

Electronics and Computer Science
Faculty of Physical Sciences and Engineering
University of Southampton

Matthew Hutchings
12th of May 2020

Recommending and modelling optimal security practices for smart grid connected IoT devices

Project Supervisor: Dr Nawfal Fadhel
2nd Examiner: Dr Xu Fang

A report submitted for the award of
MCOMP Information Technology in Organisations

Abstract

The primary objective of this project was to firstly identify the main threats that face IoT devices in a smart grid system and then recommend security policies that could mitigate these threats. To test that the outlined policies were fit for purpose they were modelled and verified using Scyther, a tool for the automated verification of cryptographic protocols. The project found that smart grids face a diverse threat landscape and that without specific security measures IoT devices in these systems are particularly vulnerable to attack. However, verification results from Scyther for the policies developed within this project show that well implemented security policies and best practices can successfully mitigate these threats.

Contents

1	Project Description	7
1.1	Project Aims and Objectives	8
2	Literature Review	9
2.1	Internet of Things (IoT) Devices	9
2.2	Smart Grids	9
2.3	IoT Smart Grid, the Threats, Attitudes and Best Practices	9
2.4	Verification of Security Policies and Protocols	10
2.5	Discussion	10
2.6	Conclusion	11
3	Research and Design	12
3.1	Scyther development environment	12
3.1.1	Requirements and development methodology	13
3.1.2	Development plan	15
3.2	Threat Model	16
3.2.1	Weak/Default Password Fuzzing Attack	17
3.2.2	Man In The Middle (MITM) Attack	17
3.2.3	Passive Eavesdropping	18
3.2.4	Replay Attack	18
3.2.5	Impersonation Attack	19
3.2.6	Open Port Scanning	19
3.2.7	Network Traversal	20
3.2.8	Software Vulnerability	20
4	Recommendation of policies and practices	21
4.1	Communication Policies	21
4.2	Description and specification of IoT Best Practices	22
4.2.1	Password Management	22
4.2.2	Network Segregation	22
4.2.3	Patch Management	23
4.2.4	Minimum Design	24
5	Implementation and modelling of communication protocol policies	25
5.1	Message Encryption	25
5.1.1	Design	25
5.1.2	Implementation	26
5.1.3	Review	27
5.2	Implicit key authentication	28
5.2.1	Design	28
5.2.2	Implementation	30
5.2.3	Review	31
5.3	Unique Session Keys	32
5.3.1	Design	32
5.3.2	Implementation	33
5.3.3	Review	34
5.4	Mutual Authentication	35
5.4.1	Design	35
5.4.2	Implementation	36
5.4.3	Review	37
6	Project Management	38
6.1	Coronavirus mitigation	39
6.2	Risk Management	40

6.3	Scyther development environment	41
7	Project Evaluation	43
7.1	Achievement against project goals	43
7.2	Conclusion	44
7.3	Future Plans	44

Nomenclature

Nonce A randomly generated value that is used only a single time in a cryptographic protocol. Often used as a timestamp to prevent the reuse of old message credentials.

Adversary A term used to describe a party attempting to disrupt the security of a system.

Hypervisor A piece of software that creates an instance of a virtual machine from a virtual machine file.

Communication party An intended sender/recipient of a communication.

Malware Malicious software designed to attack the security of a computer/computer network.

Statement of Originality

- I have read and understood the ECS Academic Integrity information and the University's Academic Integrity Guidance for Students.
- I am aware that failure to act in accordance with the Regulations Governing Academic Integrity may lead to the imposition of penalties which, for the most serious cases, may include termination of programme.
- I consent to the University copying and distributing any or all of my work in any form and using third parties (who may be based outside the EU/EEA) to verify whether my work contains plagiarised material, and for quality assurance purposes.

I have acknowledged all sources, and identified any content taken from elsewhere.

I have not used any resources produced by anyone else.

I did all the work myself, or with my allocated group, and have not helped anyone else.

The material in the report is genuine, and I have included all my data/code/designs.

I have not submitted any part of this work for another assessment.

My work did not involve human participants, their cells or data, or animals.

1 Project Description

IoT devices present many exciting applications for both industrial and consumer use. However, increased dependence on these devices opens up new consequences and attack vectors that an adversary can use to attack a target. This is of particular importance in the case of IoT devices connected to smart grid infrastructure as cyberattacks could be used to disrupt critical national infrastructure.

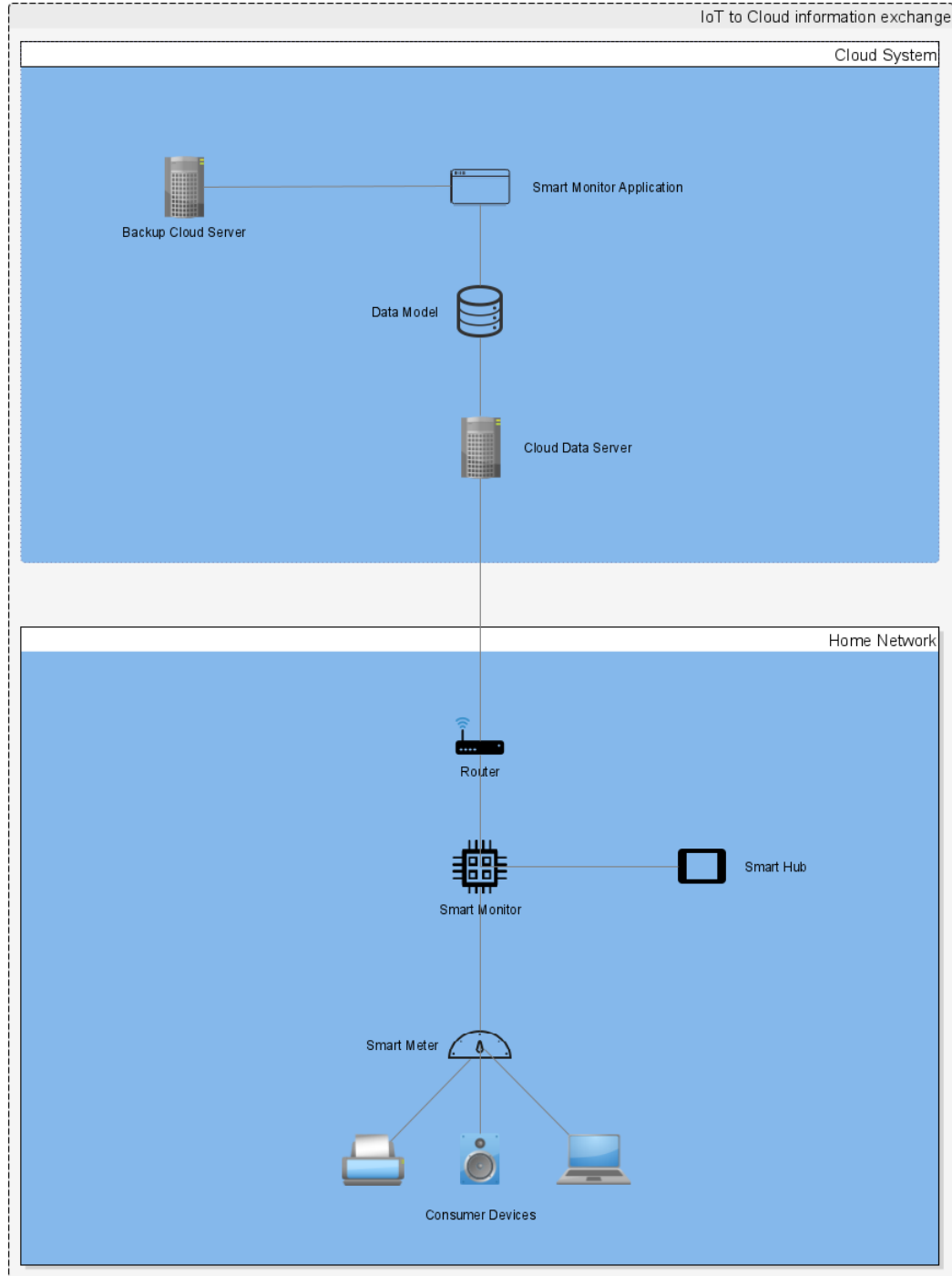


Figure 1: Reference diagram of my smart grid scenario

The scenario for my project is an IoT based smart grid with a focus on the communications between IoT devices and their interactions with the cloud layer. The reference diagram above details the devices present in the system. The devices within the home network are installed in each household and interact with their local cloud server which manages all the households in an area.

1.1 Project Aims and Objectives

This project aims to produce, model and verify a collection of policies and protocols that are suitable for mitigating the threats that a IoT enabled smart grid may face. I wish to focus on the following goals within this project:

- Investigate and conduct a risk assessment on the main vulnerabilities and threats faced by IoT devices within a smart grid environment.
- Recommend security policies that can mitigate these threats, justifying these policies by taking into account secondary factors including the cost to implement and any loss to productivity these policies might incur.
- Implement and verify that these communication protocols mitigate the identified vulnerabilities using Scyther, a formal method based protocol verification tool.
- Clearly explain the impact of each of my policies by comparing the possible attack vectors with and without each policy using Scyther.
- Create a purpose built, portable Scyther virtual machine environment allowing myself and others to quickly set up and start using Scyther on a new device. Therefore allowing others to verify and extend upon the results of the project.

The scope of this project will be investigating, modelling and verifying the best policies and practices for IoT devices and their communications in my smart grid scenario. The project focuses mainly on IoT communication protocols and their configuration rather examine flaws in the hardware or firmware ran by these devices.

2 Literature Review

My literature review explores the IoT and smart grid landscape before looking into the cybersecurity issues that a smart grid implementation may face. Finally, the review discusses the verification of cryptographic protocols in the context of my scenario

2.1 Internet of Things (IoT) Devices

IoT as a general concept can be described as physical objects also being network identifiable devices that are able to communicate without the need for human interaction [1]. These devices can be used in a home or industrial context to automate processes or afford additional functionality. IoT devices can do this as they are able to leverage information by collecting/receiving it across a network. As an example of an IoT implementation in a chemical production plant, IoT monitoring devices could be used to monitor the temperature of a reaction. If the temperature fell outside of the requirement, the device could communicate with another IoT device that controls the coolant flow through the reaction and correct it without the need for any human interaction.

These IoT networks can offer benefits for existing processes such as improved efficiency, fewer employees required to manage it and data which can be used to improve the process. However, it is important to consider from a cybersecurity perspective that the introduction of networked devices to a process opens it up to the possibility of cyberattacks.

2.2 Smart Grids

The term smart grid refers to the integration of technology into electrical grid systems allowing them to dynamically change to meet the current needs of consumers [2]. Whilst the implementation of smart grids can vary significantly, several elements generally remain constant:

- **Smart Meters and Monitors** - These IoT devices are used to measure and analyse the energy usage within a single home. Typically smart meters simply collect energy readings from a room and send this information to the smart monitor. This monitor relays energy information to a collection server and receives information on current energy prices. [3]
- **Smart Hub** - This device allows the homeowner to track their electricity usage as well as view the current electricity price to help time their electricity usage to get the best price resulting in a better distribution of power demand across the power grid.
- **Cloud Layer** - This layer communicates with the Smart Meter to receive electricity usage information and send electricity pricing information. This information can then be used by the rest of the smart grid system to adjust the routing and production of electricity based on current demand.

2.3 IoT Smart Grid, the Threats, Attitudes and Best Practices

A key finding from my research, summarised by Robles [4] is that one of the key differences between securing a traditional system compared with a national infrastructure system, such as smart grid, is the reduction in the effectiveness of standard security measures such as patches, password management and access control. Stating that this is due to the size and diverse combination of hardware and software that comprises this class of system. Whilst traditional controls do have their place in smart grid security Sajid [5] identifies the need for specific security measures that directly mitigate the threats smart grids face. This point is further explored by Bere [6] which states that large industrial control systems are often the target of state-funded Advanced Persistent Threat (APT) groups whose capabilities and resources far outmatch the typical threat actors a system faces. [6] Bere goes on to recommend that the security protocols and controls implemented should be layered, providing a 'defence in depth' security approach which Virvilis [7] states as a key countermeasure against APT groups as these groups have the ability to execute zero-day exploits. Zero-day exploits offer very little chance

of mitigating an attack against part of a system as the vulnerability is only known to the adversary at the time of execution [8]. However, a layered system means that in the event of such an attack, the entire system will not be compromised due to the presence of other security measures and protocols.

Another area of difficulty when it comes to securing these systems is the perspective and attitudes of governments and other organisations when it comes to securing these systems. Wang [9] states that many organisations do not see investing in the protection of these systems as economically viable. Virvilis [7] adds that disruption to productivity and user experience due to the increase in latency or removal of features that hardened security protocols may necessitate is another factor in the lack of implemented protocols on these systems. Mcqueen [10] suggests that it is difficult to quantify cyber risk using traditional risk assessment methods. This may further contribute to the reluctant attitude towards cybersecurity investment as it is difficult to quantify the benefits of a reduction in risk to management.

2.4 Verification of Security Policies and Protocols

Creamers [11] states that it is very difficult for humans to analyse and find flaws in cryptographic protocols, as evidenced by the number of protocols that are found to have security flaws after their release. An example of this is the Needham-Schroeder key distribution protocol which even after extensive analysis and verification by hand was found to have a security flaw which allowed an adversary to pass off an old session key as a new and valid one [12]. Meadows [12] goes on to suggest that formal methods are a good choice for analysing these cryptographic protocols as they are enclosed enough to make modelling and verification feasible whilst also having the potential for subtle and counter-intuitive flaws that an informal analysis may miss.

In order to verify a protocol using automated formal methods, it must first be modelled so that it can be interpreted by a protocol verification tool. In my research, I have found two tools that are the most suitable for this purpose; Pro-Verif and Scyther. In their comparative analysis of these two tools Dalal et al. [13] identifies that whilst the two tools share several similarities, there are key differences as shown below:

- **Modelling Language** - Scyther uses 'security protocol description language' (SPDL) described as "a mix between java and C" by creator Cas Creamers [11] to model protocols. Whereas Pro-Verif protocols are represented using horn clauses or pi calculus [13]. The SPDL used by Scyther is closer to pseudo-code than Pro-Verif making it more suitable for illustrating the implementation of protocols as well as being more fitting to my skillset.
- **Attack Graphs** - Scyther automatically generates attack graphs when a flaw is found in verification, generating a visual flow diagram of the attack. Pro-Verif does not support this feature.

2.5 Discussion

Based on the easier to understand modelling language and attack graph feature identified in Dalal's comparison of Scyther and Pro-verif, the project will use scyther for the modelling and verification of security protocols.

2.6 Conclusion

The key finding from my research into the literature surrounding smart grid security is that a smart grid system faces additional cybersecurity obstacles compared with a traditional industrial network. This is partially due to the nature of the system itself as smart grids comprise of mainly low power devices being distributed across a large area, resulting in additional factors that must be considered when applying common cybersecurity controls. Another reason for these additional obstacles is the cyber threat landscape that smart grids face. As smart grids form part of a nation's critical national infrastructure, they can become a target for state-funded advanced persistent threat groups capable of more advanced attacks than a typical cyber threat actor.

3 Research and Design

When it comes to implementing a best practice cybersecurity strategy, NIST [14] recommends a five step process for analysing and securing smart grid systems. This process is defined below along with how the project will implement each of the points outlined.

1. Defining use cases - *The use cases of the system should be defined.*

Though the use of a reference diagram, the scope and elements comprising the project's IoT system are clearly defined.

2. Risk Assessment - *The vulnerabilities, threats and the impact these threats can cause should be evaluated for the system.*

A threat model based on the reference diagram will be used to illustrate where vulnerabilities in the system are present. The vulnerabilities will be further described in isolation with emphasis placed on the threats posed by this vulnerability and the impact of these threats in the context of the project's scenario.

3. Specification of Security Requirements - *The security requirements for the system should be stated and specified.*

Taking into account the threats outlined in the previous step, A list of policies will be produced outlining the security requirements of the system.

4. Design and Development of a Security Architecture - *A security architecture to protect the smart grid system should be designed and implemented.*

Protocols will be designed and then implemented in Scyther that satisfy the policies defined in the previous step.

5. Assessment of implementation - *The architecture should be assessed against the defined security requirements to test if it is fit for purpose.*

The project will use Scyther's protocol verification tools to test the protocols against the requirements defined.

3.1 Scyther development environment

One of the key goals of the project was to create a portable environment for Scyther development. When researching what the most suitable tools to create this would be, particular emphasis was placed on the following criteria:

- **Self-contained** - Once installed, the virtual machine should contain all the components and dependencies required to develop using Scyther
- **Portable** - The machine must be simple to install and ideally small enough that it can be hosted using common software distribution tools such as GitHub.
- **Configurable** - Users should be able to make changes to the environment to suit their individual preferences and needs. This includes elements such as the flavour of Linux used and the resources assigned to the machine.
- **Versionable** - Changes made to the machine configuration should be versionable using git, the most common version control method. This allows for future adaptations and iterations upon the machine to be easily tracked as well as making it easy for users to revert changes made to the machine if issues occur.
- **Compatible** - The machine should be formatted so it does not require the use of proprietary software to run.

During research, two virtual machine formats stood out as being suitable for the project; the Open Virtualisation Format (OVF) and Vagrant. Both of these formats are non-proprietary and easily

compatible with common hypervisors like VirtualBox. Either format would also allow for the packing of all the required software to develop using Vagrant.

One of the key advantages Vagrant possess over OVF is the vagrant file. Vagrant environments are packaged in formats known as boxes, vagrant files allow a user to customise and specialise a box to suit their needs. As Vagrant files are written in the Ruby programming language, they are versionable and compatible with git, this makes the process of collaborating with others and extending vagrant files simple. However, Vagrant virtual boxes are slightly more complex to install as they require the vagrant software to be installed on the host machine, OVF files can just be imported into the chosen hypervisor. Despite this, the configurability and git compatibility that vagrant files provide offer great value in this use case making them the best choice for the project's Scyther environment.

3.1.1 Requirements and development methodology

Based on the criteria outlined in the section above, a list of requirements for the Scytherbox have been developed. To prioritize these development requirements, the MoSCoW method has been used:

Table 1: Table showing the prioritisation of the scytherbox requirements

Must Have	Should Have
Scyther v1.13 installation from VagrantFile	Linux flavour options
Scyther dependencies installed from VagrantFile	Shared folder between host and Scytherbox
Versionable using git	Installation Instructions
	Example Protocols within the box
	Scytherbox documentation
Could Have	Won't Have Now
Git configuration within the box	Windows version of scytherbox
Scyther user guide	

Despite a lack of clearly defined stakeholders for this part of the project, elements of the AGILE methodology will be used to plan and manage the development of the Scytherbox. The main reason for this is that a basic version of the box that can run Scyther is required to complete the modelling and verification of the suggested protocols making a initial working release a high priority.

To estimate the development time each requirement will take to implement, t-shirt sizing has been used. The table below shows a description of each requirement and it's estimated size.

Table 2: Table describing and estimating the size of each Scytherbox requirement

Requirement Title	Description	Estimated Size
Scyther v1.13 installation	The latest version of Scyther should be imported into the box using the VagrantFile	Medium
Scyther dependencies installed	Python 2.7, as well as the GraphViz and wx-Python libraries, should be installed on the box using the VagrantFile	Medium
Versionable using git	Changes to the box configuration should be visible in git so they can be versioned.	Small
Linux flavour options	Allow changing of Linux flavour during box configuration	Small
Git configuration	Implement functionality allowing git repositories to be cloned and configured on the box during configuration	Large
Shared folder	Implement a synced folder on the host machine and Scytherbox allowing for the easy transfer of files between the two	Small
Installation instructions	A set of instructions explaining how to install Scytherbox	Medium
Example protocols	A folder on the host machine attached to the box allowing sample protocols to be included in an installation of the box.	Medium
Scytherbox documentation	A documentation of the VagrantFile and shell scripts used to create the Scytherbox, making it easier for others to iterate on in the future	Large
Scyther user guide	A guide installed on the box explaining the basics of Scyther and the scyther protocol description language (SDPL)	Medium
Windows version	A version of the Scytherbox that uses Windows as the box's operating system	Extra Large

Key:

Small - 1 Hour

Medium - 2 Hours

Large - 4 Hours

Extra Large - 10 Hours

3.1.2 Development plan

The development of the Scytherbox environment is scheduled to take place over a 3 week period with 1 week allotted for each sprint. The modelling and verification of the project's policies is scheduled to take place 1 week into this period hence the core features required to develop using Scyther being assigned to the first sprint.

Table 3: Scytherbox sprint plan

Sprint 1	Sprint 2	Sprint 3
Scyther Installation	Git configuration within the box	Scyther user guide
Scyther dependencies installed	Linux flavour options	Example Protocols within the box
Versionable using git	Scytherbox documentation	Installation Instructions
Shared folder		
Sprint Hours: 6	Sprint Hours: 9	Sprint Hours: 6

3.2 Threat Model

The threat model below shows the key attack vectors and vulnerabilities an adversary could exploit within the scenario:

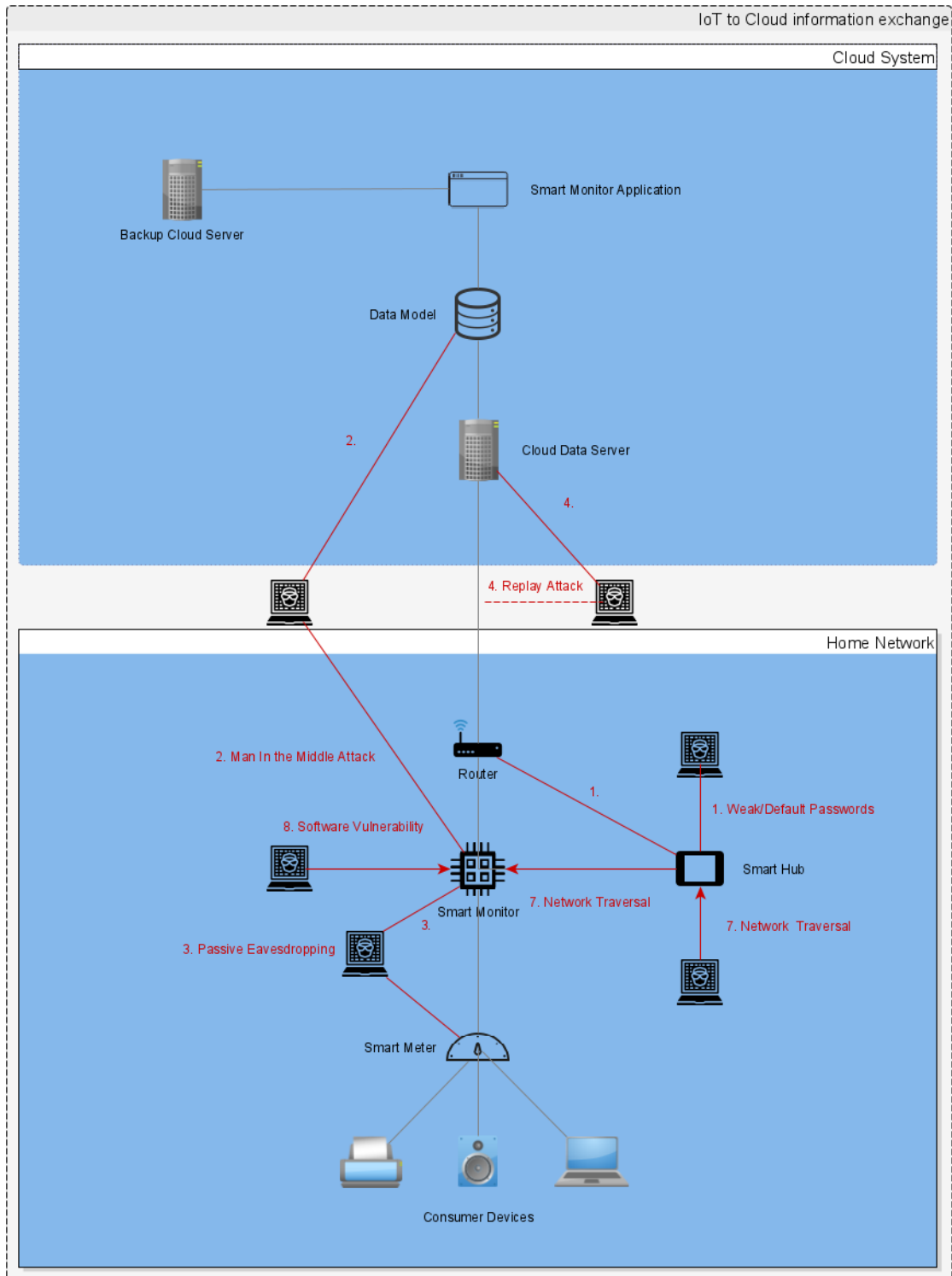


Figure 2: Threat Model of my smart grid scenario

A detailed breakdown of each threat and how they can be mitigated through the application of security protocols can be found in the preceding sections.

3.2.1 Weak/Default Password Fuzzing Attack

OWASP [15] states that the most common vulnerability exploited in IoT devices is the use of weak or default passwords. These attacks are usually performed by automated scripts commonly referred to as bots that scan the internet for connected devices and run a list of common passwords. As an example of this, the Mirai botnet was able to infect and recruit over 500,000 IoT devices using a list of just 60 common passwords.

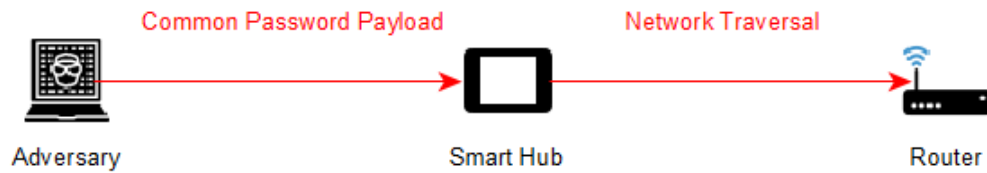


Figure 3: Adversary using a common password to compromise the network

In this scenario, an adversary could exploit an internet connected smart hub with a weak or guessable password to recruit the device into a botnet or potentially use the compromised device as an attack vector to mount further attacks on the rest of the network.

3.2.2 Man In The Middle (MITM) Attack

Man in the middle attacks occur when an adversary is able to act as an intermediary or proxy between communication parties without their knowledge. This allows the adversary to view the contents of the messages sent between parties as well as silently modify their contents.

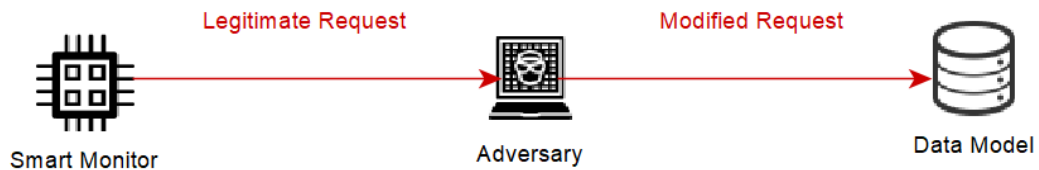


Figure 4: Adversary relaying and modifying smart monitor data

An Adversary could perform a MITM attack by secretly relaying and modifying the electricity usage information sent to the data model. A large scale attack of this kind effecting many monitor to model connections could cause a false data injection attack on the smart grid system where false data could cause the system to make an incorrect decision when routing power.

3.2.3 Passive Eavesdropping

Low power IoT devices commonly use weak or no cryptography in their communications protocols, this means an adversary could use devices such as a wireless packet sniffer to intercept traffic sent between devices and read the contents of the communications sent. OWASP [15] lists these insecure protocols as the 2nd most common IoT vulnerability.



Figure 5: Adversary reading an insecure communication

This attack could occur anywhere in the scenario where devices communicate with each other without the use of an encrypted communication protocol. For example, the adversary could sniff packets between the smart meter and monitor to know if a home is occupied based on their current electricity usage or to gather information on the network for further attacks.

3.2.4 Replay Attack

Replay attacks occur when an adversary is able to identify and collect authentication credentials from a legitimate communication and use those credentials in a later communication to bypass authentication. This commonly occurs when communication partners do not make use of a unique identifier for each communication such as a session key.

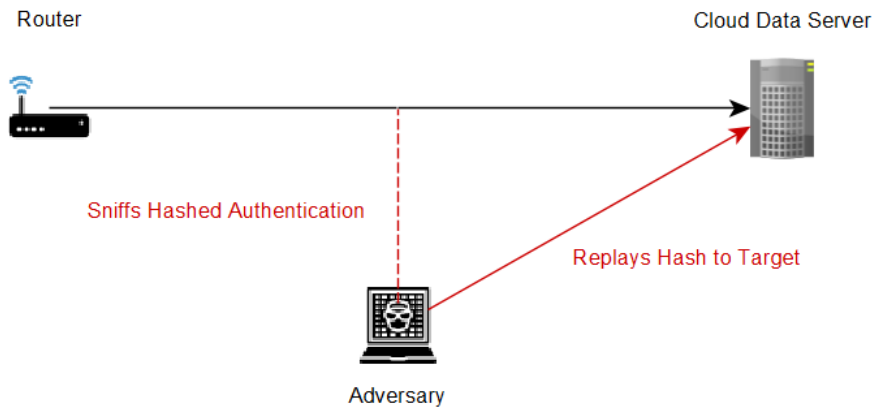


Figure 6: Adversary sniffing and reusing hashed authentication credentials

The adversary could sniff an encrypted communication between router and server used for the transmission of energy usage data. With this they could use the hashed authenticator code to send messages to the server posing as that home network without needing to know the actual authenticator code.

3.2.5 Impersonation Attack

An Impersonation attack occurs when an Adversary is able to pose as the identity of a legitimate party in a communication protocol allowing them to bypass authorisation or act on the legitimate user's behalf [16]. Protocols that do not use unique tokens for each communication are particularly susceptible to this form of attack.

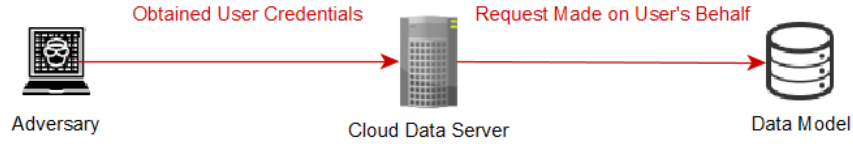


Figure 7: Adversary posing as a legitimate smart meter

An adversary could use this attack to pose as a monitor communicating data model and may use this to report false energy readings reducing trust in the system or use this access to perform further attacks against the infrastructure.

3.2.6 Open Port Scanning

An open port is a port number that a device accepts packets from. If ports are not configured correctly, adversaries can use an insecure port that has not been blocked as an attack vector. Botnet recruitment malware such as the Mirai botnet scan for insecure open ports to identify IoT devices that can be compromised. [17]

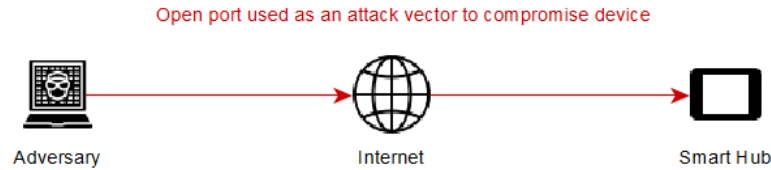


Figure 8: Adversary using an open port to attack a device

This attack can occur in the scenario where any devices are configured to allow network traffic in from unnecessary communication protocols. Whilst any unnecessary open ports increase risk exposure, communication ports such as Telnet (port 23) are particularly dangerous as they lack any in-built security measures.

3.2.7 Network Traversal

Network traversal can allow malware on a compromised network to spread to other networks connected to it. Malware's such as ransomware and worms often employ this technique to spread to as many users as possible. This is a serious issue for high security systems that have to interface with home networks as the home network is likely to have much poorer security controls.

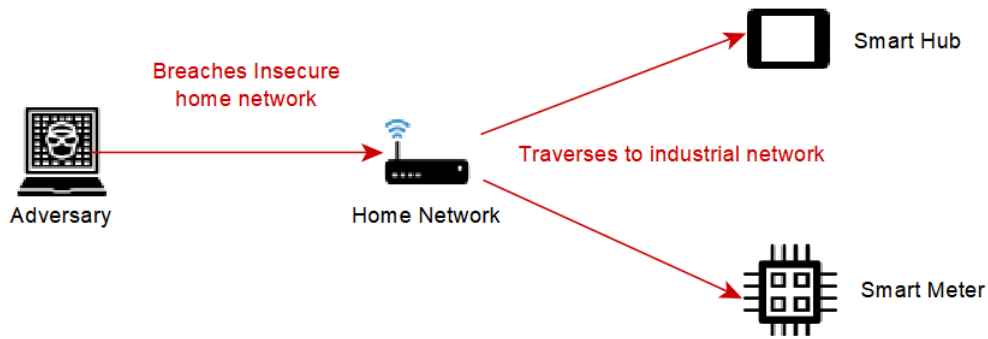


Figure 9: Malware traversing from an infected home network to industrial network

This attack could occur in the IoT scenario as the smart hub is connected to the homeowner's network and communicates with the smart monitor which is part of the industrial network. If a malware on a user's home network is able to compromise the smart monitor, an adversary could gain remote control over the device and use it to make further attacks on the system.

3.2.8 Software Vulnerability

As time passes, it is likely that vulnerabilities will be found in a given piece of software, patches are usually released quickly to fix these vulnerabilities. However, some of the most devastating cyberattacks of recent years relied on already patched exploits to work. A clear example of this is the famous WannaCry ransomware, the Windows vulnerability that the malware exploited had been patched over a month before it appeared [18]. Despite this, thousands of organisations were hit that did not patch this vulnerability including critical national infrastructure such as the National Health Service.

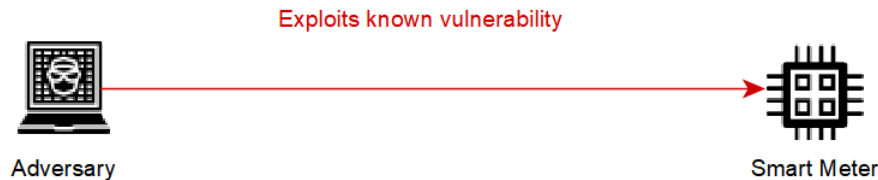


Figure 10: Adversary exploiting an unpatched IoT device

Software vulnerabilities could occur on software used by any of the devices within the smart grid system. Patches must be applied in a timely and sensible fashion to minimise the risk exposure one of these vulnerabilities being exploited.

4 Recommendation of policies and practices

Based on the threats identified above and research into the security needs of smart grid systems, a set of policies have been defined. These policies are split into two groups, communication policies which define the standards that communications between devices in the scenario must meet to ensure they are cryptographically secure. As well as best practices which define non-communication policies to protect the devices in the scenario against other forms of attack.

4.1 Communication Policies

These are the policies which are to be modelled and verified using Scyther. A brief description of each policy is defined below explaining why they have been included, as well as any potential drawbacks that implementing each policy may cause.

Message Encryption

This policy is the most fundamental aspect of a cryptographic protocol and is designed to mitigate the threat of passive eavesdropping. Message encryption defines that the contents of all communications between parties should be encoded in such a way that only those with access to the decryption key can decode and read the message.

In a 2018 investigation into the impact of the popular encryption standard AES on IoT communications, Hung CW and Hsu WT [19] found that Hardware AES increased power consumption of the average communication by 31%. Despite this increased power draw, encryption is an essential component of a cryptographic protocol as without it, an adversary can simply eavesdrop on messages in transit and view their contents as shown in **section 4.2.3**.

Implicit Key Authentication

Implicit key authentication is a policy that will be implemented during the key distribution process. This process is where communication parties share the secrets keys allowing them to generate session keys for the messages they send between each other. The policy defines that only the authorised parties should be able to access these keys. Without this policy, an adversary could perform a man in the middle (**section 4.2.2**) or impersonation (**section 4.2.5**) attack.

The reason for implementing a key distribution process is because symmetric encryption protocols require a shared secret key between communication parties in order to function. Whilst it is the case that asymmetric encryption algorithms do not require this step, asymmetric encryption is also more computationally intensive as it takes more CPU cycles to encrypt and decrypt messages. This factor becomes significant when considering the low computational power possessed by the IoT devices with the scenario and the need to communicate energy readings frequently.

Session Keys

Session keys are a randomly generated keys that parties agree on to encrypt and decrypt a single communication. These keys are generated and shared with the assistance of the secret keys that the parties already shared in the key distribution process. Unique session keys are being implemented to mitigate the threat of replay attacks (**section 4.2.4**) as they prevent the use of intercepted credentials. This is because the session key intercepted from a previous communication will never be reused as the session keys are regenerated for each new communication.

Mutual Authentication

Mutual Authentication requires that both parties are able to authenticate and trust the identity of each other. This policy is being implemented to prevent impersonation attacks (**section 4.2.5**). Particularly impersonation attacks that involve the mass creation of fake meters which send false information to the cloud server as well as the prevention of false cloud server identities causing meters to send their information to an adversary's server instead of the smart grid cloud server.

4.2 Description and specification of IoT Best Practices

The policies defined in this section are designed to mitigate the non-communication based threats a smart grid system faces. The sections below justify the need to implement each practice and provide information how each practice can be implemented into a smart grid scenario.

4.2.1 Password Management

The password management policy is designed to prevent the success of popular automated password fuzzing attacks which often target IoT devices (**section 4.2.2**)

In a study examining user behaviour toward password policies, Inglesant [20] found that excessively restrictive password policies with too much of a focus on password complexity caused users to adopt insecure workarounds such as writing down passwords or using the same password across multiple accounts. Therefore, a good password management policy should aim to balance the need for users to choose a password that is unique and complex enough to not fall victim to common password lists and brute force attacks whilst also not being too restrictive or complicated that the user has to resort to insecure methods of remembering it.

Based on this, the following key points are recommended for the implementation of an effective IoT password management policy:

- Default passwords should be a 15 character string of random characters, this serves two purposes by making the default password less susceptible to fuzzing attacks whilst also encouraging the user to change their password from the default due to the inconvenience of entering the default password.
- Passwords must be at least 8 characters long. This is the minimum length recommended by NIST for memory based authentication methods as of 2017 [21]. Whilst even longer passwords do provide more security, a minimum of 8 allows the password to be easier for the user to remember, reducing the chance of insecure workarounds being used.
- Before hashing, passwords should be checked against OWASP's top 10,000 password list [22]. This is because password fuzzing attacks look to try the most common passwords before using brute force methods.
- User created passwords must not be reused across other platforms. User credentials stolen in large commercial data breaches are usually sold online, this often occurs without the user knowing that their password has been leaked. The database Collection#1 was found by security researchers on the dark web in 2019 and it contained over 21 million leaked passwords.

One notable exception from this list common in other password policies is a requirement to change the password every set period. Whilst in circumstances where the expectation on the user is higher for managing their passwords such as a organisational account with access to confidential information. It was not deemed necessary here as the advantage of having a password change every few months is outweighed by the frustration this will causes users likely leading to the use of low effort passwords.

4.2.2 Network Segregation

Whilst the IoT smart grid system as a whole is an industrial system, meters and monitors are installed in peoples homes. Because of this, the implementation of a network segregation policy is necessary to prevent vulnerabilities in other networks manifesting in the IoT system (**section 4.2.7**) . The main challenge when considering the given scenario is the smart hub. The smart hub connects to the homeowner's WiFi network and visually displays information about their electricity usage to them. The concern is that without proper controls, a poorly secured home network could provide a route for malware to enter the industrial system.

To effectively segregate a network from another two criteria are required. The first criteria is to reduce the points of contact between the two networks as much as possible, networks that are deeply intertwined make the sanitisation of network traffic between them much harder. In the given IoT scenario this has

been reduced to a single point of contact between the smart monitor and smart hub. As the smart hub has no direct contact to either the cloud server or the smart meters in the household. The number of attack vectors for network traversal is decreased and therefore securing the points of contact is less complex.

The second criteria is using a well configured firewall at points of contact between the two networks. Firewalls are a security control used to monitor and block communications between parties based on a set of defined rules. In the case of the IoT scenario, rules should be defined to only allow outgoing traffic from the smart monitor to the smart hub, this can be implemented as the smart hub's function is to display information to the homeowner that the smart monitor sends it therefore a one way communication is sufficient to do this.

4.2.3 Patch Management

Software vulnerabilities being discovered over time are an inevitable reality that every secure system must plan for. When a vulnerability is discovered in software used by a system there are two options for mitigation. The first is to isolate the system from any outside connections and employ restrictive policies to limit how the system can be accessed. This is not an option in a smart grid scenario as the IoT devices need to communicate with each other to perform their core functionalities, the distributed layout of a smart grid also makes isolation much more challenging as devices are spread across different networks and physical locations. The second option is apply a security patch fixing the vulnerability. Large, always on and distributed systems such as smart grids face difficult challenges when managing the application of patches as applying a security update across the system is a much more time intensive process compared to a traditional network. This becomes an important factor when considering that a smart grid system does not have inactive hours where software updates can be installed overnight.

Patch bundling is a technique which can be used to reduce the number of security patches required over a period of time by bundling all required updates into a single patch which is applied every 3 months. However, this comes with significant drawbacks as it introduces a delay between the time a patch is released and the time it applied, this increases the time that the system is vulnerable and therefore the chance that an adversary will exploit it. NIST recommends that patches be prioritised based on the threat that a vulnerability poses to a system. With severe vulnerabilities that are being actively exploited, patches should be applied as soon as possible whereas the patching of vulnerabilities which pose a lower threat can be delayed until the next patch bundle [23]. CVSS(Common Vulnerability Scoring System) is a method of rating the threat that a vulnerability poses on a 0-10 scale, it takes into account the severity of vulnerability if it were to be exploited, how easy it is to exploit and how much access an adversary needs to exploit it. By using this score, a patch management policy can be created for each level of threat.

Table 4: Patch Management policy based on the CVSS score of a vulnerability

CVSS score	Patching Policy
0-4	Updates for these vulnerabilities pose a reduced threat to the system and should only be implemented as part of the next planned patch.
4-6	Updates for vulnerabilities in this category should be evaluated on a case by case basis and either implemented as part of the next planned patch or applied after a week to give time for testing
6-8	Updates for these vulnerabilities should be applied in the week that they are released.
8-10	Vulnerabilities in this category must be patched as soon as possible, if an update is not yet available the possibility of workarounds or reducing functionality to mitigate the threat should be considered.

4.2.4 Minimum Design

Minimum design refers to the practice of limiting the configuration of a device to the absolute minimum required for the device to function within its role. A well implemented minimum design policy reduces a device's risk exposure to cyberattacks whilst not compromising its functionality. One of the key threats the minimum design policy counters is open port scanning (**section 4.2.7**). By default the IoT devices used in the smart grid system will have been configured to accept packets from a large range of network services. However, the vast majority of these ports are not necessary for their functionality within the smart grid system.

To limit these network services there are two opposing filtering techniques that can be used, blacklisting and whitelisting. Blacklisting refers to creating a list of in this case network services that are to be filtered out whereas whitelisting refers to filtering out everything except for a specific list of services. In this scenario whitelisting is the most sensible option as the smart devices have explicit communication roles and all other communication ports are not required.

5 Implementation and modelling of communication protocol policies

5.1 Message Encryption

The first policy to be implemented is message encryption. The aim of this policy is to mitigate the threat of passive eavesdropping. Whilst the implementation of this policy alone forms a very basic protocol, it makes up the backbone that the more advanced protocols will be based on.

Policy Criteria

- Message contents must be secure against a adversary passively eavesdropping on the communication.
- The monitor must be able to decrypt and read the message contents from the meter.

5.1.1 Design

The protocol's design is a basic symmetric key encryption protocol where the Meter sends the Monitor a message containing electricity usage information. Once the message is received, the Monitor sends back a confirmation message.

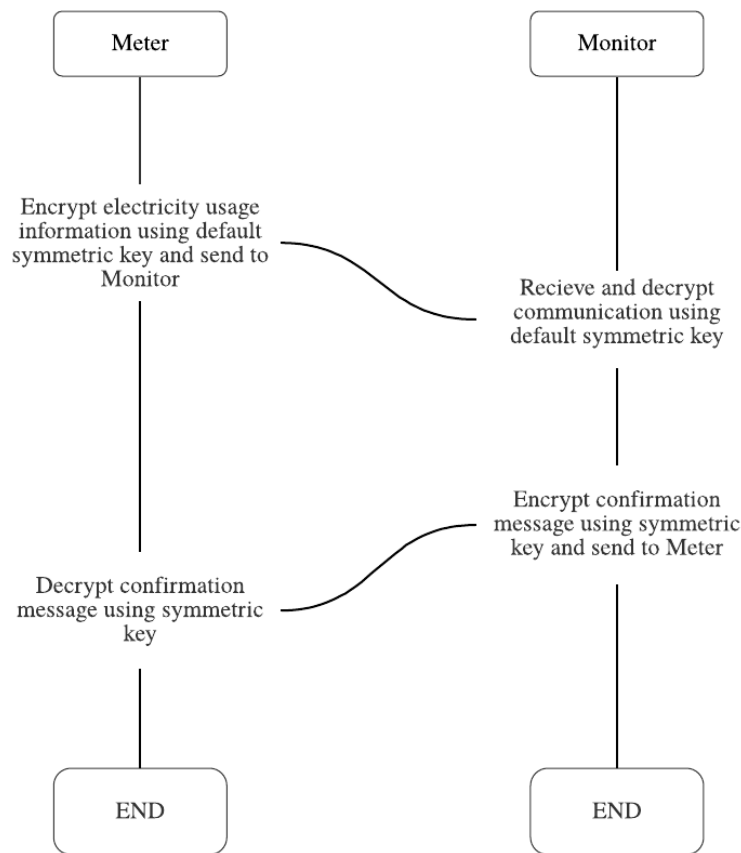


Figure 11: Message encryption protocol design

5.1.2 Implementation

To model the symmetric encryption method used for this protocol, the key (k) is used to represent a symmetric key. As this protocol does not feature a key distribution element, the key is a default key loaded onto each device as part of their initial configuration for use in the smart grid system.

```
0 protocol smartExchange(Meter,Monitor)
1
2   {
3
4   role Meter {
5
6   fresh Message: Nonce;
7   var Confirm;
8
9   send_1(Meter,Monitor,{Message}k(k));
10
11  recv_2(Monitor,Meter,{Confirm}k(k));
12
13  claim_Meter1(Meter, Secret, (k));
14  claim_Meter2(Meter, Secret, Message);
15
16  }
17
18  role Monitor {
19
20  var Message;
21  fresh Confirm: Nonce;
22
23  recv_1(Meter,Monitor,{Message}k(k));
24
25  send_2(Monitor,Meter,{Confirm}k(k));
26
27  claim_Monitor1(Monitor, Secret, (k));
28  claim_Monitor2(Monitor, Secret, Message);
29
30  }
31
32 }
```

Figure 12: Message encryption protocol design

5.1.3 Review

To model the criteria of the message contents not being readable to an eavesdropping adversary, Scyther's Secret claim was made on the message which models an adversary attempting to eavesdrop on the message during communication. By modelling this claim on both the Monitor and Meter it can also be used to check that the monitor has received the message thereby fulfilling the second policy criteria.

Without the implementation of the symmetric key encryption, running the secret claim generates a successful eavesdropping attack.

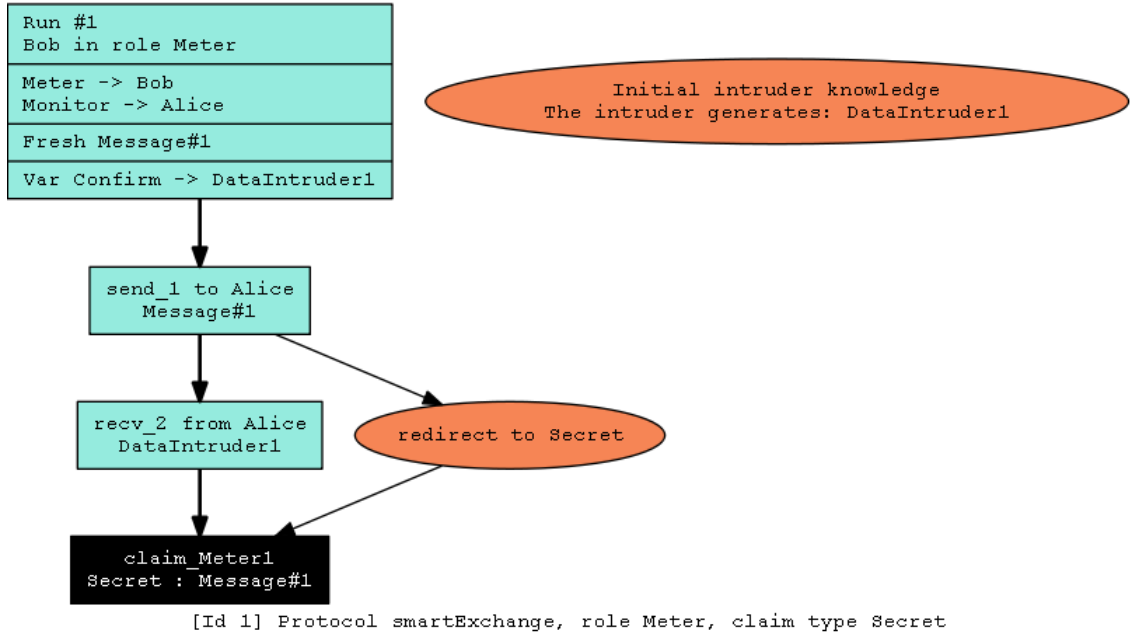


Figure 13: Message encryption protocol test results

The figure above shows that by eavesdropping the un-encrypted commutation, demonstrated in this case by DataIntruder1 the adversary can read the contents of the message therefore disproving the secret claim.

The first iteration of the protocol the defined tests successfully with Scyther showing that no attacks of this type are possible within the bounds of the protocol. Results using a wider range of claims however show that threats such as man-in-the-middle attacks can easily break this protocol demonstrating the need to iterate upon it and implement the remaining policies

Table 5: Log of claims made against the message encryption protocol and their results

Role	Claim	Result
Meter	Secret Message	Pass
	Secret key(k)	Pass
Monitor	Secret Message	Pass
	Secret key(k)	Pass

5.2 Implicit key authentication

Later policies will require the use of secret keys to create symmetric encryption protocols, implicit key authentication must be enforced when these keys are distributed to prevent an adversary posing as a legitimate communication party using an intercepted secret key. To implement this policy a secure key distribution protocol is required.

Policy Criteria

- The secret key must not be interceptable by an eavesdropping adversary
- Parties must be able to identify the sender of each step of the protocol to prevent an adversary impersonating a party to get the key.
- Parties must both have the same secret key at the end of the protocol.

5.2.1 Design

When looking to design this protocol, a decision had to be made on which style of key distribution was most suitable. These two styles being considered are best represented by the two popular key distribution protocols signed Diffie-Hellman and Needham-Schroeder. Needham-Schroeder makes use of a trusted 3rd party to securely establish authentication which can be used to exchange symmetric secret keys over an insecure network as shown in the figure below.

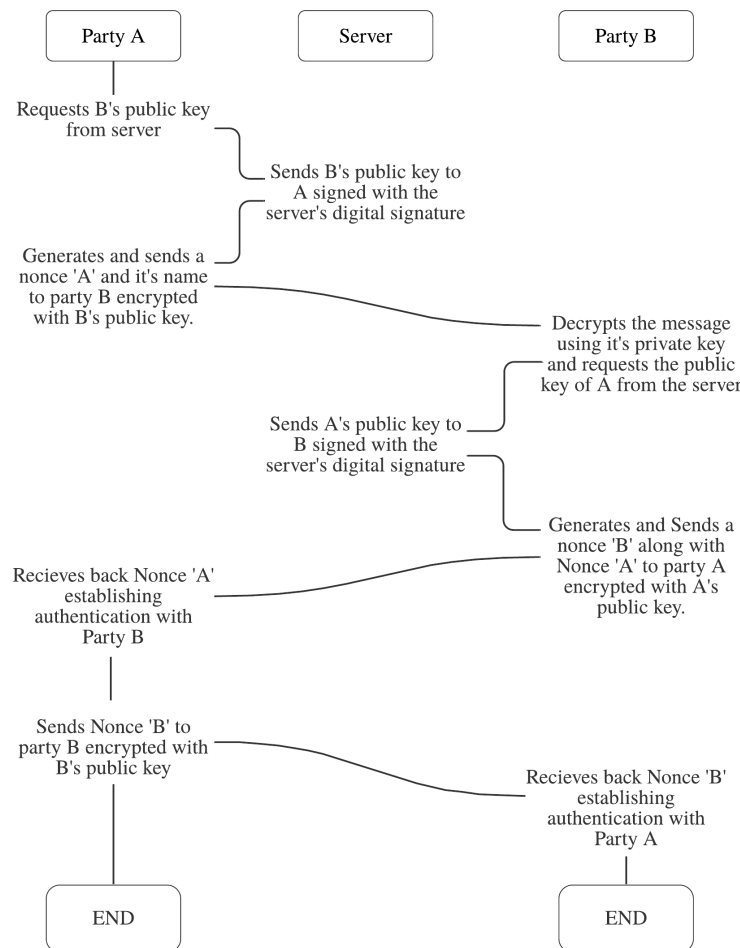


Figure 14: Illustration of the Needham Schroeder authentication process [24]

In comparison, Diffie-Hellman does not require the use of a third party and instead relies on the concept of never actually sending a shared key over an insecure network but instead exchanging information that along with secret information both parties possess, allows the parties to generate the same key which becomes the shared secret key to be used in future communications.

For the given IoT scenario a Diffie-Hellman style solution is the most appropriate based mainly on the fact that it does not require an additional third party. This makes the protocol less computationally complex for the IoT devices as they only need to communicate with each other, as well as negating the need infrastructure in the form of an additional server.

The design for this policy in the project scenario, as illustrated below, works by having the two communication parties each come up with a fresh value. The aim of the protocol is then to allow the two parties to exchange their values whilst ensuring that an adversary cannot either eavesdrop on the message or pose as one of the parties to gain both of the values.

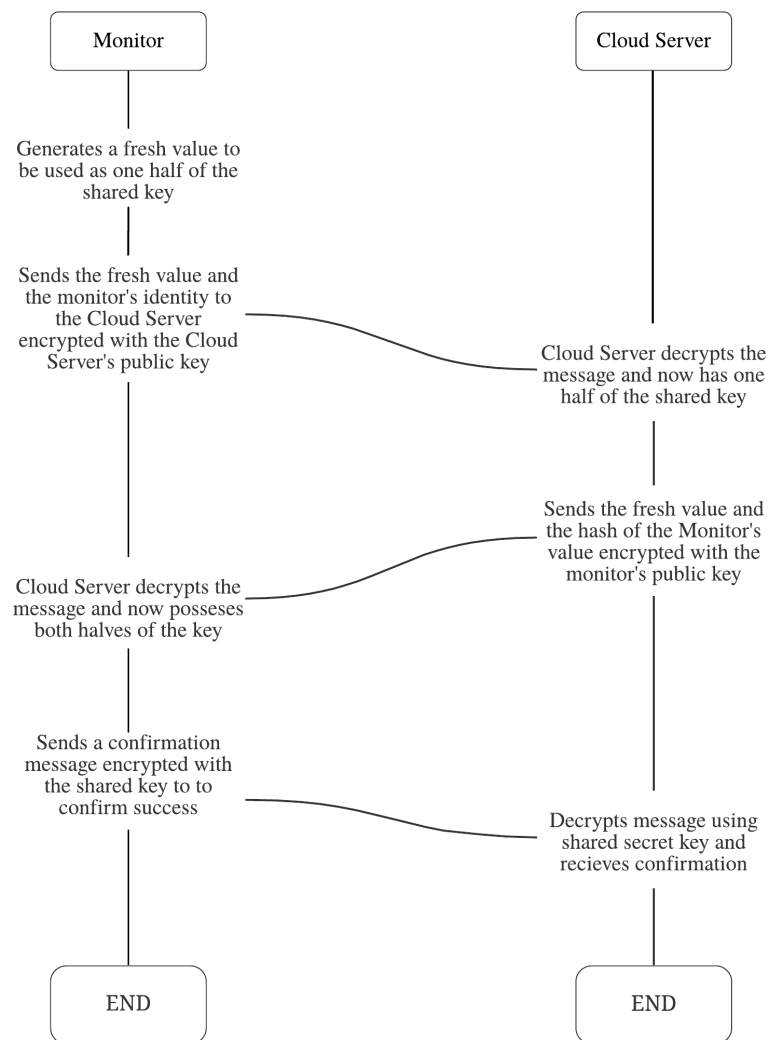


Figure 15: Illustration of the planned key exchange [24]

Once each party exchanges its generated value with the other, they are then combined together using a predetermined hash function. This means that the two values are inputted into a function that is computationally expensive to reverse engineer, because of this an adversary cannot work out what the

two inputs were from the resulting output. This output is then used as the shared secret key that all session keys for future communications will use as the base.

5.2.2 Implementation

```

0
1 hashfunction hashed;
2 usertype Message;
3
4 protocol KeyExchange(Monitor,CloudServer)
5 {
6     role Monitor
7     {
8
9
10        fresh MonitorValue : Nonce;
11        fresh Confirm: Message;
12        var CloudServerValue : Nonce;
13
14        send_1(Monitor,CloudServer,{Monitor,MonitorValue}pk(CloudServer));
15
16        recv_2(CloudServer,Monitor, {CloudServerValue,hashed(MonitorValue),
17        CloudServer}pk(Monitor));
18
19        send_3(Monitor,CloudServer, hashed(CloudServerValue, Confirm));
20
21        claim_Monitor1(Monitor,Niagree);
22        claim_Monitor2(Monitor,Nisynch);
23        claim_Monitor3(Monitor, Secret, MonitorValue);
24        claim_Monitor4(Monitor, Secret, CloudServerValue);
25
26    }
27
28    role CloudServer
29    {
30
31
32        var MonitorValue: Nonce;
33        var Confirm: Message;
34        fresh CloudServerValue: Nonce;
35
36        recv_1(Monitor,CloudServer,{Monitor,MonitorValue}pk(CloudServer));
37
38        send_2(CloudServer,Monitor, {CloudServerValue,hashed(MonitorValue),
39        CloudServer}pk(Monitor));
40
41        recv_3(Monitor,CloudServer, hashed(CloudServerValue, Confirm));
42
43        claim_CloudServer1(CloudServer,Niagree);
44        claim_CloudServer2(CloudServer,Nisynch);
45        claim_CloudServer3(CloudServer, Secret, MonitorValue);
46        claim_CloudServer4(CloudServer, Secret, CloudServerValue);
47
48    }
49 }

```

5.2.3 Review

To test the criteria that the secret key is not interceptable through eavesdropping, Scyther's Secret claim will be used on both the cloud server and monitor value for both communication parties. If these secret claims hold, the criteria will be verified as an adversary would need to eavesdrop on at least one of the values to intercept the key.

By using the Niagree and Nisynch claims, Scyther can verify if the roles within a protocol are able to authenticate the identity of a message sender. The Ni prefix denotes that the attack scope is limited to non-injective attacks, these are attacks that do not assume the adversary has knowledge from a previous run of the protocol [25]. If these claims hold then a impersonation attack is not possible and the 2nd criteria which specifies security against this form of attack will be verified.

The 3rd Criteria, which states that both parties must be able to agree on a shared secret key can be verified by ensuring that both parties posses both the cloud server and monitor value at the end of the protocol. This can again be verified using the Secret claim as it will only pass if the role specified does have access to the object the secrecy claim is being made on.

Table 6: Log of claims made against the key distribution protocol and their results

Role	Claim	Result
Monitor	Niagree	Pass
	Nisynch	Pass
	Secret, MonitorValue	Pass
	Secret, CloudServerValue	Pass
Cloud Server	Niagree	Pass
	Nisynch	Pass
	Secret, MonitorValue	Pass
	Secret, CloudServerValue	Pass

The required claims all pass for this protocol demonstrating that it is fit for purpose and allows for the secure distribution of secret keys.

5.3 Unique Session Keys

Implementing fresh session keys between parties for each communication is a policy aimed at preventing replay attacks when the monitor sends energy usage information to the cloud server. Communication parties have established a shared secret key using the key distribution policy in the section above. A protocol is now required that generates session keys using this secret key and defines how messages are to be encrypted with them.

Policy Criteria

- The session key used for each communication must be unique
- The cloud server must be able to authenticate that the monitor it sends information to is legitimate

5.3.1 Design

By deriving session keys from the shared secret key now present between both communication parties, it is possible to design a simple protocol that meets the policy criteria. Simplicity is an important factor for this protocol as the communication between the monitor and cloud server where current electricity usage is sent is very frequent and therefore should be as lightweight as possible.

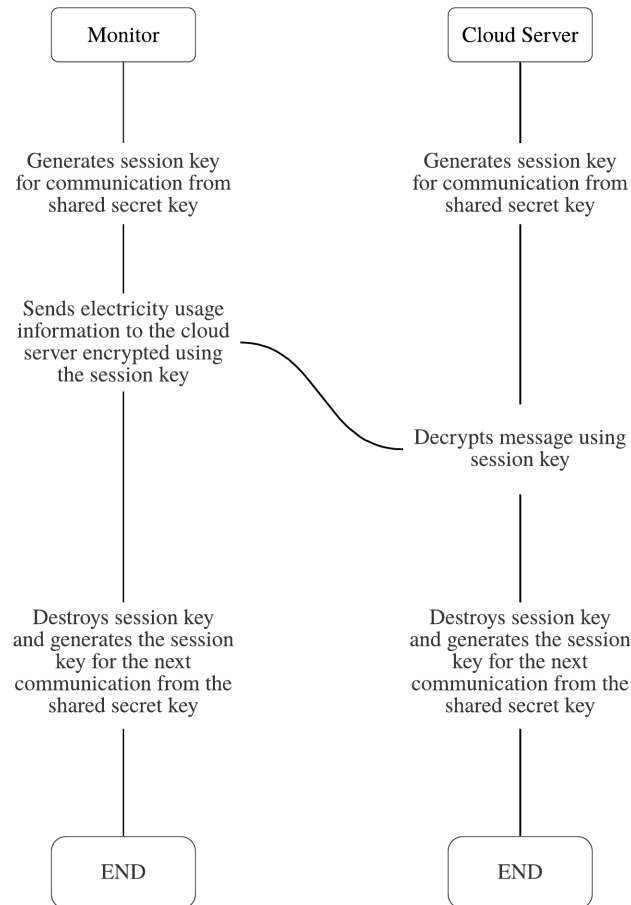


Figure 16: Illustration of a one-way communication utilising session keys [24]

5.3.2 Implementation

In Scyther, the SessionKey usertype can be used to model a session key. The fresh prefix during the initialisation of the key (line 14 and line 24) denotes that a new key is generated for each communication. To model the information exchange between the two parties after the session key is established, a Message user type has been defined.

```
0 hashfunction hashed;
1 hashfunction sharedkey;
2 usertype Message;
3 usertype SessionKey;
4
5 protocol SessionKeys(Monitor,CloudServer) {
6
7   role Monitor {
8
9     fresh MonitorValue : Nonce;
10    var CloudServerValue : Nonce;
11
12    fresh info : Message;
13    var info: Message;
14    fresh sharedkey: SessionKey;
15
16    send_1(Monitor,CloudServer,{Monitor,MonitorValue}pk(CloudServer));
17
18    recv_2(CloudServer,Monitor, {CloudServerValue,hashed(MonitorValue),
19    CloudServer}pk(Monitor));
20
21    send_3(Monitor,CloudServer, {info} sharedkey);
22
23    claim_Monitor1(Monitor,Alive);
24    claim_Monitor2(Monitor,Secret, info);
25  }
26
27  role CloudServer {
28
29    var MonitorValue: Nonce;
30    fresh CloudServerValue: Nonce;
31
32    var info: Message;
33    fresh info: Message;
34    fresh sharedkey: SessionKey;
35
36    recv_1(Monitor,CloudServer,{Monitor,MonitorValue}pk(CloudServer));
37
38    send_2(CloudServer,Monitor, {CloudServerValue,hashed(MonitorValue),
39    CloudServer}pk(Monitor));
40
41    recv_3(Monitor,CloudServer, {info} sharedkey);
42
43    claim_CloudServer1(CloudServer,Niagree);
44    claim_CloudServer2(CloudServer,Nisynch);
45    claim_CloudServer3(CloudServer,Alive);
46    claim_CloudServer4(CloudServer,Secret, info);
47  }
48 }
```

5.3.3 Review

To model the policy criteria in Scyther, the Alive and Niagree/Nisynch claims will be used to verify that the cloud server is able to authenticate the monitor, for this protocol only one way authentication is required due to the protocol being one-way. Alive verifies that the protocol has completed all the steps of the protocol in the specified order. This ensures that the session key is agreed upon before communications occur and is how the unique session key requirement for this protocol will be verified.

As shown in the figure below, without the implementation of session keys it is trivial for an adversary to reuse credentials from previous communications.

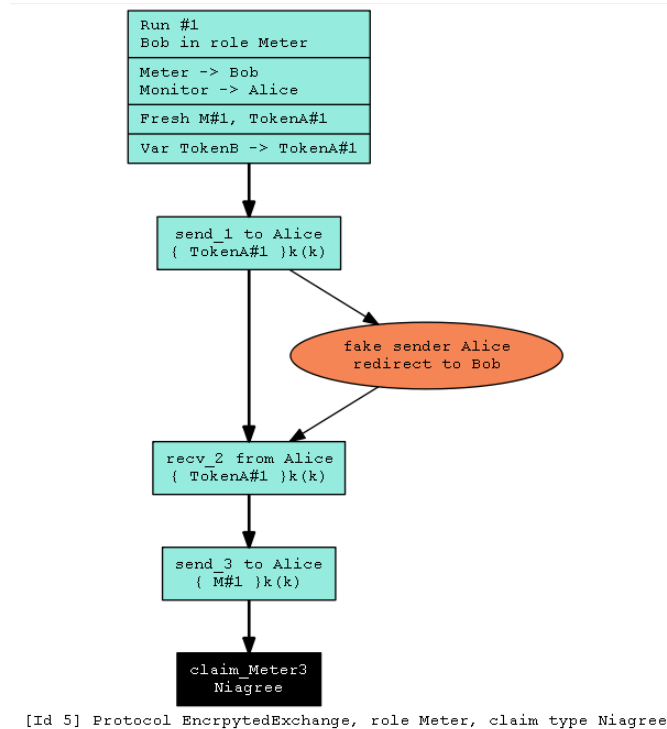


Figure 17: Replay attack before the implementation of session keys

After the implementation of session keys this replay attack is no longer possible and the protocol is able to meet all of the specified criteria as shown by the results table below .

Table 7: Log of claims made against the session key protocol and their results

Role	Claim	Result
Monitor	Monitor, Alive	Pass
	Secret, info	Pass
Cloud Server	Niagree	Pass
	Nisynch	Pass
	Cloud Server,Alive	Pass
	Secret, info	Pass

5.4 Mutual Authentication

Whilst the session keys protocol is effective at establishing a secure connection between monitor and cloud server as well as allowing the cloud server to authenticate the identity of the monitor sending information. It is possible for an adversary to pose as a cloud server and establish a communication channel with the monitor. This would allow the adversary to send malicious information to smart monitors such as fake electricity information or potentially use the communication channel as an attack vector for malware.

Policy Criteria

- The session key used for each communication must be unique
- The Monitor must be able to authenticate that the Cloud Server it sends information to is legitimate
- The Cloud Server must also be able to authenticate that the Monitor it sends information to is legitimate

5.4.1 Design

The design for this protocol is based of the design of the preceding session keys protocol. Due to the increased number of messages sent in a single round of this protocol and the need for additional authentication, this protocol will not be as lightweight as the session keys protocol.

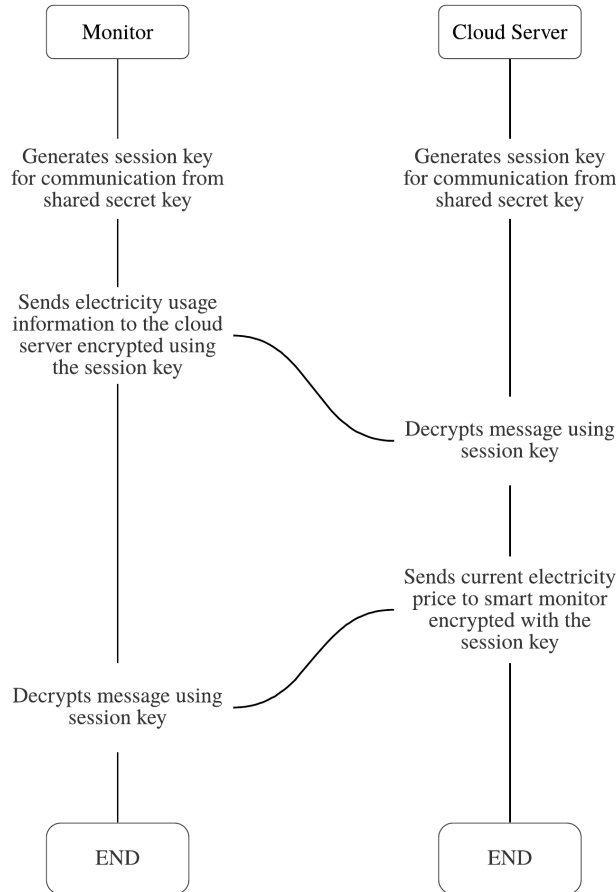


Figure 18: Illustration of a two-way mutually authenticated protocol [24]

However, the two-way communication where the Cloud Server sends electricity pricing information to the smart monitor as well as receiving electricity usage does not occur as frequently as the one-way sending of electricity usage meaning the complexity of this protocol is not as much of a concern.

5.4.2 Implementation

```

0  hashfunction hashed;
1  hashfunction sharedkey;
2  usertype Message;
3  usertype SessionKey;
4
5  protocol MutualAuthentication(Monitor,CloudServer)
6  {
7      role Monitor
8
9      {
10
11          fresh MonitorValue : Nonce;
12          var CloudServerValue : Nonce;
13
14          fresh MonitorInformation : Message;
15          var CloudServerInformation: Message;
16          fresh sharedkey: SessionKey;
17
18
19          send_1(Monitor,CloudServer,{Monitor,MonitorValue}pk(CloudServer));
20
21          recv_2(CloudServer,Monitor, {CloudServerValue,hashed(MonitorValue),
22          CloudServer}pk(Monitor));
23
24          send_3(Monitor,CloudServer,{CloudServerValue, MonitorInformation} sharedkey );
25
26          recv_4(CloudServer,Monitor,{MonitorValue,CloudServerInformation} sharedkey);
27
28          claim_Monitor1(Monitor,Niagree);
29          claim_Monitor2(Monitor,Nisynch);
30          claim_Monitor3(Monitor, Secret, CloudServerInformation);
31          claim_Monitor4(Monitor,Alive);
32
33      }
34
35      role CloudServer
36
37      {
38
39          var MonitorValue: Nonce;
40          fresh CloudServerValue: Nonce;
41
42          fresh CloudServerInformation: Message;
43          var MonitorInformation: Message;
44          var sharedkey: SessionKey;
45
46          recv_1(Monitor,CloudServer,{Monitor,MonitorValue}pk(CloudServer));
47          send_2(CloudServer,Monitor, {CloudServerValue,hashed(MonitorValue),
48          CloudServer}pk(Monitor));
49
50          recv_3(Monitor,CloudServer, {CloudServerValue, MonitorInformation} sharedkey );
51
52          send_4(CloudServer,Monitor,{MonitorValue,CloudServerInformation} sharedkey);

```

```

53
54     claim_CloudServer1(CloudServer, Niagree);
55     claim_CloudServer2(CloudServer, Nisynch);
56     claim_CloudServer3(CloudServer, Secret, MonitorInformation);
57     claim_CloudServer4(CloudServer, Alive);
58
59 }
60 }

```

5.4.3 Review

To model the mutual authentication policy criteria that both parties must be able to authenticate the identity of each other, Niagree/Nisynch as well as the alive claim will be verified for both the cloud server and monitor, this provides assurance that both parties in the communication are able to authenticate each other. Secret claims on the monitor and cloud server information will be used to verify that the message contents cannot be intercepted through eavesdropping. To verify that the session key is unique the Alive claim will be used as it was in the session key protocol.

Table 8: Log of claims made against the mutual authentication protocol and their results

Role	Claim	Result
Monitor	Niagree	Pass
	Nisynch	Pass
	Secret, CloudServerInformation	Pass
	Alive, Monitor	Pass
Cloud Server	Niagree	Pass
	Nisynch	Pass
	Secret, MonitorInformation	Pass
	Alive, CloudServer	Pass

As shown by the Scyther results, this protocol has been verified to meet all of the policy criteria. This protocol implements all of the preceding policies as well and therefore is cryptographically secure against all the identified communication based threats in the threat model (**section 4.2**).

6 Project Management

All deliverables for this project were completed within the required deadlines, this section details the measures that were taken to manage the project and ensure that this was the case

The main tool used for macro project management was the Gantt chart. The chart below details how I planned to complete the required tasks for this project.

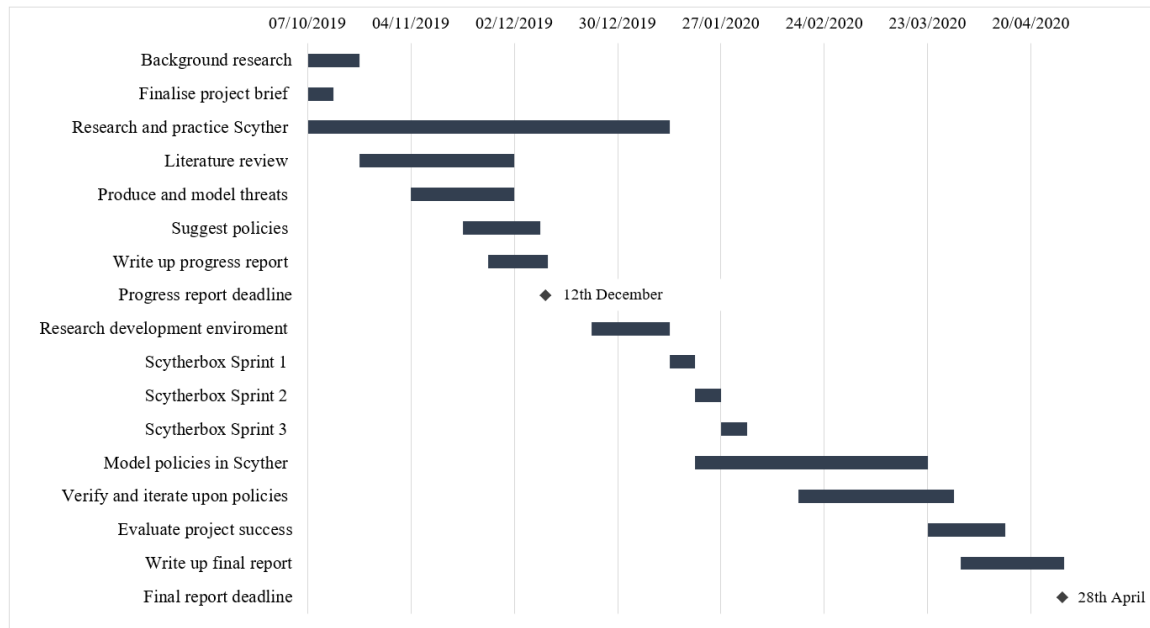


Figure 19: Gantt chart detailing planned project scheduling

The majority of the project plan featured time spent on either learning or developing using the Scyther tool. This was a good decision in hindsight as coming into Scyther for the first time was a challenging experience and it took time to learn how the software worked before I could produce anything of value.

6.1 Coronavirus mitigation

The threat that access to Scyther and university computers could be lost was identified on my risk matrix and was one of the reasons the creation of a portable Scyther development environment is part of this project. As a result of this, the effect that coronavirus had on my ability to complete my project goals was negligible. Despite this, the virus had a large impact on the scheduling of the project as shown by the gantt chart below detailing how project work was actually completed.

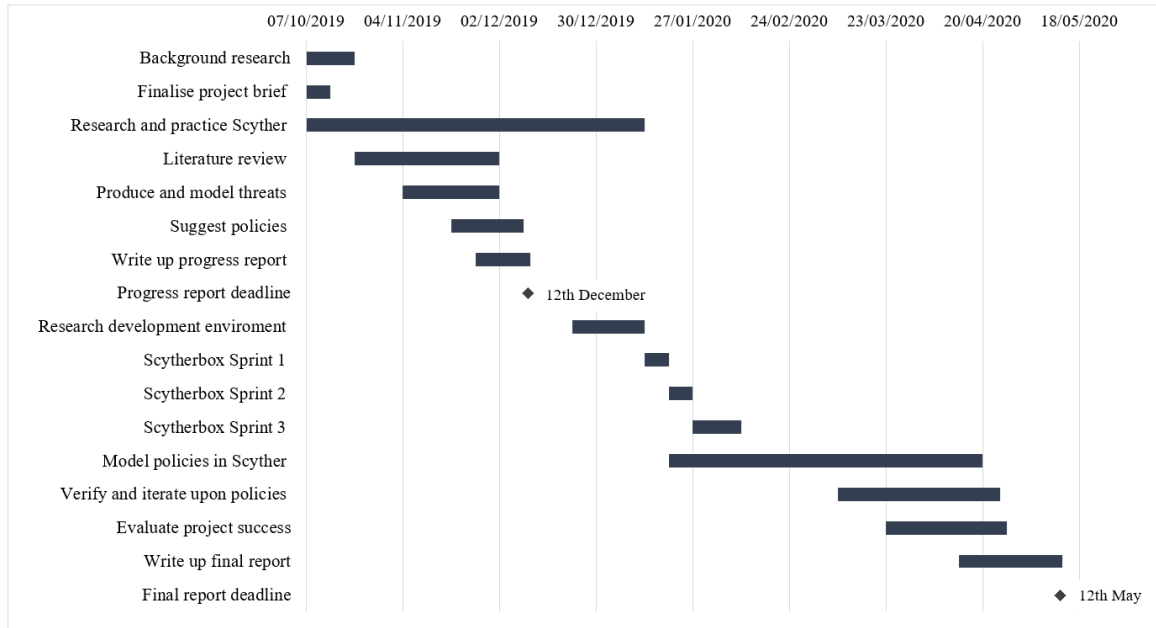


Figure 20: Gantt chart detailing how work on the project was actually completed

Tasks in progress around the 23rd of March such as the modelling of policies in Scyther took longer than anticipated due to relocating home, preparing for and settling into the quarantine however the 2 week deadline extension given negated this.

To stay in contact with my supervisor and ensure I still benefited from their feedback without physical meetings, Microsoft Teams was used to schedule online meetings. The project is hosted on the University GitLab and issue boards were used to track my progress through each task as well as keep my supervisor informed about what I was working on at a given time.

6.2 Risk Management

To catalogue and plan mitigation against risks which could pose issues for the project, a risk management matrix was created outlining the key risks the project faces. To measure the threat level each risk posed a scoring system was used, each risk was rated out of 5 based on how likely the risk was to occur and the magnitude of impact if it did, these values were then multiplied together to give a risk score out of 25. Each risk was assigned a baseline score which represents the level of risk before any mitigation was put in place and a residual risk score showing the level of risk after the mitigation plan is put in place.

Table 9: Qualitative risk analysis and mitigation plan for key project risks

Risk	Baseline	Mitigation	Residual
Scyther stops being supported/allowed on the university computers and I lose my access to the software	Impact: 5 Likelihood: 2 Score: 10	I am using vagrant to set-up a box with Scyther and all the software required to run it installed so I always have it available, the vagrant box has a cloud backup. This allows for the easy transfer of work over to personal devices	Impact: 5 Likelihood: 0 Score: 0
I fail to manage time correctly on the project and do not finish parts	Impact: 4 Likelihood: 3 Score: 10	My Gantt chart will help when identifying if I am falling behind schedule on certain parts. Meeting weekly with my supervisor where I share my progress will also help me hold myself accountable for work.	Impact: 4 Likelihood: 1 Score: 4
My laptop is lost, stolen, or damaged causing me to lose all the content on the hard drive	Impact: 4 Likelihood: 2 Score: 8	My project files are uploaded to Git and frequently pushed to the remote branch when I make changes. I can continue to work on my desktop and the university computers.	Impact: 3 Likelihood: 1 Score: 3
Sections of work are larger or more difficult than I anticipated meaning I fail to complete parts of the project	Impact: 4 Likelihood: 3 Score: 12	My background research and experience of learning Scyther in the last month has helped me estimate the difficulty of each task. The scheduling of tasks in the project will assign extra time in each task for potential delays that may occur due to unexpected difficulties.	Impact: 3 Likelihood: 2 Score: 6
Personal/family issue occurs	Impact: 4 Likelihood: 3 Score: 12	Whilst this risk is difficult to mitigate, the university support service will be used if needed and I will keep my supervisor informed of any issues.	Impact: 3 Likelihood: 3 Score: 9

6.3 Scyther development environment

To manage and prioritise the development of the Scytherbox, requirements were prioritised using MoSCoW and divided into development sprints, allowing for the rapid creation of a development environment with the 'Must' requirements. This allowed for the modelling of protocols to begin within the scheduled time-frame.

The first sprint featured the essential elements required to develop using Scyther, development for this sprint went largely as expected and after overcoming an issue in the first two days of the sprint relating to creation of the initial base box, the remaining tasks fell away quickly.

Hours Remaining	Sprint 1: Work Breakdown						
6	Shared Folder	Shared Folder					
5	Versionable using git	Versionable using git					
4	Scyther Dependencies Installed	Scyther Dependencies Installed	Shared Folder				
3			Versionable using git				
2	Scyther Installation	Scyther Installation	Scyther Dependencies Installed	Shared Folder			
1				Versionable using git	Shared Folder	Shared Folder	
	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday

Figure 21: Breakdown of the work completed in the first sprint

One 'Could Have' requirement from Sprint 2 was not completed which was to allow the user to set up a Git configuration within the environment using the VagrantFile. After researching the methods that could be used to do this, I came to the conclusion that a VagrantFile git setup would be more time intensive to set up than I estimated in my requirements planning and it would be difficult to design in a way that is easy to use.

Hours Remaining	Sprint 2: Work Breakdown						
9	Scytherbox Documentation						
8		Scytherbox Documentation	Scytherbox Documentation	Scytherbox Documentation			
7							
6							
5	Linux options						
4	Git configuration within box	Git configuration within box	Git configuration within box	Git configuration within box	Git configuration within box	Git configuration within box	Git configuration within box
3							
2							
1							
	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday

Figure 22: Breakdown of the work completed in the second sprint

To Mitigate this issue I found that creating a section within the shared folder for Git repositories was a more elegant solution and much easier to use. This means that changes made to the repository in the development environment are automatically reflected in the host machine side of the shared folder. Changes can then be pushed to the remote git branch using the user's choice of git client from the host machine.

Sprint 3 featured the requirements that would improve the experience for others using the Scyther environment in the future. The implementation of the Scyther user guide differs from the original design as the official Scyther manual was found to explain the basics of Scyther development well. Instead of writing a general Scyther guide, this manual was included in the virtual machine.

Hours Remaining	Sprint 3: Work Breakdown						
6	Installation instructions	Installation instructions					
5							
4	Example Protocols within box	Example Protocols within box	Example Protocols within box				
3							
2	Scyther user guide	Scyther user guide	Scyther user guide	Scyther user guide	Scyther user guide	Scyther user guide	
1							
	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday

Figure 23: Breakdown of the work completed in the third sprint

7 Project Evaluation

This section examines the success of the project and details the future plans for this work.

7.1 Achievement against project goals

To evaluate the project's success, the achievements made during the course of the project will be compared against the initial project goals

1. Investigate and conduct a risk assessment on the main vulnerabilities and threats faced by IoT devices within a smart grid environment.

After extensive research, a threat model showing the key vulnerabilities faced by a smart grid system was produced. Each vulnerability was described, and the specific threat it posed to the smart grid system documented.

2. Recommend security policies that can mitigate these threats, justifying these policies by taking into account secondary factors including the cost to implement and any loss to productivity these policies might incur.

A set of policies based on industry standards was created to specifically mitigate the vulnerabilities identified during the threat analysis. Design choices for these policies were justified with supporting literature and where necessary the potential disadvantages of each policy were discussed.

3. Implement and verify that these communication protocols mitigate the identified vulnerabilities using Scyther, a formal method based protocol verification tool.

Communication policies and their criteria were modelled in Scyther, the review section for each policy explained how each criteria was converted into Scyther claims. The final versions of all policies passed their verification in Scyther confirming that they are fit for purpose.

4. Clearly explain the impact of each of my policies by comparing the possible attack vectors with and without each policy using Scyther.

Where appropriate, attack graphs were included in the review section of each policy illustrating the attacks possible before each protocol was implemented.

5. Create a purpose built, portable Scyther virtual machine environment allowing myself and others to quickly set up and start using Scyther on a new device. Therefore allowing others to verify and extend upon the results of the project.

The Scyther environment was successfully created and utilised in the modelling and verification of the project's policies. The versionable nature of VagrantFiles make it easy for others to set up and add to the virtual machine to suit their needs.

7.2 Conclusion

Though the course of this project, a set of security policies have been designed, implemented and verified to mitigate the threats that IoT devices within a smart grid system face. The project shows that whilst the introduction of smart grids to a country's power grid does introduce the possibility of cyberattacks against critical national infrastructure, with proper security controls these threats can be identified and mitigated. The project also explored the use of Scyther for the modelling and verification of security protocols, demonstrating it's ability to find flaws based on the security requirements defined for each protocol. Overall, the project has achieved it's primary goals and provides a overview of the security measures that can be taken to secure IoT devices within a smart grid scenario.

7.3 Future Plans

An aim for the future is to submit this project to the 2020 International Verification and Security Workshop. This will allow for the sharing of this project's results with the wider cybersecurity community as well as gaining the perspective of industry leaders as to how this project could be developed on and improved in the future.

One of the key difficulties of learning Scyther for this project was the lack of internet resources available for those new to cryptographic protocol verification. This project has the capability to provide a starting point for others inside or outside the university that wish to verify security policies using Scyther. Through my time on this project, my supervisor has found and I have made contact with a PhD student at the university who is interested in using Scyther as part of their research project. The research and protocols developed in this project will be made available to them. With the longer project schedule and higher level of academic experience the student could take the cryptographic protocol modelling demonstrated in this report to a higher level. The Scyther development environment created for this project will also be made publicly available for others to use.

This project provides a high level overview of the cyber threat landscape faced by smart grids. Further work could focus on a single threat in far greater detail than was possible in this project. For example, a project focusing on session keys alone would allow for an analysis into additional factors such as the specific algorithm used to derive keys and a more detailed analysis of how replay attacks occur.

References

- [1] Keyur K Patel, Sunil M Patel, et al. Internet of things-iot: definition, characteristics, architecture, enabling technologies, application & future challenges. *International journal of engineering science and computing*, 6(5), 2016.
- [2] Xinghuo Yu, Carlo Cecati, Tharam Dillon, and M Godoy Simoes. The new frontier of smart grids. *IEEE Industrial Electronics Magazine*, 5(3):49–63, 2011.
- [3] Boris Kuslitskiy. Smart grid features - ansi blog, Jul 2019.
- [4] Rosslin John Robles and Min-kyu Choi. Assessment of the vulnerabilities of scada, control systems and critical infrastructure systems. *Assessment*, 2(2):27–34, 2009.
- [5] A. Sajid, H. Abbas, and K. Saleem. Cloud-assisted iot-based scada systems security: A review of the state of the art and future challenges. *IEEE Access*, 4:1375–1384, 2016.
- [6] Mercy Bere and Hippolyte Muyingi. Initial investigation of industrial control system (ics) security using artificial immune system (ais). In *2015 International Conference on Emerging Trends in Networks and Computer Communications (ETNCC)*, pages 79–84. IEEE, 2015.
- [7] Nikos Virvilis, Dimitris Gritzalis, and Theodoros K. Apostolopoulos. Trusted computing vs. advanced persistent threats: Can a defender win this game? *2013 IEEE 10th International Conference on Ubiquitous Intelligence and Computing and 2013 IEEE 10th International Conference on Autonomic and Trusted Computing*, pages 396–403, 2013.
- [8] Leyla Bilge and Tudor Dumitraş. Before we knew it: An empirical study of zero-day attacks in the real world. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security, CCS '12*, pages 833–844, New York, NY, USA, 2012. ACM.
- [9] Yongge Wang. sscada: securing scada infrastructure communications. *arXiv preprint arXiv:1207.5434*, 2012.
- [10] Miles A McQueen, Wayne F Boyer, Mark A Flynn, and George A Beitel. Quantitative cyber risk reduction estimation methodology for a small scada control system. In *Proceedings of the 39th Annual Hawaii International Conference on System Sciences (HICSS'06)*, volume 9, pages 226–226. IEEE, 2006.
- [11] Cas JF Cremers. The scyther tool: Verification, falsification, and analysis of security protocols. In *International Conference on Computer Aided Verification*, pages 414–418. Springer, 2008.
- [12] Catherine A Meadows. Formal verification of cryptographic protocols: A survey. In *International Conference on the Theory and Application of Cryptology*, pages 133–150. Springer, 1994.
- [13] Nitish Dalal, Jenny Shah, Khushboo Hisaria, and Devesh Jinwala. A comparative analysis of tools for verification of security protocols. *International Journal of Communications, Network and System Sciences*, 3(10):779, 2010.
- [14] George W Arnold, David A Wollman, Gerald J FitzPatrick, Dean Prochaska, David G Holmberg, David H Su, Allen R Hefner Jr, Nada T Golmie, Tanya L Brewer, Mark Bello, et al. Nist framework and roadmap for smart grid interoperability standards, release 1.0. Technical report, 2010.
- [15] Owasp internet of things project, 2018.
- [16] Carlisle Adams. *Impersonation Attack*, pages 286–286. Springer US, Boston, MA, 2005.
- [17] Yuchen Yang, Longfei Wu, Guisheng Yin, Lijie Li, and Hongbin Zhao. A survey on security and privacy issues in internet-of-things. *IEEE Internet of Things Journal*, 4(5):1250–1258, 2017.
- [18] Tobias A Mattei. Privacy, confidentiality, and security of health care information: Lessons from the recent wannacry cyberattack. *World neurosurgery*, 104:972–974, 2017.
- [19] Chung-Wen Hung and Wen-Ting Hsu. Power consumption and calculation requirement analysis of aes for wsn iot. *Sensors*, 18(6):1675, 2018.

- [20] Philip G. Inglesant and M. Angela Sasse. The true cost of unusable password policies: Password use in the wild. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '10, page 383–392, New York, NY, USA, 2010. Association for Computing Machinery.
- [21] Paul A Grassi, EM Newton, RA Perlner, AR Regenscheid, WE Burr, and JP Richer. Nist 800-63b digital identity guidelines: Authentication and lifecycle management. *McLean, VA, Tech. Rep*, 2017.
- [22] OWASP. Most common password list.
- [23] Murugiah Souppaya and Karen Scarfone. Guide to enterprise patch management technologies. *NIST Special Publication*, 800:40, 2013.
- [24] Catherine A. Meadows. Analyzing the needham-schroeder public key protocol: A comparison of two approaches. In Elisa Bertino, Helmut Kurth, Giancarlo Martella, and Emilio Montolivo, editors, *Computer Security — ESORICS 96*, pages 351–364, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg.
- [25] C.J.F. Cremers. *Scyther : semantics and verification of security protocols*. PhD thesis, Department of Mathematics and Computer Science, 2006.