Prototyping a Secure, Hierarchical Internet of Things System

C. Iddon E. Tattershall J. Jensen

August 2017

1 Introduction

As computers and microcontrollers become smaller and more affordable, they have begun to find their way into everyday objects, forming a distributed network of smart devices called the Internet of Things (IoT). The owners of these smart devices are able to send and receive data remotely. There are potential applications for this technology in the areas of home automation [7], citizen science [1][4] and smart cities [2].

However, recent rapid development has led to security concerns as manufacturers and researchers race to be the first to implement their products. Securing these systems is made more difficult by the devices' limitations in terms of memory, processor speed and connectivity; it is sometimes simply not possible to store a long cryptographic key, apply a time-expensive algorithm for encryption, or transmit a long key over the network. Lapses in security implementation have allowed the spread of IoT specific malware such as the Mirai botnet, which takes advantage of factory default username-password pairs used to secure home routers and IP cameras [5].

As we build larger, more complex and increasingly distributed systems, the question of architecture and security becomes ever more important. The EU H2020 project, mF2C, has been set up to explore the issues of architecture, security and implementation in large, distributed IoT systems by considering a number of use cases. These use cases include emergency situation management in smart cities, enriched navigation for vessels at sea and smart hub services for public environments, such as airports, hospitals and shopping centres [3]. This last case, in the context of a Smart Airport in which consumers are able to check up-to-date flight information, find their friends and receive special offers is of particular interest to us and we have used it to inform our thinking in the areas of architecture and security during this project.

In this report, we present a hierarchical architecture in which more powerful devices can act as hubs to ensure the security and privacy needs of smaller devices. We have developed a prototype architecture using off-the-shelf components for use as a test bed for building a comprehensive IoT security protocol.

2 Theory and Specification

2.1 Architecture

Our architecture must be multilayered so that small, limited devices may be supported by more powerful hubs. Most of our potential use cases require that information be aggregated at some central cloud-based server. Therefore, we have chosen to implement a three layer hierarchical network of devices [see Figure 1]. At the bottom of the hierarchy are Edge Devices; microcontrollers with low memory, power and connectivity which are not capable of storing and/or transmitting large cryptographic keys or executing secure cryptographic algorithms. Moving up the pyramid, a group of local Edge Devices may be managed by a single Smart Agent, a more powerful device with an operating system and full TCP/IP network connectivity. At the top of the hierarchy are Cloud Agents; devices and applications that operate in the cloud. These devices link Smart Agents together, provide telemetry, monitoring and data aggregation and can take coordinated action in the event of a suspected attack on the network.

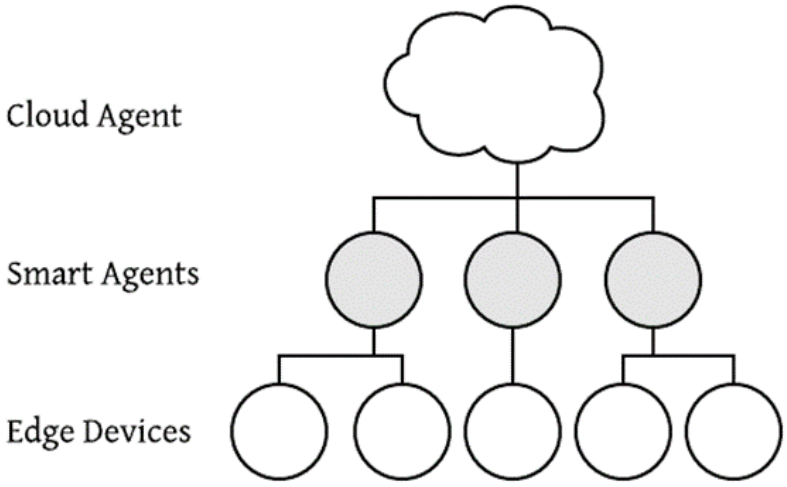

Figure 1: Architecture diagram: this architecture represents a simplified version of the full mF2C architecture that is being developed with our partners.

2.2 Communication

Communication between Smart Agents and Edge Devices is achieved via JSON packets sent over serial while between Smart Agents and the Cloud Agent is via wired and/or wireless TCP/IP network connections.

We have chosen to use a protocol called MQTT to facilitate communication at this upper layer. MQTT, which stands for Message Queue Telemetry Transport, is a lightweight messaging protocol designed specifically for IoT. It relies on a publish-subscribe messaging pattern in which devices may subscribe to a topic to receive any messages subsequently published on it. A MQTT system requires that one device (in this case a machine in the cloud) acts as a broker. The broker is responsible for storing device subscriptions, handling connection status and distributing new messages based on topic.

2.3 Privacy and security

There are three security levels defined by mF2C [3]:

- Public: No special protection requirements
- Protected: Identity of sender must be proven, but the payload does not need to be encrypted
- Private: Identity of sender must be proven, and the payload must be encrypted using strong asymmetric keys.

Security level	Sender signature	Payload encryption	Example uses
Public	No	No	Directory of airport shops
Protected	Yes	No	Flight times
Private	Yes	Yes	Locations of user's friends

Table 1: Security levels specification

Each Agent generates a public/private key pair when it is initialised, or loads one from a file. Rather than have each agent store the public keys of every other device on the network, a Cloud Agent that we call

Broker Services facilitates secure message delivery. Each Smart Agent knows only its own public and private keys and the public key of the Cloud Agent. The Cloud Agent maintains a list of all the Smart Agents that have previously connected, their current connection status and their public keys. When Alice wishes to send a message to Bob via her device, her Smart Agent encrypts the message with the Cloud Agent's public key and sends it up the hierarchy. The Cloud Agent then decrypts the message and re-encypts it with Bob's public key before sending it back down the hierarchy to his device.

2.4 MQTT Topic Structure

Using the MQTT protocol for secure point-to-point messaging requires careful thinking on the chosen structure of topics. Topics are hierarchical and subscription messages allow wildcards, so that if an agent subscribes to *mf2c/#* (where # is the wildcard symbol), it receives messages sent to *mf2c*, *mf2c/agent1*, *mf2c/agent2* and *mf2c/agent1/edge1*.

We assign each agent a number of inboxes. When an agent would like to send a message to another agent, it sends it to that agent's relevant inbox. It also regularly checks its own inboxes for new messages. Some of these inboxes are used in ping to verify connection status and speed.

2.4.1 Standard Inboxes

All smart agents subscribe to these topics, replacing *agent1* with their ID:

- *mf2c/agent1/public*
- *mf2c/agent1/public/pingreq*
- *mf2c/agent1/public/pingack*
- *mf2c/agent1/public/handshake*
- *mf2c/agent1/protected*
- *mf2c/agent1/private*

2.4.2 Inboxes for Broker Services

The Broker Services Cloud Agent has its own public/private key pair and is used to facilitate sending protected and private messages. It has all of the inboxes shown above (with *broker-services* substituted for *agent1*) but additionally it subscribes to:

- *mf2c/broker-services/status*
- *mf2c/broker-services/discovery*

2.4.3 Topics for Discovery

In the topics above, the handshake topic is used for sharing public keys, while the *broker-services/status* message is used by agents reporting a change in their connection status, where connection status is one of:

- Connected
- Disconnected gracefully (for intentional disconnections; e.g. user logged out)
- Disconnected ungracefully (for unintentional disconnections; e.g. loss of internet service)

The *broker-services/discovery* topic is used by the broker to regularly post a list of the agents that have set their visibility appropriately.

2.5 Sequence of events

2.5.1 Smart Agent Arrival

When a Smart Agent arrives on the network:

- The Smart Agent subscribes to its inbox topics.
- The Smart Agent publishes its new connection status on *mf2c/broker-services/status*.
- The Smart Agent publishes its public key to *mf2c/broker-services/handshake*.
- Broker Services saves the name of the agent and its public key, then publishes its own key on *mf2c/[agent name]/handshake*.

2.5.2 Normal Communication

While the Smart Agent is on the network:

- The Smart Agent checks its inbox topics for new messages.
- If any of those messages is a ping request (i.e. it was published on topic *mf2c/[agent name]/public/pingreq*), it sends back a ping acknowledgement on *mf2c/[ping source name]/public/pingack*.
- The Smart Agent returns a list of decrypted message objects to the user.

2.5.3 Private Communication

When the Smart Agent sends a private message to one or more recipients:

- The Smart Agent uses its own private key to sign a hash of its own name and adds this signature to the message.
- The Smart Agent encrypts the message payload using broker-services' public key.
- The Smart Agent publishes the packaged message to *mf2c/broker-services/private*.
- Broker Services decrypts the message payload and verifies the signature. It signs the message and encrypts the payload using its own private key.
- Broker Services publishes the message to each of the recipients private inbox topics (e.g. *mf2c/[destination agent name]/private*.

2.5.4 Smart Agent Departure

When the Smart Agent leaves the network:

- It publishes a disconnect message on *mf2c/broker-services/status*.
- Broker Services records that the device has left the network.

3 Implementation

We implemented a test bed for future development of a security protocol. Our proof-of-concept used standard IoT components to produce our architecture and move data over it, demonstrating common features of a larger IoT system.

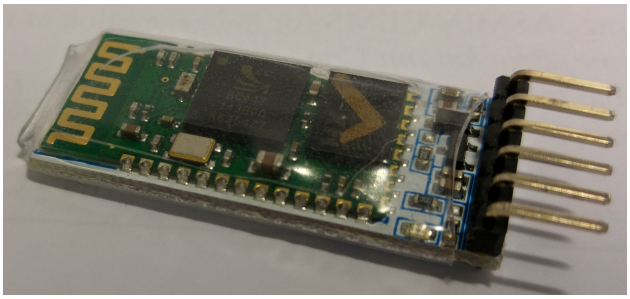

Figure 2: The HC-05 chip for serial over Bluetooth. It costs roughly £4 and has a range of about 10m. Requiring a pin to pair the devices, it can have the role of either slave (wait for connection passively) or master (search for the device and actively initiate connection).

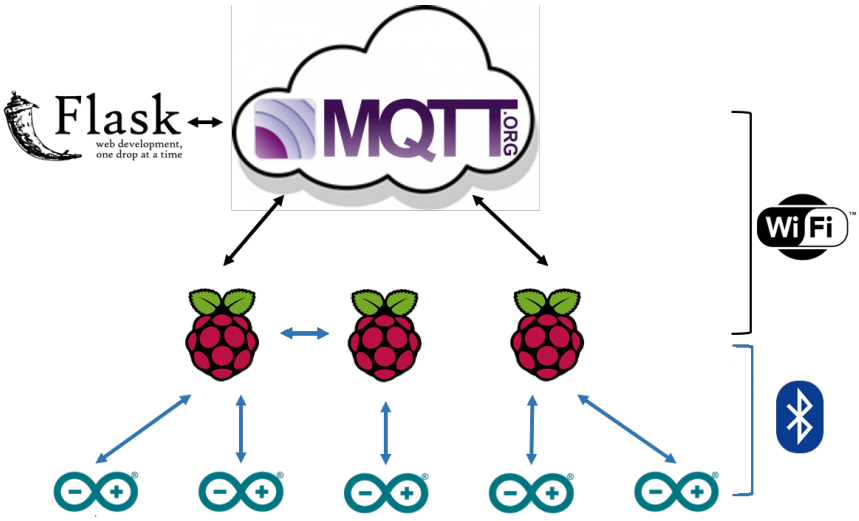

Figure 3: The architecture we implemented.

3.1 Equipment

For Edge Devices, we used Arduino Unos which are inexpensive (£25-30), low powered and portable making them a popular choice for IoT. With 14 inputs and outputs (analog and digital), these microcontrollers can be loaded with pre-compiled programs to read from sensors and write to outputs. To communicate with Smart Agents, we used Serial over either USB or Bluetooth (through the addition of a HC-05 Bluetooth module - see Figure 2).

As Smart Agents, we used the Raspberry Pi 3 which is a single board computer. Communication with Edge Devices and the cloud was possible without any additional adapters over USB, Ethernet, WiFi and Bluetooth. The Pi can run PC scale applications headless on its Linux operating system. Its low price, small form factor and low power requirement make it popular for IoT.

We used STFC's SCD Cloud for access to a virtual machine running Scientific Linux 7. A common LAN enabled TCP/IP communication between this machine and the Smart Agents or the web application.

3.2 Setup

The cloud VM was setup as the MQTT broker using Mosquitto. There were three endpoints contacting it: Smart Agents, Broker Services and the web application. Using the Paho MQTT client library for Python

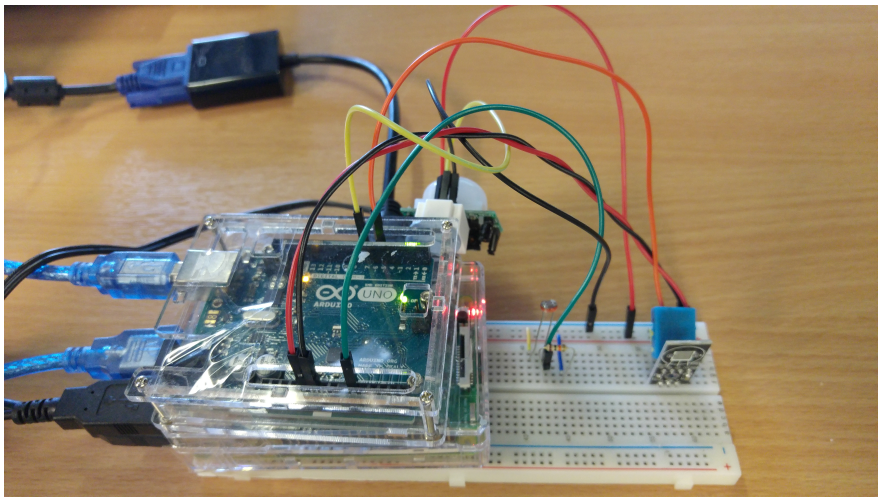

Figure 4: A Raspberry Pi (bottom) connected to an Arduino (top) relaying data to the MQTT broker in the cloud passed from the Arduino which is reading the attached sensors.

as well as the PyBluez module, the Smart Agents acted as a gateway to the cloud for any connected Edge Devices. If one Smart Agent could not directly connect to the hub, Bluetooth tethering was used to share the connection of a nearby Smart Agent. Broker Services ran on the cloud as a Python application (also using Paho) to publish a list of currently connected devices for discovery. A Flask module (Flask-MQTT) served a web application, allowing a user to subscribe to topics for an overview of the data flows in the network.

There were several problems with using Bluetooth. In our initial setup, the HC-05s were configured in slave mode with names according to a pattern. The Smart Agent would periodically scan for new devices according to the pattern and then attempt to connect. However, with only one Bluetooth chip, each Pi could not send/receive while scanning so communications were delayed substantially (roughly 20s). To counter this, we used a fixed list of Edge Devices per Smart Agent which were always available. While seemingly restrictive, this would work adequately for a variety of use cases - such as a central Smart Agent controlling statically placed sensors in a house or vehicle. Additionally, the number of devices in each piconet was restricted to seven. This didn't affect our prototype but may be an issue for scaling in a real setup.

Depending on the use case, different communication technologies may be more applicable. With Edge Devices that are relatively fixed, our serial protocol worked over USB - where there was no delay to scanning but was it was not wireless. Some protocols that are wireless - such as Zigbee - can connect a larger number of devices but require bespoke hardware at both ends. Our technique of sending a JSON encoded packet could work on any text communication protocol - we implemented it over USB and Bluetooth serial.

Edge devices interacted directly with each sensor, sending JSON packets of tagged data for the Smart Agent to forward to the broker. The programs were written in Arudino C++ and were compiled and uploaded using the Arduino IDE.

3.3 Data

Figure 5 shows an example of data that was collected during a test of our system. Using four Smart Agents at different places around the office overnight, we connected Edge Devices with sensors whose data was published to the MQTT broker for recording. Including LDRs, Thermistors and Humidity sensors, the different combination of detectors attached to each device demonstrated the system handling various data streams concurrently.

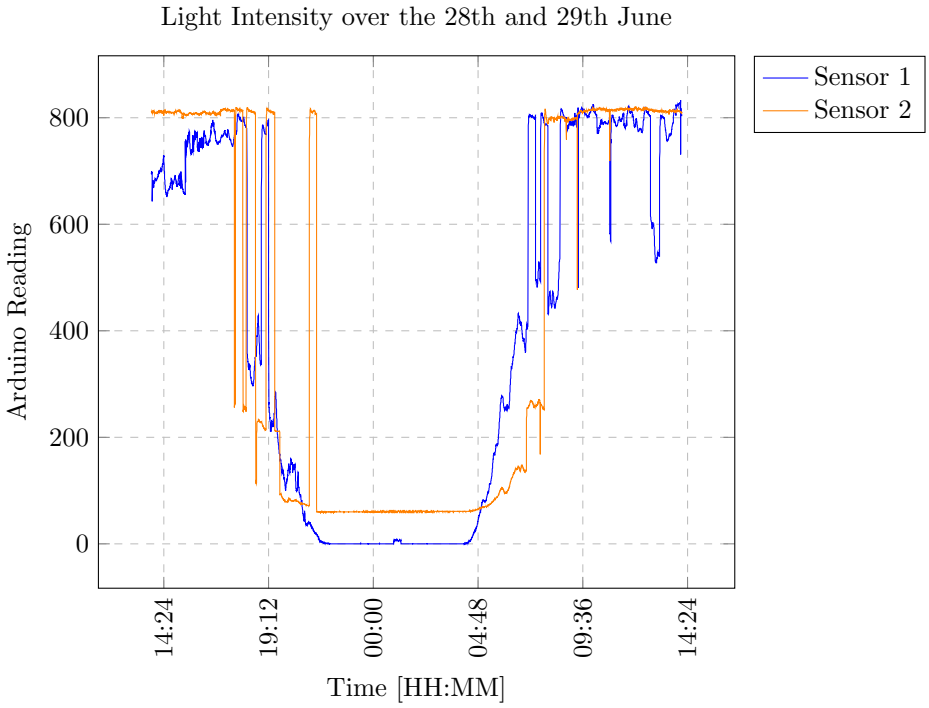

Figure 5: The data collected by different Arduinos, aggregated in the cloud after being published to the MQTT broker via the Smart Agents. The light intensity is recorded as the voltage over an LDR.

The data itself is interesting. Sensor 1 - which was closer to a window - shows a steeper increase in light intensity in the morning compared to Sensor 2 where the most significant rise comes later, from the lights turning on. At different points during the night (particularly around 9:15 near Sensor 2) the lights were turned back on indicating that someone returned to the office or something triggered their movement sensors. We used this setup as a part of STFC outreach events to record similar data for visitors (students, children) to analyse.

4 Dissemination and Future Work

During the course of this project, we built up a substantial codebase which is available on our public GitHib repository [8]. This includes the Python and Arduino C++ code used to interface between the Raspberry Pis and Arduinos and a Python API built on top of the paho-mqtt library that allows researchers to easily implement our architecture and security specification.

Our setup could be used to simulate the attack of a botnet. When testing a security protocol, our implementation would give a chance to test possible attack vectors on an IoT network. As the bulk of our setup uses TCP/IP for communication, it would be possible to virtualise the simulated attackers via Docker containers on machines on the same network - perhaps pretending to be Smart Agents. This would allow simulation of large scale attack (much greater than the number of physical devices we have available). Not only would this give an indication of the resistance of the protocol. It would also give a practical way to evaluate different protocols against each other, highlighting the strengths and weaknesses. Additionally, the response to unexpected events could be investigated.

With Smart Agents as sophisticated gateways, ostracising devices could be implemented in the event of a security breach. The cloud could ignore connections from a list of compromised Smart Agents or they could

ignore specific Edge Devices. Provided that this list could be established in time, this would act as a barrier to prevent the flow of malicious data throughout the network.

One way to establish this list would be to apply machine learning. It might be possible to train a classifier on network activity to distinguish compromised devices. Our use of MQTT - where any application is free to subscribe to all topics - would allow this classifier access to the number of communications from each machine and even the payload of those messages (where encryption isn't used). It is suspected that malicious devices would offer some kind of signal: perhaps a common payload or even just an abnormal number of messages at specific times which would be possible to spot by statistical or even classical techniques (peak detection).

5 Conclusion

We developed a prototype of an Internet of Things architecture for future testing of a security protocol. If applied to our main use case, a Smart Airport, our implementation of a distributed network would allow users to connect, share and receive data; interacting with a central broker through a Smart Agent according to their defined privacy level. We used the three core privacy specifications of mF2C and integrated them into an MQTT based API for use by the project in testing its use cases. STFC could benefit from our prototype if it was adapted for use as a distributed monitoring system for the facilities. Additionally, the inexpensive design - mainly using Raspberry Pis and Arduinos - makes this setup applicable to outreach events where Edge Devices can be distributed amongst visitors/students, uniquely involving them in data collection.

6 Acknowledgements

This research was conducted at STFC and received funding from the mF2C EU Horizon 2020 partners. We are grateful for the support of our project partners at ATOS, UPC, TUBS, Intel, BSC, WoS, XLAB, SIXSQ and Tiscali.

Project Number	730929
Start Date	01/01/2017
Duration	36 months
Topic	ICT-06-2016 Cloud Computing

References

[1] K. Fukuoka and S. Awai and S. Markon *A mobile cloud-based radiation monitoring system for citizens.* 2015 IEEE Region 10 Humanitarian Technology Conference (R10-HTC), 2015.

[2] A. Zanella, N. Bui, A. Castellani, L. Vangelista and M. Zorzi *Internet of Things for Smart Cities.* IEEE Internet of Things Journal vol. 1, no. 1, pp. 22-32, Feb. 2014.

[3] mF2C Website
`http://www.mf2c-project.eu/`

[4] Citizen Flood Detection Network
`https://www.postscapes.com/citizen-flood-detection-network/`

[5] B. Herzberg, D. Bekerman, I. Zeifman *Breaking Down Mirai: An IoT DDoS Botnet Analysis*
`https://www.incapsula.com/blog/malware-analysis-mirai-ddos-botnet.html`

[6] Our Github Repository
https://smartcitizen.me/

[7] Openhab: Home Automation
https://www.openhab.org/

[8] Our Github Repository
https://github.com/etattershall/internet-of-thingies