - Cloud approach[1]

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 purchasing 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 show 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 with. 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

3.3 WSN Security - Symmetric Cryptography

Much research has been carried out to secure WSNs, to ensure data within the network can't be compromised, which could lead to an negative alteration of academic experiment results or cause a leakage 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.

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 apparent drawbacks of 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(2 byte counter) byte IV 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 e.g. to nodes send the same data event and both happen to have the same counter value,

but differ in source (src), so the IV is different, therefore preserving the security.

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 could 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 to 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 pretend to be any other node in the network, making it difficult to protect, never mind counter against.

3.3.2 MiniSec

MiniSec was created to tackle several problems apparent in the then current WSN security protocols, TinySec and Zigbee. 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 no replay prevention the overall security of it was deemed insufficient to protect a WSN. A commercial alternative, Zigbee, exhibits significantly higher security, but does so at the cost of high 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. In the paper, the authors chose 3 as n. 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 removing the need for the extra byte to store those bits.[2]

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 would be dropped.

3.3.3 ContikiSec

[12] Discuss port of security to contiki, comparisons of different encryption algorithms, use of OCB.

3.4 Key Distribution Problem

Whilst previous work has demonstrated it is feasible to secure a wireless sensor 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 at all 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 another method in 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, as they now require additional components, but it can also increase the physical size and complexity of using them.

[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 benefit of the proposed reduced state based counter.

3.4.2 Message in a Bottle

Tried to solve using small Faraday cages to encapsulate the keying devices...[16]

3.4.3 Asymmetric Cryptography

Another possible alternative for distributing the symmetric shared keys, would be to use asymmetric cryptography to establish a secure channel between the two participants and then exchange the key over this secure channel. The following section describes this in more detail and also discusses some previous work and existing solutions.

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 on 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 they 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 only 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 an efficient implementation of an asymmetric cryptography algorithm for WSN, as well as a possible solution to the authentication problem.

3.5.1 TinyECC

[18] A public key crypto implementation of Elliptic Curve cryptography, supporting higher effective security with smaller keys (than RSA).

3.5.2 Certificates and Public Key Infrastructures

Discuss the use of certificates and certificate authority to pass trust up the chain.

3.6 Other Works

3.6.1 MQTT

3.6.2 IETF Work

[13, 22]

3.7 Problems with previous work

Typical wireless sensor network security has considered and used symmetric cryptography[14, 19, 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 i.e. PC.

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[18]; Therefore allowing pairs of nodes in a network to dynamically create shared keys. Whilst there is some improvement in performance, it still considerably slower and consumes far more resources than symmetric cryptography, as shown in table **??**.

	Symmetric(MiniSec)	Asymmetric(TinyECC)
ROM size	3KB	15.65KB
RAM size	300B	1.8KB
Encrypt	<30ms/pkt	3.2s/pkt
Decrypt	<30ms/pkt	2.1s/pkt

Table 1: Crypto implementation sizes and performances on TelosB mote

There has been significant research into WSN security [14, 19, 12, 18, 16, 21, 15, 20], covering both symmetric and asymmetric cryptography for various WSN platforms. However, in networks where symmetric cryptography was used, deployment of shared encryption keys was either not described or burned to ROM[14, 19, 12] prior to a node's deployment. In the case of asymmetric cryptography, research has just proven that it was viable on WSN platforms[18]. Little has been done to present a fully deployable and dynamic secure WSN that is suitable for the home.

4 Proposed Approach

4.1 Solving the Problem

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] Chris Karlof and David Wagner. Secure routing in wireless sensor networks: attacks and countermeasures. *Ad Hoc Networks*, 1(2-3):293–315, Sep 2003.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] Adrian Perrig, John Stankovic, and David Wagner. Security in wireless sensor networks. volume 47, pages 53–57, New York, NY, USA, June 2004. ACM.

[21] Pawani Porambage, Pardeep Kumar, Corinna Schmitt, Andrei Gurtov, and Mika Ylianttila. Certificate-based pairwise key establishment protocol for wireless sensor networks.

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