University of Glasgow | School of Computing Science

Integrating a Secure Intranet of Things into a Smart Home Router

Fergus Leahy (0908622L)

May 19, 2014

ABSTRACT

The Internet of Things (IoT) is a rapidly growing area of computing, with manufacturers rushing to market and standards bodies sticking to tried and tested architectures. However, when considering an Internet of Things within the home, many of these activities are ill-thought out, inappropriate and even possibly physically dangerous, as evidenced by various attacks [1, 5]. We introduce the concept of an Intranet of Things (iot) by defining a suitable model and device roles to represent the ecosystem of devices typically found within an Intranet of Things. We subsequently present a secure iot protocol implemented on the TelosB mote for TinyOS, which enables users to easily and securely add new Things to the network with minimal configuration, as well as protect the devices, data and user privacy against common attacks. Additionally, we demonstrate integrating our iot protocol into the Homework router Cache, a high-performance complex event processing engine, providing a powerful and customisable closed-loop of control for the iot.

1. INTRODUCTION

The modern home is becoming increasingly filled with a variety of connected devices, each providing myriad different and often overlapping services within the home. More recently, "smart-appliances" and the Internet of Things have begun to enter our homes, attempting to digitize our already existing "dumb-appliances" and objects within the home. As these devices enter our homes, in an often piecemeal fashion, bringing with them their own distinct ecosystems, protocols and standards, the user is faced with the increasingly difficult burden of managing this network of heterogeneous devices. Due to the sheer number and diversity of these devices, problems arise with respect to how these devices co-operate, as well as how to ensure the user's network and information remains secure against new and unanticipated threats.

Many such devices present a severe lack of thoughtful integration into existing networks, infrastructure and the technology into the devices themselves. Existing approaches either integrate traditional power-hungry 802.11 WIFI chipsets or run 6LowPan, a scaled down IPv6 for low-power 802.15.4 radios. However, both of these approaches can severely limit the lifetime of a battery-powered device. With TCP/IP built into these devices, naturally, many of them offer services directly to the Internet [4, 13, 30] or connect to a cloud service [8,11]. Whilst this reaps the benefits of anywhere access and scalable compute power/storage, it opens up multiple points of failure by relying on such connectivity to the Internet or Cloud, including reliability, security, privacy and data-ownership; such an approach has already resulted in attacks similar to that of Stuxnet [5].

In this paper we present a solution for creating a low-power and secure Intranet of Things (iot) integrated into the Homework router platform, by designing an entirely new protocol with a realistic iot model, described in section 3.3, in mind and taking advantage of a combination of symmetric and asymmetric cryptographic algorithms, as well as primitives to enable efficient, secure and authenticated entry into a new network with minimal user configuration. By integrating the iot protocol into the Homework router, we are able to create a powerful, customisable and scalable closed-loop of control within the local network, without relying on connectivity to the Internet or Cloud [3, 20].

The main contributions of this paper are:

- An Intranet of Things model describing three equivalence classes of Things and their operational semantics within the network.
- The design and implementation of a secure, fit-for-purpose iot protocol, implemented over TinyOS running on TeloB motes.
- A user-friendly security scheme for low-power Things, using symmetric and asymmetric cryptography to enable new trusted devices to enter the network securely while minimising user interaction and configuration.
- A controller implementation integrated into the Homework Cache, provided by a modified controller role for TinyOS and a Java-based proxy communicating between the iot and Homework Cache. Additionally, to "close the loop", a sample Homework automaton has been created to demonstrate the integration.

2. MOTIVATION

2.1 A Secure Intranet of Things Protocol

In recent years the Internet of Things has resurfaced, from its beginnings as RFID-tracked products in a warehouse stock floor, transformed into embedded low-power wireless devices in everyday objects around us, under the guise of "smart-appliances". Many manufacturers have rushed to market with new smart devices to replace our old unintelligent ones, but in so doing, they've integrated these devices and services into our home networks and the Internet without much thought or consideration for the device's and network's power, reliability, security and privacy.

The first problem this paper addresses is that many current approaches have integrated expensive wireless chipsets

and/or heavyweight/inefficient protocols (IPv6/6Lowpan, TCP, MQTT [4], [12]) into Things, or simply manipulated Things to fit into the traditional RESTful client-server architecture [15]. However, given that many of these Things are expected to run unattended on battery-power for as long as possible, these approaches are largely inappropriate. Therefore, it's necessary to engineer a new, lightweight, power-efficient protocol, specifically tailored for the typical iot model discussed later in section 3, which also scales well as the number of devices inevitably increases.

Our model and protocol targets a network of Things within the home, with devices communicating and forming the closed-loop of control locally, as opposed to in the Cloud; thus, in lieu of an Internet of Things (IoT), a more suitable name, an Intranet of Things (iot), will be used.

2.2 Security

In the context of an iot, security is of paramount importance due to the rich and sensitive nature of the data that sensors gather, as well as the level of control available from actuators. In a similar fashion to how Stuxnet was directly targeted at machine-to-machine (M2M) networks [10], recently there have been several significant attacks targeted towards IoT devices [1, 5]. For example, over half a million users' Belkin IoT devices and home networks were vulnerable to attackers being able to remote control and monitor these devices, as well as allowing attackers access to the home network [1]. This resulted in risks ranging from electricity wastage, to presence monitoring and, in severe cases, possibilities of home fires being caused by the appliances attached to these devices.

Thus, the second problem this paper attempts to address is securing an iot to ensure the user's devices, network and data are protected against possible attacks and also allow new "approved" devices to join the network with minimal user effort and interaction. Due to the scale of an iot network, containing potentially tens if not hundreds of devices, this needs to be possible without the user being required to manually configure each device with a security key.

Deployable security in wireless sensor networks continues to be a significant problem for several reasons. Firstly, power is a major concern in wireless sensor networks (WSN); thus, running expensive conventional cryptography algorithms in order to keep transmitted data secret can be detrimental to the lifetime of a node. Secondly, WSN nodes are often extremely constrained in terms of memory (ROM/RAM), requiring cryptography algorithms to fit within extreme size constraints and is especially problematic when trying to reduce computational load by storing pre-computed tables. Lastly, being able to dynamically add new nodes to a network post-deployment, as well as (re)distribute keys for the network, enables the network to scale, replace failed nodes and protect against attackers. Previous work has demonstrated various solutions to the first and second part of this problem [19, 23, 24], but have largely ignored the third part, assuming that keys or shared secrets are distributed at install time.

The types of attack to which IoT networks are vulnerable are:

- Eavesdropping - An attacker can overhear messages broadcast by nodes in the network, using the information learned to potentially perform a physical attack (when house doors are unlocked), or log for later analysis (activity monitoring).
- Masquerading - An attacker masquerades as a legitimate node within the network and is able to inject packets as well as abuse the closed-loop of control within the network with potentially dangerous consequences e.g., transmit cold temperature readings to trick the boiler to increase the temperature.
- Man-in-the-middle - An attacker intercepts communications between two nodes and is able to overhear, manipulate and inject packets without either node detecting it.
- Replay - An attacker records messages between nodes in the network and rebroadcasts them in an attempt to manipulate the network. This attack can be performed even when packets are encrypted.
- Denial of service - An attacker abuses the resources available on a node by overwhelming it with expensive operations e.g., verifying a certificate.
- Node Capture - An attacker captures a node and can retrieve data/keys stored on the device, potentially compromising the security of the network.

2.3 Integration

Without integrating the iot into a more powerful and user-accessible platform, the iot risks becoming static and inflexible, incapable of meeting the needs of the constantly changing and evolving lives of the users. To tackle this problem, most existing activities have pursued Cloud integration [2, 8, 11]; as described section 1, whilst this approach provides benefits such as scalable compute power/storage and anywhere-access, reliability and security become serious issues, especially in cases where Things become part of security systems or health-care devices.

Similarly, within the home networking domain, they are many issues regarding integration and usability of commodity home network routers, in which, despite their widespread adoption, the majority of routers are still not designed for the average user, making controlling and managing a home network a difficult and neglected task. In an attempt to solve this, the Homework project redesigned the home router from the ground up, re-engineering the protocols, models and architectures to truly suit the target users and the home environment [3, 25]. A key component of the platform is the Cache [32], a high-performance complex event processing engine, enabling the router to perform closed-loop control over the network and events which occur. On top of this, users can create customised rules to alert them of certain usage patterns occurring within the network.

Thus, the third and last problem this paper addresses is integrating the aforementioned secure iot protocol into the Homework Cache, to utilise the event processing engine for "closing the loop" in an iot.

3. DESIGN

The design of our secure iot protocol largely consists of three parts, the core iot protocol, which encapsulates the iot model, the security protocol, which ensures secure packet transport and entry into the network, and the Cache integration, enabling the closed-loop of control within the iot.

This section will first describe our assumptions, followed by the design of the iot protocol, security scheme and Cache integration, respectively.

3.1 Assumptions

In designing the iot protocol, the following assumptions were made:

- The controller role is implemented on a PC and has sufficient resources available to process and store a large number of public keys for nodes within the network.
- Typical raw sensor data is non-critical, ephemeral and frequent enough such that occasional packet loss is acceptable. Additionally, we expect the network to be sufficiently well engineered, such that a sufficient percentage of packets are received to provide utility.
- The controller has sufficient resources available to generate and store secure pair-wise session keys for each node in the network.
- Things are typically retrofit into existing homes and structures, thus connecting these devices directly to power lines is impractical in most cases; instead Things must rely on battery-power and are expected to run for as long as possible.
- The network is deployed over a geographically small enough area, the home, such that it's unnecessary to perform packet gathering and compression between the source sensor and sink controller. Thus, intermediary nodes need not store keys for each of the surrounding nodes and can, if necessary, forward any encrypted packets received towards their destination.
- The majority of nodes deployed in an iot are situated within a secure property e.g., the home or office; due to the increased cost of tamper-proofing nodes, node capture is not deemed to be a significant issue and will not be mitigated against in this security protocol.

3.2 Hardware and Software Platforms

Whilst many newer and more powerful sensor platforms are available today, much of the existing work for WSNs and WSN security has been developed for the TelosB mote, a 8Mhz TI MSP430 microcontroller with 48KB ROM and 10KB RAM due to its low-power operation and TinyOS support. TinyOS[1] is a lightweight operating system designed for embedded devices in WSNs and has strong support in academia, with many libraries and tools available.

3.3 iot Protocol

In the design of an improved and fit-for-purpose iot protocol, we attempt to minimise radio usage in order to maximise the lifetime of a battery-powered Thing, sacrificing reliability where appropriate, whilst still ensuring robust and predictable operation. In this section, we first present the architectural model of a typical iot network, followed by the key design areas for achieving these goals within the protocol: reliability, integrity, liveness, minimising packet headers and ensuring correct operational semantics (at-most-once).

[1] `http://www.tinyos.net/`

iot Model

Within the Intranet of Things ecosystem, three distinct equivalence classes of devices exist:

- **Sensors**, devices which sense and receive input from the real world or another service, such as temperature sensors, presence sensors etc. Data from these devices is typically periodic, thus non-critical, only requiring integrity and not complete reliability.
- **Actuators**, devices which represent physical or digital outputs, such as displays, lights and functions on appliances, that present an RPC interface for a controller to call. Functions provided by these devices are expected to be carried out once and only once when requested, thus reliability and acknowledgements are needed to ensure reliable operation.
- **Controllers**, typically one device such as a PC or home router, which queries, connects and manages the network of sensors and actuators, orchestrating the closed-loop of interaction between sensor inputs and actuator outputs based on user-specified actions or rules.

Figure 1 presents the iot model with a variety of typical Things connected to the controller, with matching colours representing the closed-loop of control between each of the Things in the network. As discussed below, reliability varies with respect to the typical operations a device performs, which is portrayed with the solid and dashed directed lines.

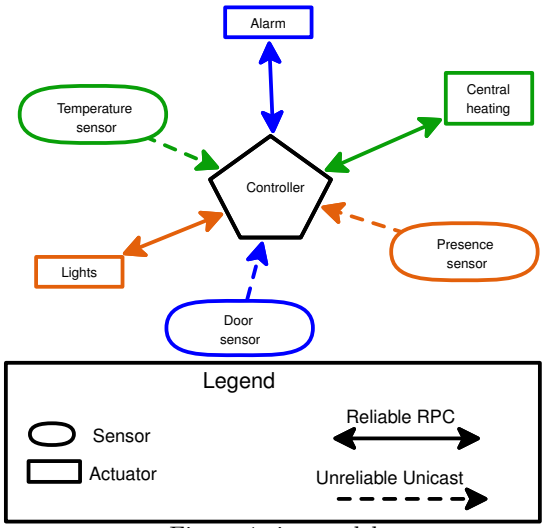

Figure 1: iot model

Reliability

Typical Things within the iot are low-power embedded devices, relying on wireless communication which is inherently unreliable. Thus, it's necessary to implement reliability on top in order to ensure correct operation. However, whilst other approaches have simply chosen to run over TCP, which provides a fully reliable connection, our design opts for selective reliability on top of the unreliable packet stream. This enables the protocol to ensure critical packets are sent reliably, such as connection handshakes, as well as allow devices to save power and sleep after a transmission instead

of waiting for an acknowledgement when the data isn't critical, such as ephemeral sensor readings. Similarly, this also ensures commands sent to actuators from the controller are carried out at-most-once.

Liveness

Ensuring Things within the network are alive and accessible is another key issue, since wireless interference and depleted batteries are a real problem, causing the closed-loop interaction to hang or fail and Things within the network to waste power sending packets to devices that are no longer "out there". To resolve this issue, checks must be performed, using piggybacked messages where possible, to ensure that no extra bandwidth is wasted, such as a sensor periodically requesting a response to a reading. Otherwise standard pingack messages are exchanged.

Integrity

Whilst ensuring reliability of some data is not essential, it is however extremely important that any received data be correct. Values corrupted in transit would appear to the controller as seemingly correct, which could then cause the controller to issue invalid commands to an actuator. To ensure integrity of the data, a checksum of the data, provided by the security protocol, is transmitted with it and compared on the receiving side, verifying that it hasn't been altered in transit.

Protocol data unit

One of the key aims of this protocol is to ensure low energy consumption, which on an embedded devices means transmitting as little as possible over the radio, as radio transmission is the most power-hungry operation for a node. Therefore it was imperative that our protocol adds very little overhead to the already small payload size.

Figure 2 shows the secure protocol data unit layout for the iot protocol. Channels are synonymous to ports found in TCP/UDP and enable devices to hold state for more than one connection and potentially take on more than one role. Sequence numbers ensure packet ordering, at-most-one semantics and protect against possible replay attacks. The command field designates the type of payload each packet holds e.g. sensor reading, handshake ack, actuator command etc. The low order bit of the command field, designated R, is used to indicate that the packet requires an acknowledgement. Additionally, the shaded region shows the encrypted and authenticated bytes, ensuring the sensitive data is kept secret and secure. The MAC field, contains the message authentication code (MAC), used for authenticating the integrity of the encrypted fields.

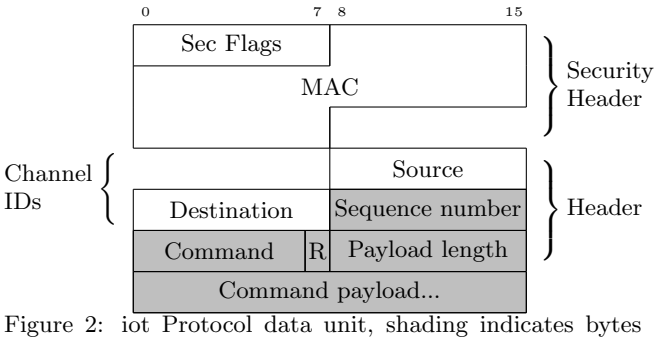

Figure 2: iot Protocol data unit, shading indicates bytes are encrypted and authenticated using a pairwise symmetric key.

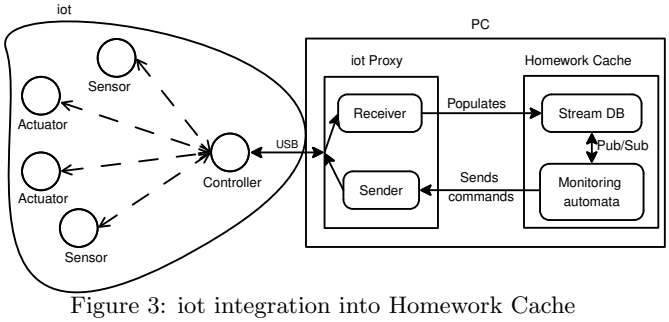

Figure 3: iot integration into Homework Cache

3.4 Integration

In this section we present the architecture design for integrating the iot into the Homework Cache, as well as describe in more detail how the closed-loop of control functions using the Cache and its related concepts.

As shown in figure 3, the integration architecture comprises of 3 main components, the iot network managed by the controller device, the iot proxy and the Homework Cache.

Controller role

In the previously described iot model, the controller role manages the network and orchestrates the closed-loop of control, receiving sensor messages and issuing actuator commands. The current implementation of the controller is implemented on TelosB a mote for TinyOS. Thus, in order for the controller to communicate with the Cache, the TelosB mote is connected to the host PC and provides a bridge into the iot network, allowing a PC-side application to issue commands, via the serial link, for the controller to carry out, and receive responses from the network e.g., sensor messages.

iot Proxy

To allow the controller and Cache to communicate, a suitably designed proxy is necessary. TinyOS provides the necessary libraries to facilitate mote-to-PC communication for a variety of languages, including Java, Python and C. These libraries communicate over the serial link to the mote and translate the incoming packets to the correct format and endianness. Due to the abundance of documentation available, the proposed proxy design is implemented using the Java library. Similarly, the relevant libraries and support are also available for the Cache, which use RPC, in Java.

Within the Java-based proxy, there are two threads:

- A receiver thread which receives incoming packets from the iot network and then inserts the relevant messages formatted as tuples into the Cache, e.g., sensor 2 temperature is 22C.
- A sender thread that registers automata to the Cache and then receives messages from these automata, such as alerts and commands, and forwards them to the controller to be actioned within the network e.g., sensor 2

is too hot, turn down the thermostat.

Homework Cache

The Cache combines both publish/subscribe and stream database concepts, to create a powerful and flexible complex event processing engine. Within the cache, automata, written in the Glasgow Automaton Programming Language (GAPL), are used to detect complex events patterns which occur over the cached streams and relations within the database. As previously mentioned, the proxy registers automata against the cache, after which, the automata subscribe to the necessary topics within the Cache; each time a tuple is inserted into a relevant topic stream in the Cache by the receiver thread, automata subscribed to this topic receive an event containing the tuple. The automaton can then process the tuple, lookup/update/insert tuples into relational tables or streams within the database and then, if necessary, send a relevant event to the sender thread, to be forwarded to the iot network.

This enables the iot closed-loop of control to not only obey simple rules, such as the temperature falling below a set limit, but also allows for complex patterns to be ascertained over longer periods of time, such as detecting when users are home most often and adjusting the heating appropriately before they arrive home. Similarly, because of the dynamic nature of automata, which can be registered and removed freely during run-time, new rules can be added and old ones modified easily, evolving in-sync with the user and their lifestyle.

Automata

Automata, programs written in the GAPL programming language, use memory and control structures exposed by the language, in a similar fashion to C, rather than more traditional declarative languages, such as SQL, which consist of predefined and nested queries. Additionally, automata are not only able to maintain local state, but are also able to interact with state in event streams and relational tables, both locally and remotely. These key concepts enable programmers to perform dynamic and complex pattern matching over high-speed data streams, whilst also maintaining persistent stores of selected data, in the form of relations. The rest of this section will describe a simple automaton designed for testing the iot integration architecture and describe some of GAPL's features in more detail.

Figure 4 shows a basic automaton which reports back to the proxy when a Thing, which senses temperature, has sent a temperature that has exceeded its set limits, previously set by the user via the proxy. In this example, the automaton subscribes to the *Temp* topic by binding it to a local variable t. This enables it to receive an event, containing a Thing ID and temperature reading, every time it is delivered to the *Temp* topic e.g., Thing 2 senses 22C. Similarly, associations allow the automaton to access and modify persistent tables via lookup and insert calls, in this case the *TempLimits* table, containing upper- and lower-bound limits for each temperature sensing Thing. The behavior clause allows the automaton to interact with the aforementioned subscriptions and associations each time a temperature event occurs; upon receiving an event, the automaton attempts to find upper- and lower-bound limits for the event's Thing ID; if found, it checks to see if the event's temperature reading has exceeded the limits and, if so, notifies the process which registered the automaton via a send call, passing with it the related data and a relevant message.

To extend this, a further association could be made to an average temperature table, allowing the automaton to log a Thing's temperature over a period of time, potentially using this data to inform the registered process or manipulate other streams or tables.

```
subscribe t to Temp;
associate limit with TempLimits;
identifier id;
sequence s;
behavior {
  id = Identifier(t.src);
  if (hasEntry(limit, id)) {
    upperLimit = seqElement(lookup(limit, id), 2);
    lowerLimit = seqElement(lookup(limit, id), 3);
    if (t.temp >= upperLimit) {
      s = Sequence(t.src, t.temp, upperLimit);
      send(s, 'temp exceeded upper limit!');
    }
    else if (t.temp <= lowerLimit) {
      s = Sequence(t.src, t.temp, lowerLimit);
      send(s, 'temp below lower limit!');
    }
  }
}
```

Figure 4: Temperature limit automaton

3.5 Security

In this section we present the design of our security protocol, enabling new devices to join the network without the need of a shared secret or any physical interaction with the controller. As discussed later in section 7, there are many existing security protocols available for TinyOS, including TinySec [19] and MiniSec [24] for symmetric cryptography and TinyECC [23] for asymmetric cryptography, as well as a variety of other key distribution schemes [14, 21, 26–28, 31, 33]. The security protocol design we propose does not claim to demonstrate a completely new protocol, but is instead a novel combination of the previously described work, MiniSec and TinyECC, based on a conventional security algorithm, Transport Layer Security (TLS), commonly used in desktop machines today.

The key problem with previous approaches to WSN security is key distribution, in which approaches either assume that a network-specific shared secret, key or ID is preinstalled onto some or all of the nodes and thus distribution is not an issue, or the structure of the network is planned and predetermined, with designated high-power nodes to aid in distribution, see section 7. However, this is impractical for networks containing many tens of nodes, entering in a piecemeal fashion and for administration by a novice user expecting device installation to be simple. Thus, a secure key distribution method must be devised to ensure new nodes can join the network without the need to be reprogrammed.

To solve this problem we propose the security protocol design described below and in figure 5. Our design attempts to alleviate complex physical interactions between devices performed by the user [21] and remove the need to design and configure the network, which for the average home user would be non-trivial. Later sections, 4 and 5, describe the choices made in designing this protocol.

Prior to deployment, nodes will need to be bootstrapped with unique public/private key pairs, a public key certifi-

cate and the CA's public key. In a commercial scenario this would require cooperation with trusted manufacturers to create these keys and certificates. After this the protocol is as follows:

1. Initiate discovery mode on the node; node now accepts and verifies incoming certificate requests.
2. Initiate query on controller; controller sends a broadcast query with its public key certificate.
3. The sensor receives the request with the certificate, verifies its authenticity using the CA public key and, if successful, it sends a response with its own certificate.
4. The controller receives the response with the sensor's certificate, verifies its authenticity using the CA public key and replies with an acknowledgement.
5. The sensor receives the response ack, generates a random nonce value (to protect against replay attacks), then encrypts it with the controller's public key and signs it with its own private key before sending it to the controller.
6. The controller receives the signed and encrypted nonce. It first verifies the signature; if invalid it can either request the nonce again or abort the key exchange and waste no further resources. Otherwise it then decrypts the nonce value. The controller then generates a new symmetric key, encrypts it and the nonce with the sensor's public key and signs with its own private key.
7. The sensor receives the signed and encrypted key and nonce. It first verifies the signature; if invalid it can either request the key response again or abort the key exchange. Otherwise it then decrypts the key and nonce value, verifying that the nonce is equal to the one it sent earlier, confirming the key response has not been replayed. Using the symmetric key, the sensor then initialises the symmetric encryption code and then sends a handover message, encrypted using the symmetric key, to the controller to signal it has received the key and is ready to communicate using it.
8. The controller then receives the symmetric handover message, decrypting it to confirm its authenticity, and replies with an acknowledgement, also encrypted using the symmetric key. The two devices can now communicate securely using the symmetric key.

4. ASYMMETRIC SECURITY

4.1 TinyECC

To enable secure key distribution, asymmetric cryptography provided by elliptic curve cryptography (ECC) using TinyECC [23] is used. Whilst TinyECC demonstrates the feasibility of using software-based public key cryptography via ECC on low-power microcontrollers, it is still extremely expensive in respect to code size (>15KB), RAM utilisation (3KB) and operational speed, taking up to 5 seconds per operation. Therefore, it's necessary to use it as little as possible and switch over to the far cheaper symmetric key cryptography to keep the connection secure for the rest of its lifetime, in which encrypt and decrypt operations take <2ms.

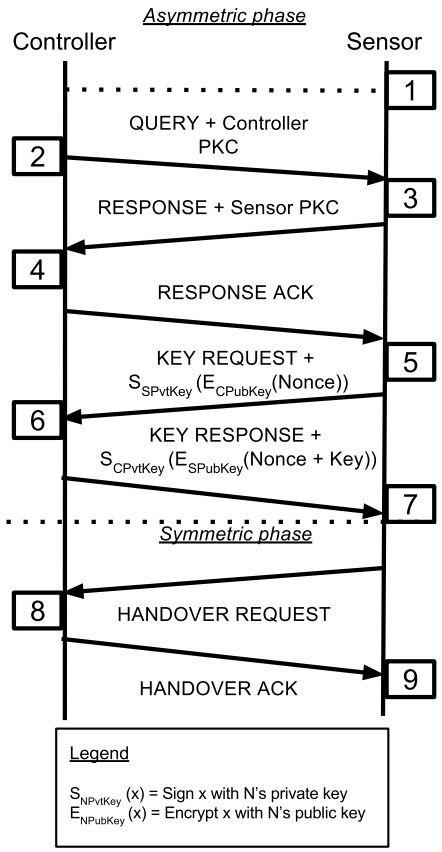

Figure 5: Security protocol sequence diagram

4.2 Denial of service protection

Because of the significant cost in verifying a certificate, it would be possible for an attacker to perform a denial of service attack on a node by sending it fake certificates. To reduce the severity of this attack, Things in the network will only accept and process asymmetric packets during a set window of time after the user presses a button on the device. This also provides demonstrative identification for the user, enabling the user to physically control and understand which devices are communicating, similar to WIFI protected setup (WPS) built into commodity home routers.

4.3 Man-in-the-middle protection

However, using TinyECC alone will not ensure authentication and won't protect against man-in-the-middle attacks. To perform authentication we propose the use of an offline trusted certificate authority (CA) to create public key certificates for nodes. These certificates are then installed on the nodes at compile time along with the node's private/public key pair and the CA's public key. This enables new nodes to have their public key certificate authenticated by other nodes using the CA's public key, before the node's respective public keys are used to exchange the key. This allows nodes to join any network without a previously bootstrapped shared secret symmetric key, unlike other pre-deployment approaches which would require re-bootstrapping the node with the new shared secret symmetric key each time the node joined a new network.

After the nodes have verified each other's authenticity, they are then able to exchange the symmetric key in secret using asymmetric encryption and decryption, followed by handing the connection over to use just symmetric cryptography.

4.4 Optimisations

To enable ECC to operate within reasonable time bounds on microcontroller platforms, such as the TelosB mote, various optimisations were made in TinyECC, which uses precomputed tables and values created at initialisation to speed up the cryptographic operations. When using TinyECC to perform encryption and authentication, the state computed by these operations overlaps, requiring re-initialisation between operations. To reduce this overhead, we implemented state saving and switching, requiring initialisation to be performed only once for each encryption and authentication module, saving valuable seconds over the entire key exchange. The benefits are further detailed in the evaluation in section 6.

4.5 Asymmetric costs

Whilst the overarching aim of our iot protocol is to minimise transmission and save as much power as possible, due to the use of asymmetric cryptography for public keys and certificates in which keys and certificates are 40bytes and 20bytes, respectively, large packet sizes are unavoidable. Similarly, whilst TinyECC makes public key computations feasible on embedded devices, combining them in our proposed protocol to provide both authentication and encryption over an insecure channel for the key exchange results in a one-time high cost, taking less than a minute to complete. However, because it's only performed once, the high initial cost is amortised over the lifetime of the node.

5. SYMMETRIC SECURITY

5.1 Block cipher algorithm and mode of operation

After the handover has occurred, symmetric cryptography is used to keep the connection secure for the duration of its lifetime. The symmetric cryptography used is based on the MiniSec implementation ported from Rogoway et al's Offset Code Book mode of operation (OCB) [29] and Law et al's Skipjack block cipher algorithm [22] implementations. However, due to the inadequacy of various parts of the extended implementation by MiniSec [6], including being written for TinyOS 1.0, allowing only one encrypted connection per node, various memory allocation bugs and inefficient initialisation vector (IV) synchronisation, much of the code was rewritten to work correctly and more efficiently for TinyOS 2.1.

5.2 Man-in-the-middle and Masquerading protection

Similar to the asymmetric component of the protocol, to prevent man-in-the-middle and masquerading attacks, authentication is necessary, ensuring that a message sent from a Thing in the network is genuine and hasn't been tampered with in transit. To do this a message authentication code (MAC) block needs to be generated from the encrypted data. Unlike cipher block chaining (CBC) and CBC message authentication code (CBC-MAC), which encrypt/decrypt and authenticate the data independently using two unique keys, we use the offset code book (OCB) mode of operation for block ciphering, enabling both encryption/decryption and authentication to be completed in one pass over the data using just one key, realising a slight decrease in time [14] and significant saving in space used for storing the key.

5.3 Semantic security

In order to ensure semantic security, in which duplicate packets don't encrypt to the same ciphertext, a 64-bit monotonically increasing IV is used for each connection. This would normally require the 64-bits IV to be transmitted to the receiving node, which is extremely costly when attached to each packet. However, using the last bits (LB) optimisation coined in MiniSec, we are able to just transmit the last n bits of the counter and with some resynchronisation logic on both ends, withstand packet loss of up to 2^n packets before resynchronisation occurs. In our protocol we chose to send the last 6 bits of the counter, due to sharing the byte with another field. In the case of the IV wrapping around, to remain semantically secure the controller only needs to generate a new symmetric key and distribute it before the wrap occurs.

When MiniSec was developed, Skipjack was chosen as the underlying block-cipher due to its efficient implementation and low memory footprint [22]. However, Skipjack uses a maximum key length of 80-bits, which as of 2010 NIST classifies as insecure for long term use [7]. As a temporary measure, the controller can choose to re-issue a new session key for each node after an arbitrary period of time.

6. EVALUATION

To evaluate our proposed protocol we observed several key attributes related to our initial requirements of an efficient, lightweight protocol that attempts to minimise radio transmission. The rest of this section will discuss the iot implementation size for the various roles, followed by the packet overheads compared to other existing protocols, the key exchange timings with the related optimisations to ECC and lastly, the overall end-to-end system performance.

Things, specifically sensors and actuators, are expected to run on low-power constrained devices, thus it's paramount to ensure an iot protocol not only fits within the constraints of a typical Thing, but that it also leaves adequate space for applications to be built on top of it. Table 1 shows the implementation size, ROM and static RAM, across the three roles. In the current implementation the actuator demonstrates the lowest overhead with only the iot protocol and LED library linked into the TinyOS compile, leaving approximately 13KB ROM and 4KB RAM for applications. Given that actuators and sensors are only expected to compute very little, this should provide significant space for applications, except, perhaps, in the case when local filtering is necessary; this is demonstrated by the sensor role which shows a significant increase in the ROM size due to the inclusion of the temperature libraries for TinyOS; however, it is still well within the constraints of the environment. Lastly, the controller role demonstrates the largest size, due to the extra processing and state involved in managing the network. Unlike the sensor and actuator, the controller is not definitively bound by the same constraints, and is expected to run on a more powerful device. Whilst old, the TelosB mote still represents a middle ground for embedded devices in terms

of ROM and RAM, compared to the popular Arduino platform sporting only 32KB ROM and 1-4KB RAM[2]. However, recent MSP430 and ARM microprocessors have significantly increased amounts of storage available, up to 512KB ROM and 64KB RAM.[3]

The most important requirement of our protocol is to minimise radio transmission; in order to ensure this, we not only try to decrease the number of packets sent, but also the size of the packets. In table 2 we compare our packet header overheads against the popular and one of the most significant IoT protocols, 6LowPan combined with the security extension, AES-CCM. For traditional 802.15.4 radio, the maximum advised packet size is 128-bytes. 6LowPan, a minified IPv6 protocol, compresses its normally 128-bit addresses to 64- or 16-bit addresses in order to reduce its overhead. Within a smaller number of bytes compared to an un-encrypted 6LowPan packet with 16-bit addresses, our proposed protocol is able to also ensure that a packet is both secure and authenticated, adding only 4 bytes of overhead to an un-encrypted packet for the MAC. In the current implementation there is still room for possible improvement, as demonstrated by MiniSec which discussed removing the group and crc fields from the TinyOS packet, reducing the overhead by a further 3 bytes.

In order to create a user-friendly, dynamic and flexible key distribution scheme, we opted to use public-key cryptography with TinyECC. In section 4, we argued that whilst TinyECC still exhibits a considerable cost in terms of storage, speed and overhead, the full key exchange only has to be performed once and is amortised over the lifetime of the node; subsequent key exchanges could use the existing symmetric-secured channel. Table 3 shows the average time for switching between modes and the overall time for the entire key exchange outlined in section 4. TinyECC uses precomputed tables and values based on the public-key to be used for an operation, thus when using different public-keys multiple times to perform a sequence of encrypted and authenticated actions, such as in the proposed iot protocol, these tables need to be recomputed each time in order for them to be effective; this adds a significant overhead of around 20 seconds between the communicating nodes. To reduce this, we added an optimisation to save previously initialised state, thus removing the need to recompute the state for a given public-key the next time it is used; this gives us a fully authenticated and secure key exchange taking just under a minute to complete, split between the two nodes. Because the ECC computations are split between the two nodes, porting the controller role to a more powerful platform would significantly reduce the exchange time, further minimising the time the sensor/actuator node needs to power and listen on the radio.

Lastly, table 4 presents a break-down of the system performance, providing average timings and their standard deviations for each part of the system. Rows *a* and *b* show the average timings for encrypt and decrypt operations of 4 bytes, including header fields and a 1 byte payload (e.g., sensor reading); contrasting this to row *c*, which presents the average time for an RPC call (consisting of two messages, each with 4 bytes encrypted), shows that the encryption/decryption operations only add around 8ms of overhead for each message, with the sending/receiving of a message averaging 26ms. Row *d* shows the length of time taken for the proxy to publish a sensor reading to the cache, followed a subsequent send() call made by a subscribed automaton which issues a command back to the proxy, based on the received reading. Row *e* includes *c*, with the addition of the sensor reading now propagating from the controller through *e* and back, via the serial link; similar to *c*, the serial link adds a significant overhead, taking around 19ms for a message to pass from the controller to the proxy. Lastly, to capture the end-to-end time, from sensing to actuation, row *f* combines rows *c* and *e*, in which the RPC is split across the sensor→controller and controller→actuator messages. Thus, the closed-loop of control takes 109ms from sensing to actuation[4]. Due to the non-time-critical nature of an iot, providing sub 100ms response-times isn't necessary, thus, the overall performance is sufficient to provide a suitably real-time iot.

As described in section 3.3, ensuring reliability, for some messages, and liveness, for all connected Things, is important for the iot to operate correctly and responsively. In the case of sensors, readings are non-critical and expected to occur frequent enough such that the iot won't be affected by occasional packet loss; thus, in the case of packet loss the closed-loop of control is only potentially delayed, if the reading isn't redundant, by the sensing frequency. However, in the case of the actuator, commands issued by the controller are sent reliably. Thus, if a packet is dropped, retries occur after a set period, exponentially increasing until the command is either acknowledged, or the controller has deemed the actuator is completely unresponsive and has potentially died. In the current design, the sender waits for 40ms for a response before retrying, doubling this wait, either until the receiver replies or the number of attempts reaches 5. The initial wait of 40ms ensures the receiver has sufficient time to decrypt larger encrypted payloads. In the case of a single packet loss, the response time increases to around 253ms; in the worst case of 5 packets lost, the response time increases to 918ms, which still ensures a reasonably responsive performance with respect to the non-critical nature of an iot.

7. RELATED WORK

7.1 IoT protocols

Given the recent surge of popularity for the Internet of Things, many developers, manufacturers and standards bodies have attempted to create a solution for their view of the IoT, each differing greatly and often being proprietary within their own ecosystems.

The IETF Core working group have proposed a new standard, the Constrained Application Protocol (CoAP) [15], which aims to coerce Things within the IoT to fit into the traditional RESTful client-server architecture that is commonplace on the Internet today, removing the need for specialised platforms or applications to access them. The protocol is built to operate on unreliable UDP links, providing support for reliable delivery on top when needed. In an attempt to be power-friendly, services can subscribe to Things,

[2]Arduino Uno - `http://arduino.cc/en/Main/ArduinoBoardUno`

[3]FreeScale ARM - `http://www.freescale.com/webapp/sps/site/taxonomy.jsp?code=KINETIS_L_SERIES`

[4]An additional ack is also sent by the actuator, but this does not have an effect on the response time of the actuation itself.

Secure iot Protocol			
	Controller	**Sensor**	**Actuator**
ROM size	38,178B (80%)	41,016B (85%)	34,988B (73%)
RAM size	7372B (74%)	6,208B (62%)	6,142B (61%)

Table 1: Secure iot protocol size for Controller, Sensor and Actuator roles on the TelosB (8Mhz, 48KB ROM, 10KB RAM)mote.

	TinyOS + Secure iot Protocol	**6LowPan + AES-CCM**
Cleartext header	14B + 6B	25B
Encrypted header	14B + 10B	46B

Table 2: Comparison of payload overhead for our Secure iot (Symmetric security payload) vs 6LowPan with 16-bit addresses + AES-CCM.

	TinyECC	**TinyECC + optimisations**
Authentication to Encryption	1.4s	0s
Encryption to Authentication	2.5s	0s
Asymmetric key exchange	80.4s	57.1s

Table 3: Comparison of the module switch over and overall asymmetric key exchange durations with normal TinyECC and TinyECC with the save state optimisations implemented.

Closed-loop flow	**Mean time**	σ
a) Symmetric encrypt sensor message (4 bytes)	5ms	0ms
b) Symmetric decrypt sensor message (4 bytes)	2.9ms	0.3ms
c) Thing → Thing RPC (symmetric encrypt/decrypt)	68.9ms	5.8ms
d) Proxy → Cache/Automaton → Proxy	1.1ms	0.9ms
e) Controller → Proxy → Cache → Proxy → Controller	40.4ms	1.5ms
f) Sense → Cache → Actuate	109.3ms	6ms

Table 4: A break-down of the steps within the closed-loop flow of control from sensing through to actuating, showing the mean time and standard deviation (σ) for each step of 200 samples.

such as sensors, instead of polling for updates from these devices, allowing them to save power and enter sleep modes. Whilst bringing Things into the client-server architecture eases the adoption of such devices, it forces them to adopt an architecture that simply isn't suited for them considering their low-power constraints, placing restrictions on their availability, reliability and resources. Similarly, it also potentially poses serious security risks if simply plugged-in to the Internet [1].

Building on top of this, the OpenWSN project at UC Berkeley proposes a software stack sitting on top of an 802.15.4 enabled device and below CoAP/HTTP, consisting of 6Lowpan, a compressed IPv6 for embedded devices, combined with RPL, an IPv6 routing protocol for lossy wireless networks and TCP/UDP transport protocols.

As previously discussed, many manufacturers have integrated power-hungry WIFI chipsets and connect the devices directly to the Internet or to their Cloud service [2, 8, 11], without significant consideration for the power and security issues that result. Whilst these devices gain the power and connectivity of the cloud, this couples them directly to it, and increases the chance of failure, due to services going down, limited Internet connectivity and potentially raises issues regarding data ownership and privacy/loss [9]. Additionally, as a result of this direct Internet and Cloud connectivity, devices needlessly waste power and are at a greater risk to security threats [1, 5].

7.2 WSN Security

There have been several different attempts to efficiently secure WSNs [14, 19, 21, 23, 24, 26–28, 31, 33], however none match the iot requirements enumerated in section 3.3.

TinySec [19], a symmetric cryptography library for TinyOS 1.0, was an initial effort on TinyOS to address security, intending to demonstrate that software-based cryptography was possible on constrained devices with minimal power overhead. To achieve this, TinySec was designed around WSNs' extreme resource constraints, taking advantage of some of these constraints, such as the limited networking bandwidth, optimising the security primitives in order to reduce the security overhead added to each packet. One such optimisation was the use of a reduced initialisation vector for the cipher block chaining mode of operation, which combined several of the existing header fields and added a 2 byte counter, ensuring the initialisation vectors wouldn't clash between nodes sending the same packet. Whilst the IV is significantly smaller than conventional security protocols, which would reduce time until IV reuse, TinySec argued that this wasn't an issue, demonstrating that using an average send rate of 1 packet per minute, IV reuse would not occur for 45 days.

MiniSec, another symmetric cryptography library for TinyOS 1.0 and a successor to TinySec, was created to further improve the security provided by TinySec, adding replay prevention and also improving upon the minimal security overheads which TinySec achieved. To achieve the increase in security, MiniSec chose to increase the initialisation vector size from 2 bytes to 8 bytes, however, instead of transmit-

ting the whole 8 bytes, MiniSec sends only the 3 last bits (LB) within the pre-existing packet length field, thus incurring no additional overhead for the IV on top of the normal TinyOS packet. To ensure the LB optimisation works correctly, both communicating nodes must maintain state for the counter and perform counter resynchronisation upon significant packet loss ($>2^3$ packets). This counter state is also used to ensure replay prevention, requiring all received packets to have a higher IV than the local state.[5]

Until recently, asymmetric security, namely RSA, was deemed infeasible on microcontroller platforms, taking on the order of tens of seconds to complete public key operations [23]. However, this is no longer the case with the development of elliptic curve cryptography (ECC), in which not only are public key operations feasible within several seconds, but key lengths are reduced whilst still providing the same level of security as longer keys in more traditional asymmetric algorithms such as RSA.

There are many other complex key distribution schemes which require hierarchical networks of devices [28, 31] and predetermined deployment strategies to enable efficient key distribution. Rahmun et al present a solution for such a hierarchical network [28], consisting of a network of heterogeneous nodes, in which high-resource nodes store node IDs for surrounding low-resource nodes and provide authentication for the key exchange process between low-resource nodes using ECC. However, this scheme requires expensive high-resource nodes that need to have the low-resource nodes' IDs stored ahead of time, and also need to be tamper resistant to protect against node capture, further increasing the cost. Another viable alternative, pairings-based key distribution schemes, such as TinyPBC [26], provide efficient key pairings (<5s) but require nodes to know each other's ID *a priori*, which can prove difficult to do with authentication.

Other proposed key distribution schemes such as TinyPK aim to sacrifice immediate authenticity and security on the mote by only performing the public key operations on the more powerful server side [33]. In contrast, Message-in-a-bottle [21] relies entirely on a portable faraday cage like barrier to allow devices to secretly communicate in the clear. Whilst it achieved the goal of distributing keys, it sacrifices elegance, usability and scalability, by having to manually place devices inside a lead pipe each time a key needs to be issued.

8. CONCLUSION

Our proposed model and protocol demonstrates a thoughtful approach to designing a secure and efficient Intranet of Things. Compared to other approaches which have used power-hungry WIFI or heavyweight protocols, our protocol attempts to minimise radio usage on 802.15.4 platforms, in an effort to prolong the lifetime of battery-powered Things. In light of existing and recent M2M/IoT attacks, our protocol has been secured against a variety of attacks, ensuring the user's data, privacy and, most importantly, home is safe from remote control, monitoring and other malicious activities. The protocol also demonstrates a usable, secure and scalable method for enabling users to add new devices to the network, without the need for network-specific pre-shared secret keys or complex configuration, via the use of public key cryptography, provided by TinyECC.

Additionally, by integrating our iot protocol into the Homework Cache locally, the closed-loop of control can be fully realised, providing the possibility for dynamic and powerful interactions between the user and Things located around the home, without the privacy and reliability risks associated with the Internet and Cloud. The implementation was developed for the TelosB mote running TinyOS, and the source code has been made available online[6].

9. FUTURE WORK

Whilst we have presented a solid foundation for the iot, in the form of a secure and efficient protocol integrated into the Homework platform, there are many possible extensions/improvements. This includes porting the controller role to more powerful hardware, as assumed in section 3.1, such as a PC or router running Homework, porting the sensor and actuator roles to other low-power platforms, and implementing ad-hoc multi-hop routing. These extensions would enable the network to easily scale up to handle hundreds of devices over large and interference-prone areas. Additionally, the current support for customising rules requires programming skill and knowledge about GAPL, thus it will be necessary to implement a user-friendly rule designer, similar to what Homework already uses [17]; or attempt to remove the interface almost entirely, using intelligent agents and machine learning to analyse and predict the user's needs within the home [16,18], removing the need for users from having to perform the difficult task of trying to create the perfect rules or update existing rules as patterns change e.g., a change in weather or timetable. Lastly, the security against brute-force attacks needs to be improved by replacing the now out-dated Skipjack block cipher algorithm with the more secure AES algorithm [7]. However, improving the security will inevitably affect the performance of the protocol due to the increased computational time for encryption and decryption, thus, the reliability measures will need to be adjusted to ensure retries don't occur when the receiver is encrypting/decrypting data.

10. REFERENCES

[1] Belkin IoT WeMo vulnerabilities press release. `http://www.ioactive.com/news-events/IOActive_advisory_BelkinWemo_2014.html#Note_1`. Accessed: 21/03/2014.

[2] Belkin WeMo IoT devices. `www.belkin.com/uk/Products/home-automation/c/wemo-home-automation/`. Accessed: 21/03/2014.

[3] Homework project. `https://www.homenetworks.ac.uk/`. Accessed: 21/03/2013.

[4] IBM MQTT specification. `http://www.ibm.com/developerworks/webservices/library/ws-mqtt/index.html`. Accessed: 21/03/2014.

[5] Linux IoT attack. `http://www.symantec.com/connect/blogs/linux-worm-targeting-hidden-devices`. Accessed: 12/12/2013.

[5]Whilst the MiniSec paper presented an efficient symmetric cryptography protocol design, the corresponding implementation provided at [6] does not appear to function correctly (with various runtime issues) or implement the IV counter using the LB optimisation described.

[6]Source code available from Github: `https://github.com/fergul/Intranet_of_Things`

[6] MiniSec Homepage. https://sparrow.ece.cmu.edu/group/minisec.html. Accessed: 21/03/2014.

[7] NIST: Recommendations for Transitioning the Use of Cryptographic Algorithms and Key Lengths. http://csrc.nist.gov/publications/nistpubs/800-131A/sp800-131A.pdf. Accessed: 21/03/2013.

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

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

[10] Stuxnet attack Symantec dossier. http://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/w32_stuxnet_dossier.pdf. Accessed: 21/03/2014.

[11] Twine IoT Thing. http://supermechanical.com/. Accessed: 21/03/2013.

[12] xAP protocol. http://www.xapautomation.org/. Accessed: 21/03/2013.

[13] Xively, IoT public cloud. https://xively.com/. Accessed: 21/03/2013.

[14] L. Casado and P. Tsigas. Contikisec: A secure network layer for wireless sensor networks under the contiki operating system. In A. JÃÿsang, T. Maseng, and S. 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] D. J. Cook, M. Youngblood, E. O. Heierman III, K. Gopalratnam, S. Rao, A. Litvin, and F. Khawaja. Mavhome: An agent-based smart home. In *Pervasive Computing and Communications, 2003.(PerCom 2003). Proceedings of the First IEEE International Conference on*, pages 521–524. IEEE, 2003.

[17] A. Crabtree, T. Rodden, P. Tolmie, R. Mortier, T. Lodge, P. Brundell, and N. Pantidi. House rules: the collaborative nature of policy in domestic networks. *Personal and Ubiquitous Computing*, pages 1–13, 2014.

[18] X. Hong, C. Nugent, M. Mulvenna, S. McClean, B. Scotney, and S. Devlin. Evidential fusion of sensor data for activity recognition in smart homes. *Pervasive and Mobile Computing*, 5(3):236 – 252, 2009. <ce:title>Pervasive Health and Wellness Management</ce:title>.

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

[20] A. Koliousis and J. Sventek. DEBS Grand Challenge: Glasgow automata illustrated. In *6th ACM International Conference on Distributed Event-Based Systems*, July 2012.

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

[22] Y. W. Law, J. Doumen, and P. Hartel. Survey and benchmark of block ciphers for wireless sensor networks. *ACM Trans. Sen. Netw.*, 2(1):65–93, Feb. 2006.

[23] A. Liu and P. 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.

[24] M. Luk, G. Mezzour, A. Perrig, and V. 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.

[25] R. Mortier, T. Rodden, P. Tolmie, T. Lodge, R. Spencer, A. crabtree, J. Sventek, and A. Koliousis. Homework: Putting interaction into the infrastructure. In *Proc. of the 25th Annual ACM Symposium on User Interface Software and Technology*, UIST '12, pages 197–206, New York, NY, USA, 2012. ACM.

[26] L. B. Oliveira, D. F. Aranha, C. P. GouvÃła, M. Scott, D. F. CÃćmara, J. LÃşpez, and R. Dahab. Tinypbc: Pairings for authenticated identity-based non-interactive key distribution in sensor networks. *Computer Communications*, 34(3):485 – 493, 2011. Special Issue of Computer Communications on Information and Future Communication Security.

[27] P. Porambage, P. Kumar, C. Schmitt, A. Gurtov, and M. Ylianttila. Certificate-based pairwise key establishment protocol for wireless sensor networks.

[28] S. M. Rahman and K. El-Khatib. Private key agreement and secure communication for heterogeneous sensor networks. *Journal of Parallel and Distributed Computing*, 70(8):858 – 870, 2010.

[29] P. Rogaway, M. Bellare, and J. Black. Ocb: A block-cipher mode of operation for efficient authenticated encryption. *ACM Trans. Inf. Syst. Secur.*, 6(3):365–403, Aug. 2003.

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

[31] Q. Shi, N. Zhang, M. Merabti, and K. Kifayat. Resource-efficient authentic key establishment in heterogeneous wireless sensor networks. *Journal of Parallel and Distributed Computing*, 73(2), 2013.

[32] J. Sventek and A. Koliousis. Unification of publish/subscribe systems and stream databases: The impact on complex event processing. In *Proc. of the 13th International Middleware Conference*, Middleware '12, New York, NY, USA, 2012. Springer-Verlag New York, Inc.

[33] R. Watro, D. Kong, S.-f. Cuti, C. Gardiner, C. Lynn, and P. Kruus. Tinypk: Securing sensor networks with public key technology. In *Proc. of the 2Nd ACM Workshop on Security of Ad Hoc and Sensor Networks*, SASN '04, New York, NY, USA, 2004. ACM.

The following paper is a shorter 8-page version of the one above, with a smaller evaluation and without the integration aspects. It was submitted to the IEEE SenseApp 2014 conference on IoT applications.

A Secure Intranet of Things for the Home

Fergus Leahy, Joseph Sventek, Paul Harvey
School of Computing Science
University of Glasgow
Email: fergus.leahy@glasgow.ac.uk, joseph.sventek@glasgow.ac.uk, p.harvey.1@research.gla.ac.uk

***Abstract*—The Internet of Things (IoT) is a rapidly growing area of computing, with manufacturers rushing to market and standards bodies sticking to tried and tested architectures. However, when considering an Internet of Things within the home, many of these activities are ill-thought out, inappropriate and even possibly physically dangerous, as evidenced by various attacks [1], [2]. We introduce the concept of an *Intranet* of Things (iot) by defining a suitable model and device roles to represent the ecosystem of devices typically found within an Intranet of Things. We subsequently present a secure iot protocol implemented on the TelosB mote for TinyOS, which enables users to easily and securely add new Things to the network with minimal configuration, as well as protect the devices, data and user privacy against common attacks.**

I. Introduction

The modern home is becoming increasingly filled with a variety of connected devices, each providing myriad different and often overlapping services within the home. More recently, "smart-appliances" and the Internet of Things have begun to enter our homes, attempting to digitize our already existing "dumb-appliances" and objects within the home. As these devices enter our homes, in an often piecemeal fashion, bringing with them their own distinct ecosystems, protocols and standards, the user is faced with the increasingly difficult burden of managing this network of heterogeneous devices. Due to the sheer number and diversity of these devices, problems arise with respect to how these devices co-operate, as well as how to ensure the user's network and information remains secure against new and unanticipated threats.

Many such devices present a severe lack of thoughtful integration into existing networks, infrastructure and the technology into the devices themselves. Existing approaches either integrate traditional power-hungry 802.11 WIFI chipsets or run 6LowPan, a scaled down IPv6 for low-power 802.15.4 radios. However, both of these approaches can severely limit the lifetime of a battery-powered device. With TCP/IP built into these devices, naturally, many of them offer services directly to the Internet [3]–[5] or connect to a cloud service [6], [7]. Whilst this reaps the benefits of anywhere access and scalable compute power/storage, it opens up multiple points of failure by relying on such connectivity to the Internet or Cloud, including reliability, security, privacy and data-ownership; such an approach has already resulted in attacks similar to that of Stuxnet [2].

In this paper we present a solution for creating a low-power and secure Intranet of Things (iot) for the home, by designing an entirely new protocol with a realistic iot model, described in section III-C, in mind and taking advantage of a combination of symmetric and asymmetric cryptographic algorithms, as well as primitives to enable efficient, secure and authenticated entry into a new network with minimal user configuration.

The main contributions of this paper are:

- An Intranet of Things model describing three equivalence classes of Things and their operational semantics within the network.
- The design and implementation of a secure, fit-for-purpose iot protocol, implemented over TinyOS running on TeloB motes.
- A user-friendly security scheme for low-power Things, using symmetric and asymmetric cryptography to enable new trusted devices to enter the network securely while minimising user interaction and configuration.

II. Motivation

A. A Secure Intranet of Things Protocol

In recent years the Internet of Things has resurfaced, from its beginnings as RFID-tracked products in a warehouse stock floor, transformed into embedded low-power wireless devices in everyday objects around, us under the guise of "smart-appliances". Many manufacturers have rushed to market with new smart devices to replace our old unintelligent ones, but in so doing, they've integrated these devices and services into our home networks and the Internet without much thought or consideration for the device's and network's power, reliability, security and privacy.

The first problem this paper addresses is that many current approaches have integrated expensive wireless chipsets and/or heavyweight/inefficient protocols (IPv6/6Lowpan, TCP, MQTT [3], [8]) into Things, or simply manipulated Things to fit into the traditional RESTful client-server architecture [9]. However, given that many of these Things are expected to run unattended on battery-power for as long as possible, these approaches are largely inappropriate. Therefore, it's necessary to engineer a new, lightweight, power-efficient protocol, specifically tailored for the typical iot model discussed later in section III, which also scales well as the number of devices inevitably increases.

Our model and protocol targets a network of Things within the home, with devices communicating and forming the closed-loop of control locally, as opposed to in the Cloud; thus, in lieu of an Internet of Things (IoT), a more suitable name, an Intranet of Things (iot), will be used.

B. Security

In the context of an iot, security is of paramount importance due to the rich and sensitive nature of the data that sensors

gather, as well as the level of control available from actuators. In a similar fashion to how Stuxnet was directly targeted at machine-to-machine (M2M) networks [10], recently there have been several significant attacks targeted towards IoT devices [1], [2]. For example, over half a million users' Belkin IoT devices and home networks were vulnerable to attackers being able to remote control and monitor these devices, as well as allowing attackers access to the home network [1]. This resulted in risks ranging from electricity wastage, to presence monitoring and, in severe cases, possibilities of home fires being caused by the appliances attached to these devices.

Thus, the second problem this paper attempts to address is securing an iot to ensure the user's devices, network and data are protected against possible attacks and also allow new "approved" devices to join the network with minimal user effort and interaction. Due to the scale of an iot network, containing potentially tens if not hundreds of devices, this needs to be possible without the user being required to manually configure each device with a security key.

Deployable security in wireless sensor networks continues to be a significant problem for several reasons. Firstly, power is a major concern in wireless sensor networks (WSN); thus, running expensive conventional cryptography algorithms in order to keep transmitted data secret can be detrimental to the lifetime of a node. Secondly, WSN nodes are often extremely constrained in terms of memory (ROM/RAM), requiring cryptography algorithms to fit within extreme size constraints and is especially problematic when trying to reduce computational load by storing pre-computed tables. Lastly, being able to dynamically add new nodes to a network post-deployment, as well as (re)distribute keys for the network, enables the network to scale, replace failed nodes and protect against attackers. Previous work has demonstrated various solutions to the first and second part of this problem [11]–[13], but have largely ignored the third part, assuming that keys or shared secrets are distributed at install time.

The types of attack to which IoT networks are vulnerable are:

- Eavesdropping - An attacker can overhear messages broadcast by nodes in the network, using the information learned to potentially perform a physical attack (when house doors are unlocked), or log for later analysis (activity monitoring).
- Masquerading - An attacker masquerades as a legitimate node within the network and is able to inject packets as well as abuse the closed-loop of control within the network with potentially dangerous consequences e.g., transmit cold temperature readings to trick the boiler to increase the temperature.
- Man-in-the-middle - An attacker intercepts communications between two nodes and is able to overhear, manipulate and inject packets without either node detecting it.
- Replay - An attacker records messages between nodes in the network and rebroadcasts them in an attempt to manipulate the network. This attack can be performed even when packets are encrypted.
- Denial of service - An attacker abuses the resources available on a node by overwhelming it with expensive operations e.g., verifying a certificate.
- Node Capture - An attacker captures a node and can retrieve data/keys stored on the device, potentially compromising the security of the network.

III. Design

The design of our secure iot protocol largely consists of two parts, the core iot protocol, which encapsulates the iot model, and the security protocol which ensures secure packet transport and entry into the network. This section will first describe our assumptions, followed by the design of the iot protocol and security scheme, respectively.

A. Assumptions

In designing the iot protocol, the following assumptions were made:

- The controller role is implemented on a PC and has sufficient resources available to process and store a large number of public keys for nodes within the network.
- Typical raw sensor data is non-critical, ephemeral and frequent enough such that occasional packet loss is acceptable. Additionally, we expect the network to be sufficiently well engineered, such that a sufficient percentage of packets are received to provide utility.
- The controller has sufficient resources available to generate and store secure pair-wise session keys for each node in the network.
- Things are typically retrofit into existing homes and structures, thus connecting these devices directly to power lines is impractical in most cases; instead Things must rely on battery-power and are expected to run for as long as possible.
- The network is deployed over a geographically small enough area, the home, such that it's unnecessary to perform packet gathering and compression between the source sensor and sink controller. Thus, intermediary nodes need not store keys for each of the surrounding nodes and can, if necessary, forward any encrypted packets received towards their destination.
- The majority of nodes deployed in an iot are situated within a secure property e.g., the home or office; due to the increased cost of tamper-proofing nodes, node capture is not deemed to be a significant issue and will not be mitigated against in this security protocol.

B. Hardware and Software Platforms

Whilst many newer and more powerful sensor platforms are available today, much of the existing work for WSNs and WSN security has been developed for the TelosB mote, a 8Mhz TI MSP430 microcontroller with 48KB ROM and 10KB RAM due to its low-power operation and TinyOS support. TinyOS is a lightweight operating system designed for embedded devices in WSNs and has strong support in academia, with many libraries and tools available.

C. iot Protocol

In the design of an improved and fit-for-purpose iot protocol, we attempt to minimise radio usage in order to maximise the lifetime of a battery-powered Thing, sacrificing reliability where appropriate, whilst still ensuring robust and predictable operation. In this section, we first present the architectural model of a typical iot network, followed by the key design areas for achieving these goals within the protocol: reliability, integrity, liveness, minimising packet headers and ensuring correct operational semantics (at-most-once).

iot Model: Within the Intranet of Things ecosystem, three distinct equivalence classes of devices exist:

- **Sensors**, devices which sense and receive input from the real world or another service, such as temperature sensors, presence sensors etc. Data from these devices is typically periodic, thus non-critical, only requiring integrity and not complete reliability.
- **Actuators**, devices which represent physical or digital outputs, such as displays, lights and functions on appliances, that present an RPC interface for a controller to call. Functions provided by these devices are expected to be carried out once and only once when requested, thus reliability and acknowledgements are needed to ensure reliable operation.
- **Controllers**, typically one device such as a PC or home router, which queries, connects and manages the network of sensors and actuators, orchestrating the closed-loop of interaction between sensor inputs and actuator outputs.

Figure 1 presents the iot model with a variety of typical Things connected to the controller, with matching colours representing the closed-loop of control between each of the Things in the network. As discussed below, reliability varies with respect to the typical operations a device performs, which is portrayed with the solid and dashed directed lines.

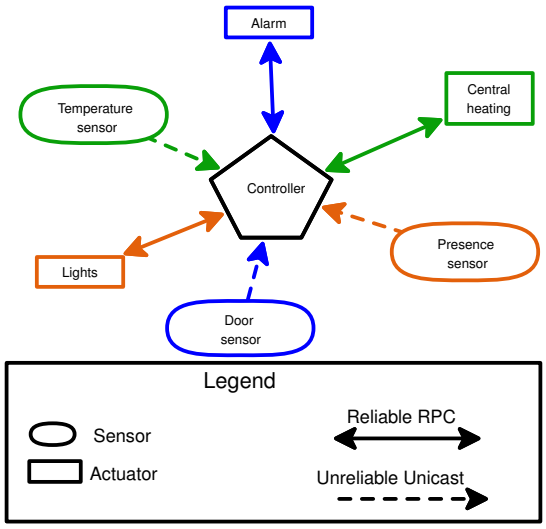

Fig. 1: iot model

Reliability: Typical Things within the iot are low-power embedded devices, relying on wireless communication which is inherently unreliable. Thus, it's necessary to implement reliability on top in order to ensure correct operation. However, whilst other approaches have simply chosen to run over TCP, which provides a fully reliable connection, our design opts for selective reliability on top of the unreliable packet stream. This enables the protocol to ensure critical packets are sent reliably, such as connection handshakes, as well as allow devices to save power and sleep after a transmission instead of waiting for an acknowledgement when the data isn't critical, such as ephemeral sensor readings. Similarly, this also ensures commands sent to actuators from the controller are carried out at-most-once.

Liveness: Ensuring Things within the network are alive and accessible is another key issue, since wireless interference and depleted batteries are a real problem, causing the closed-loop interaction to hang or fail and Things within the network to waste power sending packets to devices that are no longer "out there". To resolve this issue, checks must be performed, using piggybacked messages where possible, to ensure that no extra bandwidth is wasted, such as a sensor periodically requesting a response to a reading. Otherwise standard ping-ack messages are exchanged.

Integrity: Whilst ensuring reliability of some data is not essential, it is however extremely important that any received data be correct. Values corrupted in transit would appear to the controller as seemingly correct, which could then cause the controller to issue invalid commands to an actuator. To ensure integrity of the data, a checksum of the data, provided by the security protocol, is transmitted with it and compared on the receiving side, verifying that it hasn't been altered in transit.

Protocol data unit: One of the key aims of this protocol is to ensure low energy consumption, which on an embedded devices means transmitting as little as possible over the radio, as radio transmission is the most power-hungry operation for a node. Therefore, it was imperative that our protocol adds very little overhead to the already small payload size.

Figure 2 shows the secure protocol data unit layout for the iot protocol. The MAC field, contains the message authentication code (MAC), used for authenticating the integrity of the encrypted fields (shaded grey). Channels are synonymous to ports found in TCP/UDP and enable devices to hold state for more than one connection and potentially take on more than one role. Sequence numbers ensure packet ordering, at-most-one semantics and protect against possible replay attacks. The command field designates the type of payload each packet holds e.g. sensor reading, handshake ack, actuator command etc. The low order bit of the command field, designated R, is used to indicate that the packet requires an acknowledgement.

D. Security

In this section we present the design of our security protocol, enabling new devices to join the network without the need of a shared secret or any physical interaction with the controller. As discussed later in section VII, there are many existing security protocols available for TinyOS, including TinySec [11] and MiniSec [12] for symmetric cryptography

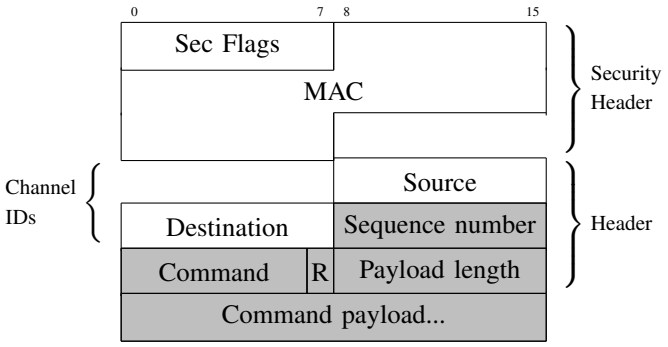

Fig. 2: iot Protocol data unit, shading indicates bytes are encrypted and authenticated using a pairwise symmetric key.

and TinyECC [13] for asymmetric cryptography, as well as a variety of other key distribution schemes [14]–[20]. The security protocol design we propose does not claim to demonstrate a completely new protocol, but is instead a novel combination of the previously described work, MiniSec and TinyECC, based on a conventional security algorithm, Transport Layer Security (TLS), commonly used in desktop machines today.

The key problem with previous approaches to WSN security is key distribution, in which approaches either assume that a network-specific shared secret, key or ID is pre-installed onto some or all of the nodes and thus distribution is not an issue, or the structure of the network is planned and predetermined, with designated high-power nodes to aid in distribution, see section VII. However, this is impractical for networks containing many tens of nodes, entering in a piecemeal fashion and for administration by a novice user expecting device installation to be simple. Thus, a secure key distribution method must be devised to ensure new nodes can join the network without the need to be reprogrammed.

To solve this problem we propose the security protocol design described below and in figure 3. Our design attempts to alleviate complex physical interactions between devices performed by the user [18] and remove the need to design and configure the network, which for the average home user would be non-trivial. Later sections, IV and V, describe the choices made in designing this protocol.

Prior to deployment, nodes will need to be bootstrapped with unique public/private key pairs,a public key certificate and the CA's public key. In a commercial scenario this would require cooperation with trusted manufacturers to create these keys and certificates. After this the protocol is as follows:

1) Initiate discovery mode on the node; node now accepts and verifies incoming certificate requests.
2) Initiate query on controller; controller sends a broadcast query with its public key certificate.
3) The sensor receives the request with the certificate, verifies its authenticity using the CA public key and, if successful, it sends a response with its own certificate.
4) The controller receives the response with the sensor's certificate, verifies its authenticity using the CA public key and replies with an acknowledgement.
5) The sensor receives the response ack, generates a random nonce value (to protect against replay attacks), then encrypts it with the controller's public key and signs it with its own private key before sending it to the controller.
6) The controller receives the signed and encrypted nonce. It first verifies the signature; if invalid it can either request the nonce again or abort the key exchange and waste no further resources. Otherwise it then decrypts the nonce value. The controller then generates a new symmetric key, encrypts it and the nonce with the sensor's public key and signs with its own private key.
7) The sensor receives the signed and encrypted key and nonce. It first verifies the signature; if invalid it can either request the key response again or abort the key exchange. Otherwise it then decrypts the key and nonce value, verifying that the nonce is equal to the one it sent earlier, confirming the key response has not been replayed. Using the symmetric key, the sensor then initialises the symmetric encryption code and then sends a handover message, encrypted using the symmetric key, to the controller to signal it has received the key and is ready to communicate using it.
8) The controller then receives the symmetric handover message, decrypting it to confirm its authenticity, and replies with an acknowledgement, also encrypted using the symmetric key. The two devices can now communicate securely using the symmetric key.

IV. Asymmetric Security

A. TinyECC

To enable secure key distribution, asymmetric cryptography provided by elliptic curve cryptography (ECC) using TinyECC [13] is used. Whilst TinyECC demonstrates the feasibility of using software-based public key cryptography via ECC on low-power microcontrollers, it is still extremely expensive in respect to code size (>15KB), RAM utilisation (3KB) and operational speed, taking up to 5 seconds per operation. Therefore, it's necessary to use it as little as possible and switch over to the far cheaper symmetric key cryptography to keep the connection secure for the rest of its lifetime, in which encrypt and decrypt operations take <2ms.

B. Denial of service protection

Because of the significant cost in verifying a certificate, it would be possible for an attacker to perform a denial of service attack on a node by sending it fake certificates. To reduce the severity of this attack, Things in the network will only accept and process asymmetric packets during a set window of time after the user presses a button on the device. This also provides demonstrative identification for the user, enabling the user to physically control and understand which devices are communicating, similar to WIFI protected setup (WPS) built into commodity home routers.

C. Man-in-the-middle protection

However, using TinyECC alone will not ensure authentication and won't protect against man-in-the-middle attacks. To

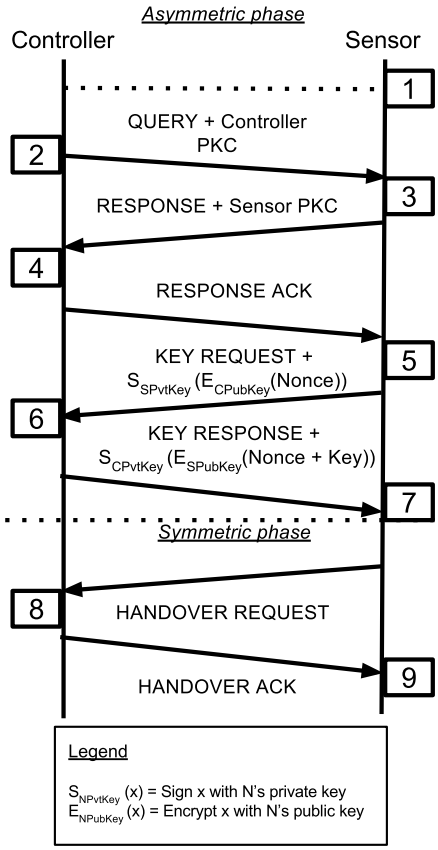

Fig. 3: Security protocol sequence diagram

perform authentication we propose the use of an offline trusted certificate authority (CA) to create public key certificates for nodes. These certificates are then installed on the nodes at compile time along with the node's private/public key pair and the CA's public key. This enables new nodes to have their public key certificate authenticated by other nodes using the CA's public key, before the node's respective public keys are used to exchange the key. This allows nodes to join any network without a previously bootstrapped shared secret symmetric key, unlike other pre-deployment approaches which would require re-bootstrapping the node with the new shared secret symmetric key each time the node joined a new network.

After the nodes have verified each other's authenticity, they are then able to exchange the symmetric key in secret using asymmetric encryption and decryption, followed by handing the connection over to use just symmetric cryptography.

D. Optimisations

To enable ECC to operate within reasonable time bounds on microcontroller platforms, such as the TelosB mote, various optimisations were made in TinyECC, which uses precomputed tables and values created at initialisation to speed up the cryptographic operations. When using TinyECC to perform encryption and authentication, the state computed by these operations overlaps, requiring re-initialisation between operations. To reduce this overhead, we implemented state saving and switching, requiring initialisation to be performed only once for each encryption and authentication module, saving valuable seconds over the entire key exchange. The benefits are further detailed in the evaluation in section VI.

E. Asymmetric costs

Whilst the overarching aim of our iot protocol is to minimise transmission and save as much power as possible, due to the use of asymmetric cryptography for public keys and certificates in which keys and certificates are 40bytes and 20bytes, respectively, large packet sizes are unavoidable. Similarly, whilst TinyECC makes public key computations feasible on embedded devices, combining them in our proposed protocol to provide both authentication and encryption over an insecure channel for the key exchange results in a one-time high cost, taking less than a minute to complete. However, because it's only performed once, the high initial cost is amortised over the lifetime of the node.

V. Symmetric Security

A. Block cipher algorithm and mode of operation

After the handover has occurred, symmetric cryptography is used to keep the connection secure for the duration of its lifetime. The symmetric cryptography used is based on the MiniSec implementation ported from Rogoway et al's Offset Code Book mode of operation (OCB) [21] and Law et al's Skipjack block cipher algorithm [22] implementations. However, due to the inadequacy of various parts of the extended implementation by MiniSec [23], including being written for TinyOS 1.0, allowing only one encrypted connection per node, various memory allocation bugs and inefficient initialisation vector (IV) synchronisation, much of the code was rewritten to work correctly and more efficiently for TinyOS 2.1.

B. Man-in-the-middle and Masquerading protection

Similar to the asymmetric component of the protocol, to prevent man-in-the-middle and masquerading attacks, authentication is necessary, ensuring that a message sent from a Thing in the network is genuine and hasn't been tampered with in transit. To do this a message authentication code (MAC) block needs to be generated from the encrypted data. Unlike cipher block chaining (CBC) and CBC message authentication code (CBC-MAC), which encrypt/decrypt and authenticate the data independently using two unique keys, we use the offset code book (OCB) mode of operation for block ciphering, enabling both encryption/decryption and authentication to be completed in one pass over the data using just one key, realising a slight decrease in time [17] and significant saving in space used for storing the key.

C. Semantic security

In order to ensure semantic security, in which duplicate packets don't encrypt to the same ciphertext, a 64-bit monotonically increasing IV is used for each connection. This would normally require the 64-bits IV to be transmitted to the receiving node, which is extremely costly when attached to each packet. However, using the last bits (LB) optimisation coined in MiniSec, we are able to just transmit the last n

bits of the counter and with some resynchronisation logic on both ends, withstand packet loss of up to 2^n packets before resynchronisation occurs. In our protocol we chose to send the last 6 bits of the counter, due to sharing the byte with another field. In the case of the IV wrapping around, to remain semantically secure the controller only needs to generate a new symmetric key and distribute it before the wrap occurs.

When MiniSec was developed, Skipjack was chosen as the underlying block-cipher due to its efficient implementation and low memory footprint [22]. However, Skipjack uses a maximum key length of 80-bits, which as of 2010 NIST classifies as insecure for long term use [24]. As a temporary measure, the controller can choose to re-issue a new session key for each node after an arbitrary period of time.

VI. Evaluation

To evaluate our proposed protocol we observed several key attributes related to our initial requirements of an efficient, lightweight protocol that attempts to minimise radio transmission. The rest of this section will discuss the iot implementation size for the various roles, followed by the packet overheads compared to other existing protocols, and lastly, the key exchange timings with the related optimisations to ECC.

Things, specifically sensors and actuators, are expected to run on low-power constrained devices, thus it's paramount to ensure an iot protocol not only fits within the constraints of a typical Thing, but that it also leaves adequate space for applications to be built on top of it. Table I shows the implementation size, ROM and static RAM, across the three roles. In the current implementation the actuator demonstrates the lowest overhead with only the iot protocol and LED library linked into the TinyOS compile, leaving approximately 13KB ROM and 4KB RAM for applications. Given that actuators and sensors are only expected to compute very little, this should provide significant space for applications, except, perhaps, in the case when local filtering is necessary; this is demonstrated by the sensor role which shows a significant increase in the ROM size due to the inclusion of the temperature libraries for TinyOS; however, it is still well within the constraints of the environment. Lastly, the controller role demonstrates the largest size, due to the extra processing and state involved in managing the network. Unlike the sensor and actuator, the controller is not definitively bound by the same constraints, and is expected to run on a more powerful device. Whilst old, the TelosB mote still represents a middle ground for embedded devices in terms of ROM and RAM, compared to the popular Arduino platform sporting only 32KB ROM and 1-4KB RAM. However, recent MSP430 and ARM microprocessors have significantly increased amounts of storage available, up to 512KB ROM and 64KB RAM.[1]

The most important requirement of our protocol is to minimise radio transmission; in order to ensure this we not only try to decrease the number of packets sent but also the size of the packets. In table 2 we compare our packet header overheads against the popular and one of the most significant IoT protocols, 6LowPan combined with the security extension, AES-CCM. For traditional 802.15.4 radio, the maximum advised packet size is 128-bytes. 6LowPan, a minified IPv6 protocol, compresses its normally 128-bit addresses to 64- or 16-bit addresses in order to reduce its overhead. Within a smaller number of bytes compared to an un-encrypted 6LowPan packet with 16-bit addresses, our proposed protocol is able to also ensure that a packet is both secure and authenticated, adding only 4 bytes of overhead to an un-encrypted packet for the MAC. In the current implementation there is still room for possible improvement, as demonstrated by MiniSec which discussed removing the group and crc fields from the TinyOS packet, reducing the overhead by a further 3 bytes. RPC calls made using symmetric encryption achieve an average round-trip time of 65ms for a 25 byte packet, including a 1 byte payload (sensor reading), demonstrating efficient and minimal use of resources whilst maintaining robust security.

In order to create a user-friendly, dynamic and flexible key distribution scheme, we opted to use public-key cryptography with TinyECC. In section IV, we argued that whilst TinyECC still exhibits a considerable cost in terms of storage, speed and overhead, the full key exchange only has to be performed once and is amortised over the lifetime of the node; subsequent key exchanges could use the existing symmetric-secured channel. Table III shows the average time for switching between modes and the overall time for the entire key exchange outlined in section IV. TinyECC uses precomputed tables and values based on the public-key to be used for an operation, thus when using different public-keys multiple times to perform a sequence of encrypted and authenticated actions, such as in the proposed iot protocol, these tables need to be recomputed each time in order for them to be effective; this adds a significant overhead of around 20 seconds between the communicating nodes. To reduce this, we added an optimisation to save previously initialised state, thus removing the need to recompute the state for a given public-key the next time it is used; this gives us a fully authenticated and secure key exchange taking just under a minute to complete, split between the two nodes. Because the ECC computations are split between the two nodes, porting the controller role to a more powerful platform would significantly reduce the exchange time, further minimising the time the sensor/actuator node needs to power and listen on the radio.

	Secure iot Protocol		
	Controller	**Sensor**	**Actuator**
ROM size	38,178B (80%)	41,016B (85%)	34,988B (73%)
RAM size	7372B (74%)	6,208B (62%)	6,142B (61%)

TABLE I: Secure iot protocol size for Controller, Sensor and Actuator roles on the TelosB (8Mhz, 48KB ROM, 10KB RAM)mote.

	TinyOS + Secure iot Protocol	**6LowPan + AES-CCM**
Cleartext header	14B + 6B	25B
Encrypted header	14B + 10B	46B

TABLE II: Comparison of payload overhead for our Secure iot (Symmetric security payload) vs 6LowPan with 16-bit addresses + AES-CCM.

VII. Related Work

A. IoT protocols

Given the recent surge of popularity for the Internet of Things, many developers, manufacturers and standards bodies

[1]FreeScale ARM - http://www.freescale.com/webapp/sps/site/taxonomy.jsp?code=KINETIS_L_SERIES

	TinyECC	TinyECC + optimisations
Authentication to Encryption	1.4s	0s
Encryption to Authentication	2.5s	0s
Asymmetric key exchange	80.4s	57.1s

TABLE III: Comparison of the module switch over and overall asymmetric key exchange durations with normal TinyECC and TinyECC with the save state optimisations implemented.

have attempted to create a solution for their view of the IoT, each differing greatly and often being proprietary within their own ecosystems.

The IETF Core working group have proposed a new standard, the Constrained Application Protocol (CoAP) [9], which aims to coerce Things within the IoT to fit into the traditional RESTful client-server architecture that is commonplace on the Internet today, removing the need for specialised platforms or applications to access them. The protocol is built to operate on unreliable UDP links, providing support for reliable delivery on top when needed. In an attempt to be power-friendly, services can subscribe to Things, such as sensors, instead of polling for updates from these devices, allowing them to save power and enter sleep modes. Whilst bringing Things into the client-server architecture eases the adoption of such devices, it forces them to adopt an architecture that simply isn't suited for them considering their low-power constraints, placing restrictions on their availability, reliability and resources. Similarly, it also potentially poses serious security risks if simply plugged-in to the Internet [1].

Building on top of this, the OpenWSN project at UC Berkeley proposes a software stack sitting on top of an 802.15.4 enabled device and below CoAP/HTTP, consisting of 6Lowpan, a compressed IPv6 for embedded devices, combined with RPL, an IPv6 routing protocol for lossy wireless networks and TCP/UDP transport protocols.

As previously discussed, many manufacturers have integrated power-hungry WIFI chipsets and connect the devices directly to the Internet or to their Cloud service [6], [7], [25], without significant consideration for the power and security issues that result. Whilst these devices gain the power and connectivity of the cloud, this couples them directly to it, and increases the chance of failure, due to services going down, limited Internet connectivity and potentially raises issues regarding data ownership and privacy/loss [26]. Additionally, as a result of this direct Internet and Cloud connectivity, devices needlessly waste power and are at a greater risk to security threats [1], [2].

B. WSN Security

There have been several different attempts to efficiently secure WSNs [11]–[20], however none match the iot requirements enumerated in section III-C.

TinySec [11], a symmetric cryptography library for TinyOS 1.0, was an initial effort on TinyOS to address security, intending to demonstrate that software-based cryptography was possible on constrained devices with minimal power overhead. To achieve this, TinySec was designed around WSNs' extreme resource constraints, taking advantage of some of these constraints, such as the limited networking bandwidth, optimising the security primitives in order to reduce the security overhead added to each packet. One such optimisation was the use of a reduced initialisation vector for the cipher block chaining mode of operation, which combined several of the existing header fields and added a 2 byte counter, ensuring the initialisation vectors wouldn't clash between nodes sending the same packet. Whilst the IV is significantly smaller than conventional security protocols, which would reduce time until IV reuse, TinySec argued that this wasn't an issue, demonstrating that using an average send rate of 1 packet per minute, IV reuse would not occur for 45 days.

MiniSec, another symmetric cryptography library for TinyOS 1.0 and a successor to TinySec, was created to further improve the security provided by TinySec, adding replay prevention and also improving upon the minimal security overheads which TinySec achieved. To achieve the increase in security, MiniSec chose to increase the initialisation vector size from 2 bytes to 8 bytes, however, instead of transmitting the whole 8 bytes, MiniSec sends only the 3 last bits (LB) within the pre-existing packet length field, thus incurring no additional overhead for the IV on top of the normal TinyOS packet. To ensure the LB optimisation works correctly, both communicating nodes must maintain state for the counter and perform counter resynchronisation upon significant packet loss ($>2^3$ packets). This counter state is also used to ensure replay prevention, requiring all received packets to have a higher IV than the local state.[2]

Until recently, asymmetric security, namely RSA, was deemed infeasible on microcontroller platforms, taking on the order of tens of seconds to complete public key operations [13]. However, this is no longer the case with the development of elliptic curve cryptography (ECC), in which not only are public key operations feasible within several seconds, but key lengths are reduced whilst still providing the same level of security as longer keys in more traditional asymmetric algorithms such as RSA.

There are many other complex key distribution schemes which require hierarchical networks of devices [16], [20] and predetermined deployment strategies to enable efficient key distribution. Rahmun et al present a solution for such a hierarchical network [20], consisting of a network of heterogeneous nodes, in which high-resource nodes store node IDs for surrounding low-resource nodes and provide authentication for the key exchange process between low-resource nodes using ECC. However, this scheme requires expensive high-resource nodes that need to have the low-resource nodes' IDs stored ahead of time, and also need to be tamper resistant to protect against node capture, further increasing the cost. Another viable alternative, pairings-based key distribution schemes, such as TinyPBC [15], provide efficient key pairings (<5s) but require nodes to know each other's ID *a priori*, which can prove difficult to do with authentication.

Other proposed key distribution schemes such as TinyPK aim to sacrifice immediate authenticity and security on the mote by only performing the public key operations on the more

[2]Whilst the MiniSec paper presented an efficient symmetric cryptography protocol design, the corresponding implementation provided at [23] does not appear to function correctly (with various runtime issues) or implement the IV counter using the LB optimisation described.

powerful server side [14]. In contrast, Message-in-a-bottle [18] relies entirely on a portable faraday cage like barrier to allow devices to secretly communicate in the clear. Whilst it achieved the goal of distributing keys, it sacrifices elegance, usability and scalability, by having to manually place devices inside a lead pipe each time a key needs to be issued.

VIII. Conclusion

Our proposed model and protocol demonstrates a thoughtful approach to designing a secure and efficient Intranet of Things. Compared to other approaches which have used power-hungry WIFI or heavyweight protocols, our protocol attempts to minimise radio usage on 802.15.4 platforms, in an effort to prolong the lifetime of battery-powered Things. In light of existing and recent M2M/IoT attacks, our protocol has been secured against a variety of attacks, ensuring the user's data, privacy and, most importantly, home is safe from remote control, monitoring and other malicious activities. Lastly, our protocol also demonstrates a usable, secure and scalable method for enabling users to add new devices to the network, without the need for pre-shared secret keys or complex configuration, via the use of public key cryptography, provided by TinyECC. The implementation was developed for the TelosB mote running TinyOS, and the source code has been made available online[3].

IX. Future Work

Whilst we have presented a solid foundation for the iot, in the form of a secure and efficient protocol, there are many possible extensions/improvements. This includes porting the controller role to more powerful hardware, such as a PC or router, porting the sensor and actuator roles to other low-power platforms, and implementing ad-hoc routing. These extensions would enable the network to easily scale up to handle hundreds of devices over large and interference-prone areas. It would also allow for rich and configurable interactions between Things by connecting the controller to a suitable platform, such as the Homework Cache [27], a high-performance complex event processing engine designed for processing data events in a closed-loop of control. Lastly, the security against brute-force attacks needs to be improved by replacing the now outdated Skipjack block cipher algorithm with the more secure AES algorithm [24].

References

[1] Belkin IoT WeMo vulnerabilities press release. http://www.ioactive.com/news-events/IOActive_advisory_BelkinWemo_2014.html#Note_1. Accessed: 21/03/2014.

[2] Linux IoT attack. http://www.symantec.com/connect/blogs/linux-worm-targeting-hidden-devices. Accessed: 12/12/2013.

[3] IBM MQTT specification. http://www.ibm.com/developerworks/webservices/library/ws-mqtt/index.html. Accessed: 21/03/2014.

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

[5] Xively, IoT public cloud. https://xively.com/. Accessed: 21/03/2013.

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

[7] Twine IoT Thing. http://supermechanical.com/. Accessed: 21/03/2013.

[8] xAP protocol. http://www.xapautomation.org/. Accessed: 21/03/2013.

[3]Source code available from Github: https://github.com/fergul/Intranet_of_Things

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

[10] Stuxnet attack Symantec dossier. http://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/w32_stuxnet_dossier.pdf. Accessed: 21/03/2014.

[11] C. Karlof, N. Sastry, and D. Wagner, "Tinysec: A link layer security architecture for wireless sensor networks," in *Proc. of the 2Nd International Conference on Embedded Networked Sensor Systems*, ser. SenSys '04. New York, NY, USA: ACM, 2004, pp. 162–175.

[12] M. Luk, G. Mezzour, A. Perrig, and V. Gligor, "Minisec: a secure sensor network communication architecture," in *Information Processing in Sensor Networks, 2007. IPSN 2007. 6th International Symposium on*. IEEE, 2007, pp. 479–488.

[13] A. Liu and P. 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*, 2008, pp. 245–256.

[14] R. Watro, D. Kong, S.-f. Cuti, C. Gardiner, C. Lynn, and P. Kruus, "Tinypk: Securing sensor networks with public key technology," in *Proc. of the 2Nd ACM Workshop on Security of Ad Hoc and Sensor Networks*, ser. SASN '04. New York, NY, USA: ACM, 2004.

[15] L. B. Oliveira, D. F. Aranha, C. P. Gouvła, M. Scott, D. F. Cmara, J. Lpez, and R. Dahab, "Tinypbc: Pairings for authenticated identity-based non-interactive key distribution in sensor networks," *Computer Communications*, vol. 34, no. 3, pp. 485 – 493, 2011, special Issue of Computer Communications on Information and Future Communication Security.

[16] Q. Shi, N. Zhang, M. Merabti, and K. Kifayat, "Resource-efficient authentic key establishment in heterogeneous wireless sensor networks," *Journal of Parallel and Distributed Computing*, vol. 73, no. 2, 2013.

[17] L. Casado and P. Tsigas, "Contikisec: A secure network layer for wireless sensor networks under the contiki operating system," in *Identity and Privacy in the Internet Age*, ser. Lecture Notes in Computer Science, A. Jsang, T. Maseng, and S. Knapskog, Eds. Springer Berlin Heidelberg, 2009, vol. 5838, pp. 133–147.

[18] C. Kuo, M. Luk, R. Negi, and A. Perrig, "Message-in-a-bottle: User-friendly and secure key deployment for sensor nodes," in *Proc. of the 5th International Conference on Embedded Networked Sensor Systems*, ser. SenSys '07. New York, NY, USA: ACM, 2007, pp. 233–246.

[19] P. Porambage, P. Kumar, C. Schmitt, A. Gurtov, and M. Ylianttila, "Certificate-based pairwise key establishment protocol for wireless sensor networks."

[20] S. M. Rahman and K. El-Khatib, "Private key agreement and secure communication for heterogeneous sensor networks," *Journal of Parallel and Distributed Computing*, vol. 70, no. 8, pp. 858 – 870, 2010.

[21] P. Rogaway, M. Bellare, and J. Black, "Ocb: A block-cipher mode of operation for efficient authenticated encryption," *ACM Trans. Inf. Syst. Secur.*, vol. 6, no. 3, pp. 365–403, Aug. 2003.

[22] Y. W. Law, J. Doumen, and P. Hartel, "Survey and benchmark of block ciphers for wireless sensor networks," *ACM Trans. Sen. Netw.*, vol. 2, no. 1, pp. 65–93, Feb. 2006.

[23] MiniSec Homepage. https://sparrow.ece.cmu.edu/group/minisec.html. Accessed: 21/03/2014.

[24] NIST: Recommendations for Transitioning the Use of Cryptographic Algorithms and Key Lengths. http://csrc.nist.gov/publications/nistpubs/800-131A/sp800-131A.pdf. Accessed: 21/03/2013.

[25] Belkin WeMo IoT devices. www.belkin.com/uk/Products/home-automation/c/wemo-home-automation/. Accessed: 21/03/2014.

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

[27] J. Sventek and A. Koliousis, "Unification of publish/subscribe systems and stream databases: The impact on complex event processing," in *Proc. of the 13th International Middleware Conference*, ser. Middleware '12. New York, NY, USA: Springer-Verlag New York, Inc., 2012.