

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

Contents

1	Project Description	4
1.1	Project Aims and Objectives	5
1.2	Project Scope	5
2	Glossary of terms	5
3	Literature Review	6
3.1	Internet of Things (IoT) Devices	6
3.2	Smart Grids	6
3.3	IoT Smart Grid, the Threats, Attitudes and Best Practices	6
3.4	Verification of Security Policies and Protocols	7
4	Research and Design	8
4.1	Scyther development environment	8
4.1.1	Requirements and development methodology	9
4.1.2	Development plan	11
4.2	Threat Model	12
4.2.1	Weak/Default Password Fuzzing Attack	13
4.2.2	Man In The Middle (MITM) Attack	13
4.2.3	Passive Eavesdropping	14
4.2.4	Replay Attack	14
4.2.5	Impersonation Attack	15
4.2.6	Open Port Scanning	15
5	Recommendation of policies and practices	16
5.1	Communication Policies	16
5.2	Best Practices	17
5.2.1	Password Management	17
5.2.2	Network Segregation	17
5.2.3	Patch Management	17
5.2.4	Minimum Design	17
6	Implementation and specification of non-communication IoT policies	18
6.1	Smart Hub password management	18
7	Implementation and modelling of communication protocol policies	19
7.1	Message Encryption	19
7.1.1	Design	19
7.1.2	Implementation	20
7.1.3	Review	21
7.2	Implicit key authentication	22
7.2.1	Design	22
7.2.2	Implementation	24
7.2.3	Review	25
7.3	Unique Session Keys	26
7.3.1	Design	26
7.3.2	Implementation	26
7.3.3	Review	27
7.4	Mutual Authentication	27
8	Plan for remaining work	27
8.1	Risk Analysis	28
8.2	Gantt Chart	29

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.

The scenario for my project is a IoT based smart grid with a focus on the IoT devices in the system and their interactions with the cloud layer.

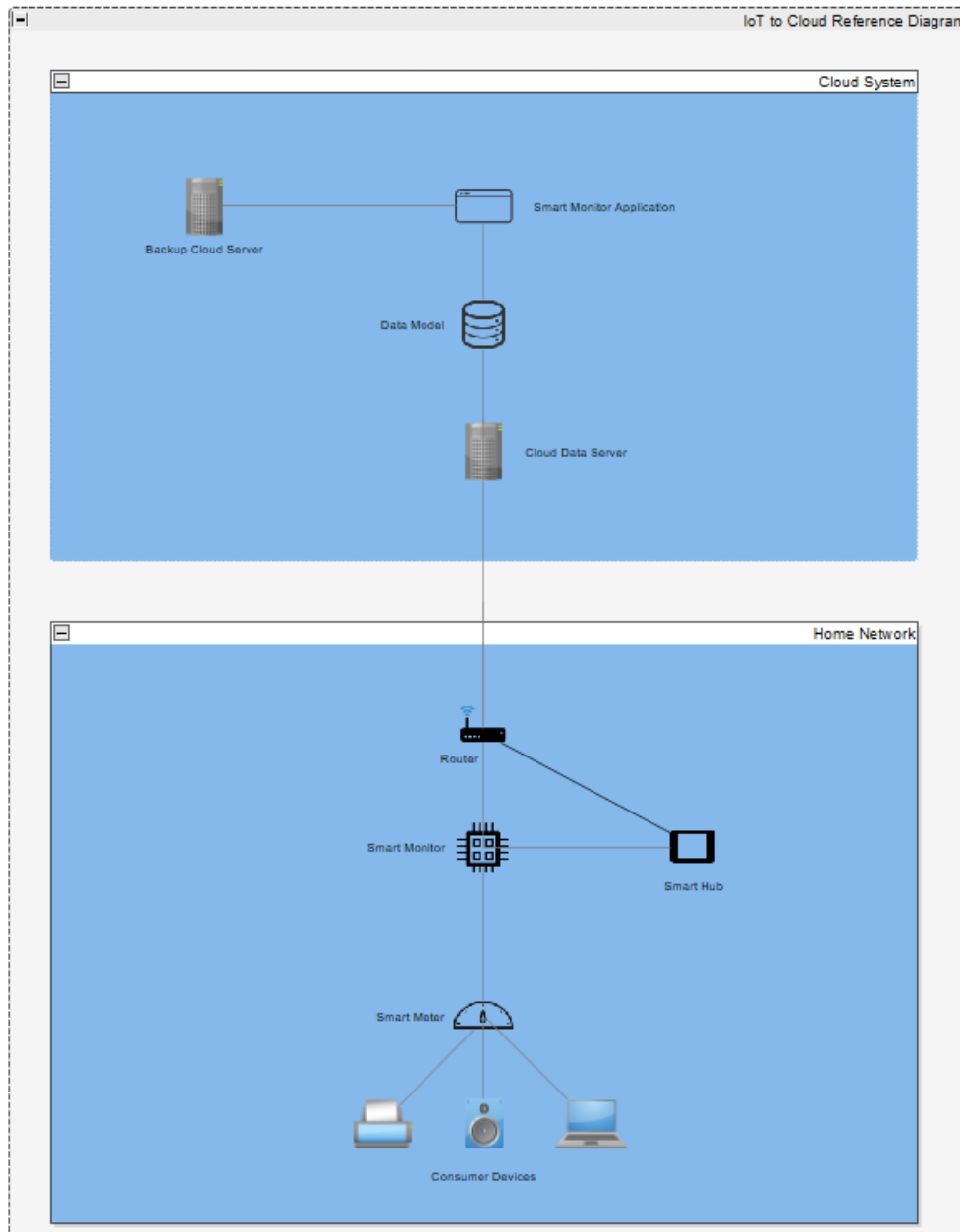


Figure 1: Reference diagram of my smart grid scenario

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 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.

1.2 Project Scope

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 will focus mainly on IoT communication protocols and their configuration rather examine flaws in the hardware or firmware ran by these devices.

2 Glossary of terms

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.

3 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.

3.1 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.

3.2 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.

3.3 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 reduction in risk to management.

3.4 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 several differences that make Scyther more suitable than Pro-Verif for my scenario and skillset.

- **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.

Based on these factors, the project will use Scyther for the modelling and verification of protocols.

4 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 of 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.

4.1 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 ideally being compatible with 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.

4.1.1 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:

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	

Table 1: Table showing the prioritisation of the scytherbox requirements

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.

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

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

Key:

Small - 1 Hour

Medium - 2 Hours

Large - 4 Hours

Extra Large - 10 Hours

4.1.2 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 sprit.

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: 8	Sprint Hours: 9	Sprint Hours: 6

Table 3: Scytherbox sprint plan

4.2 Threat Model

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

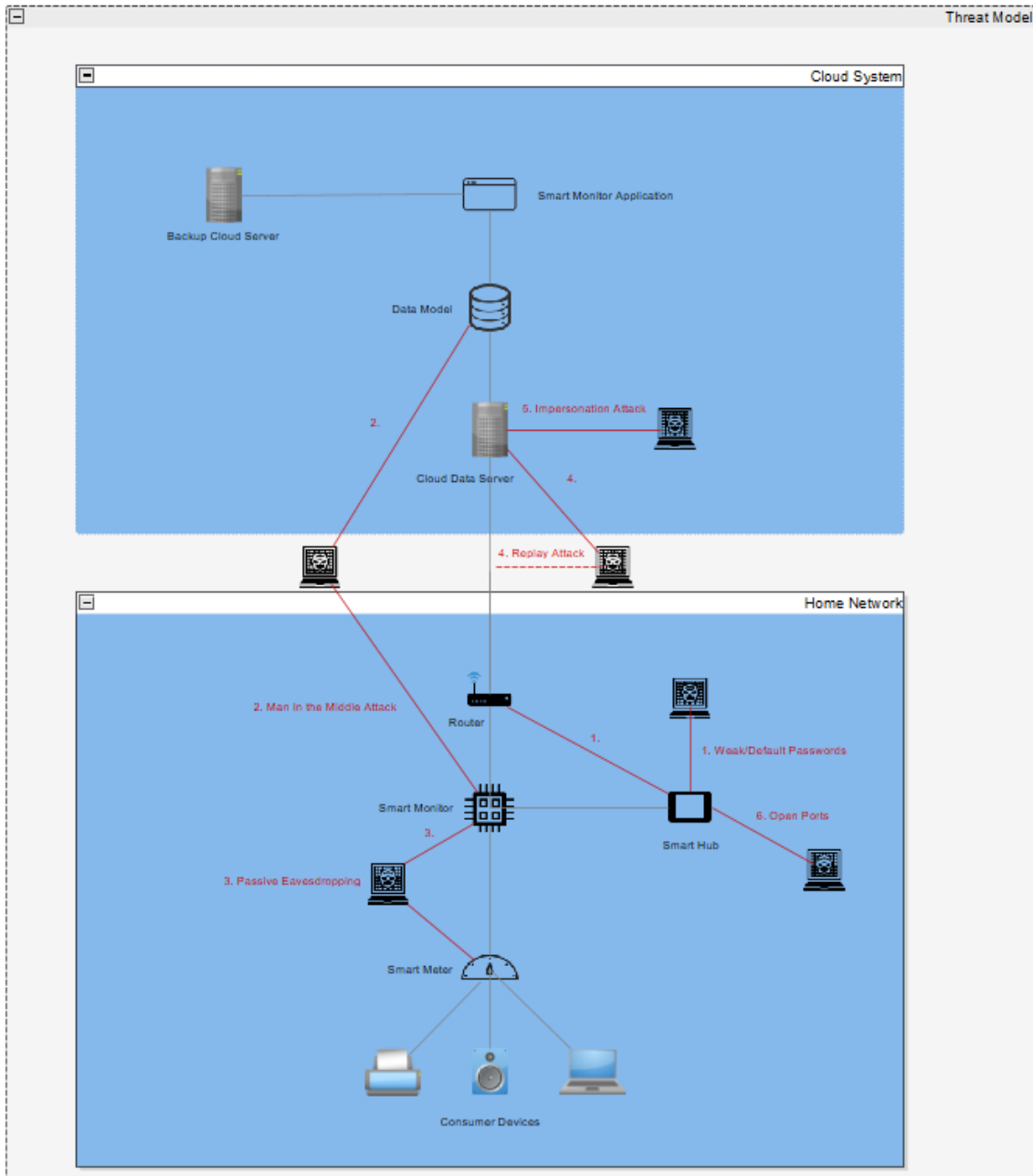


Figure 2: Threat Model of my smart grid scenario

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.

4.2.1 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.

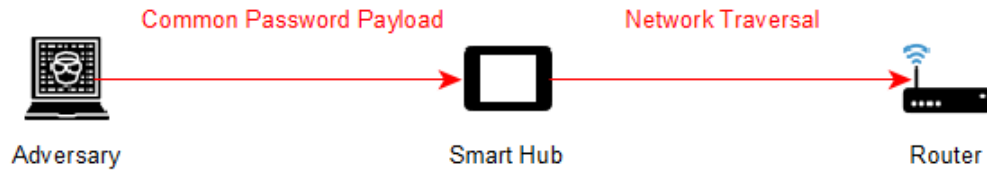


Figure 3: Adversary using a common password to compromise the network

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.

4.2.2 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 the contents of messages.

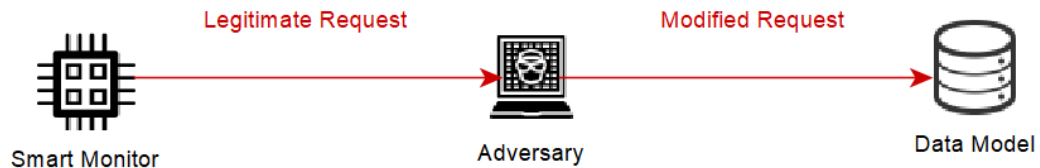


Figure 4: Adversary relaying and modifying smart monitor data

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 false data injection attack on the smart grid system where false data could cause the system to make an incorrect decision when routing power.

4.2.3 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.

4.2.4 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.

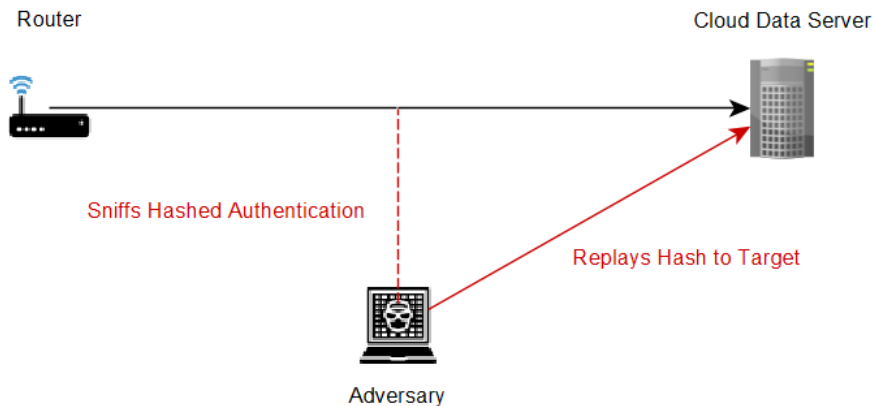


Figure 6: Adversary sniffing and reusing hashed authentication credentials

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.

4.2.5 Impersonation Attack

An Impersonation attack occurs when an Adversary is able to pose as the identify 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.

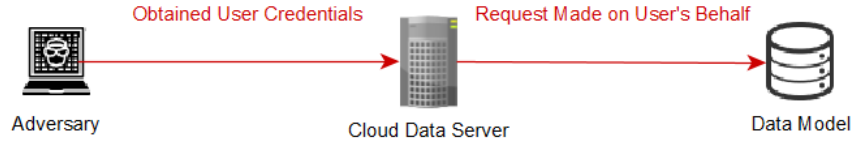


Figure 7: Adversary posing as a legitimate smart meter

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.

4.2.6 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 a insecure port that has not been blocked as an attack vector. Botnet recruitment malwares such as Mirai scan these ports to identify IoT devices that can be compromised. [17]

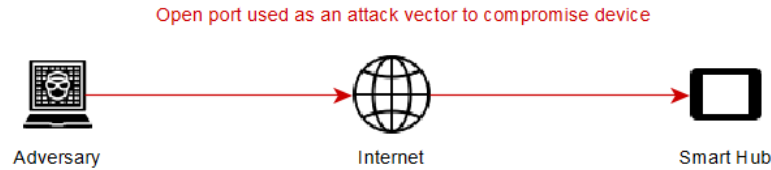


Figure 8: Adversary using an open port to attack a device

This attack can occur in the scenario where any devices are configured to allow network traffic in from unnecessary communication protocols such as telnet (port 23) and SSH (port 22).

5 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 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.

5.1 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 [18] 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.

5.2 Best Practices

Policies defined in this section are designed to mitigate non-communication attacks...

5.2.1 Password Management

5.2.2 Network Segregation

5.2.3 Patch Management

5.2.4 Minimum Design

6 Implementation and specification of non-communication IoT policies

6.1 Smart Hub password management

In a study examining user behaviour toward password policies, Inglesant [19] 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 list 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:

- Passwords must be at least 8 characters long
- Before hashing, passwords should be checked against OWASP's top 10,000 password list [20]
- Created passwords must only be used for the Smart Hub

7 Implementation and modelling of communication protocol policies

7.1 Message Encryption

The first policy to be implemented is message encryption. As this policy is designed to mitigate the threat of passive eavesdropping, I am defining the implementation successful if an adversary is unable to decrypt and read either the key or freshly generated message using a passive Eavesdropping attack.

7.1.1 Design

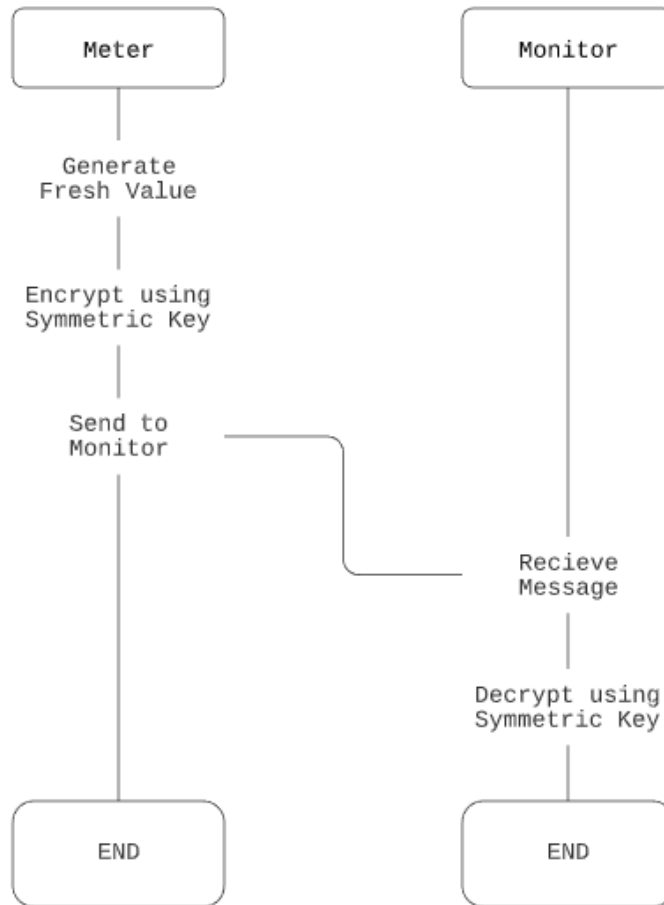


Figure 9: Message encryption protocol design

7.1.2 Implementation

The implementation of the design in Scyther is a two-way communication where the Meter sends information to the Monitor and the Monitor sends back a confirmation of having received the message.

```
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 10: Message encryption protocol design

7.1.3 Review

To model the requirements of both the freshly generated value and the key not being disclosed, Scyther's Secret claim was made on the message which models an adversary attempting to eavesdrop on the message during communication.

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

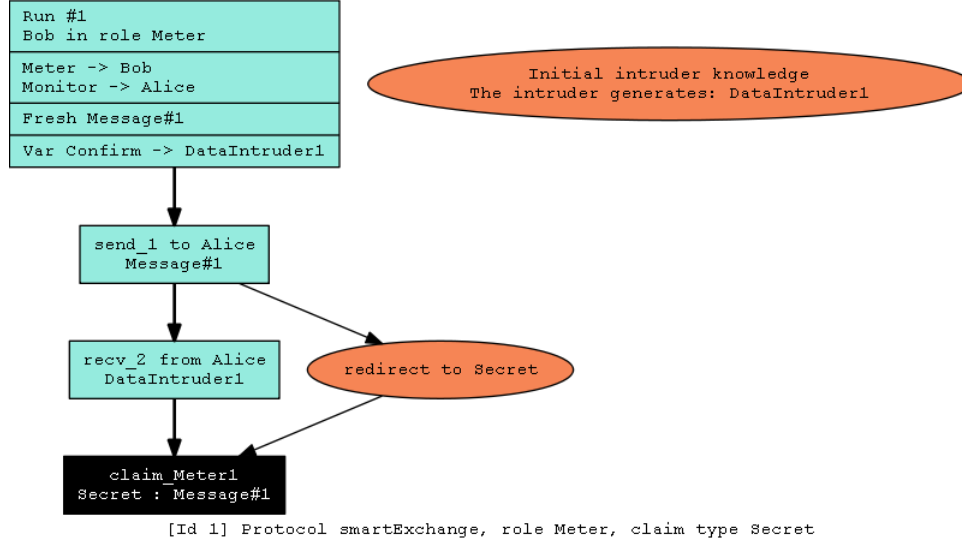


Figure 11: Message encryption protocol test results

The figure above shows by eavesdropping the message, 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 passed these 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

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

Table 4: Log of claims made against the message encryption protocol and their results

7.2 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.

7.2.1 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.

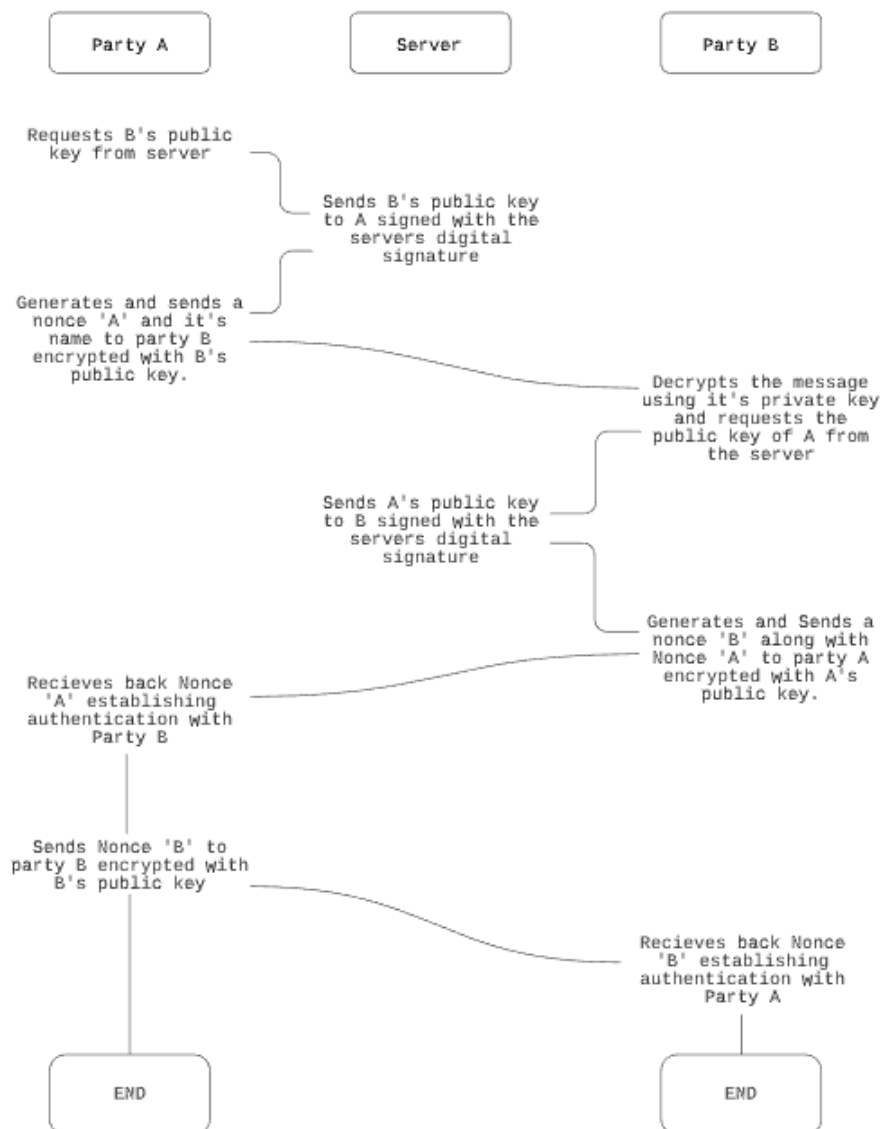


Figure 12: Illustration of the Needham Schroeder authentication process [21]

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 a 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.

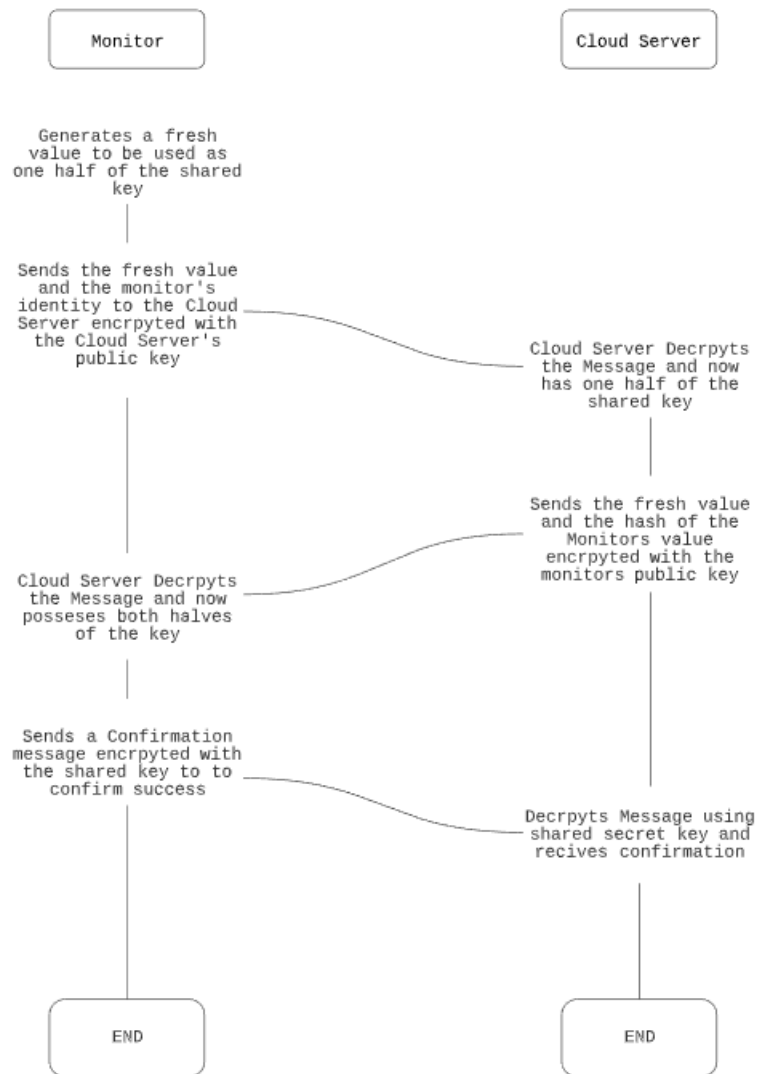


Figure 13: Illustration of the planned key exchange [21]

Once each party exchanges it's 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.

7.2.2 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 }

```

7.2.3 Review

7.3 Unique Session Keys

Implementing session keys

7.3.1 Design

7.3.2 Implementation

The Scyther implementation uses the `SessionKey` usertype to model the session keys. Once both parties have sent each other an encrypted communication containing the session key, the Meter sends the Monitor the message.

```
0
1 usertype SessionKey;
2 usertype Message;
3
4 protocol EncryptedExchange(Meter,Monitor)
5
6 {
7
8   role Meter {
9
10    fresh M: Message;
11    fresh TokenA: SessionKey;
12    var TokenB;
13
14    send_1(Meter,Monitor,{TokenA}k(k));
15    recv_2(Monitor,Meter,{TokenB}k(k));
16    claim(Meter,Running,Monitor,M);
17    send_3(Meter,Monitor,{M}k(k));
18
19    claim_Meter1(Meter, Secret, (k));
20    claim_Meter2(Meter, Secret, M);
21    claim_Meter3(Meter,Niagree);
22    claim_Meter4(Meter,Nisynch);
23
24  }
25
26  role Monitor {
27
28    var M;
29    var TokenA;
30    fresh TokenB: SessionKey;
31
32    recv_1(Meter,Monitor,{TokenA}k(k));
33    send_2(Monitor,Meter,{TokenB}k(k));
34    recv_3(Meter,Monitor,{M}k(k));
35
36    claim_Monitor1(Monitor, Secret, (k));
37    claim_Monitor2(Monitor, Secret, M);
38    claim_Meter3(Monitor,Niagree);
39    claim_Meter4(Monitor,Nisynch);
40
41  }
42 }
```

7.3.3 Review

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. [22]

As shown in the figure below, without the implementation of session keys it is trivial for an adversary to pose as a meter which would allow them to send false readings in a message or potentially use the message as an attack vector for malware.

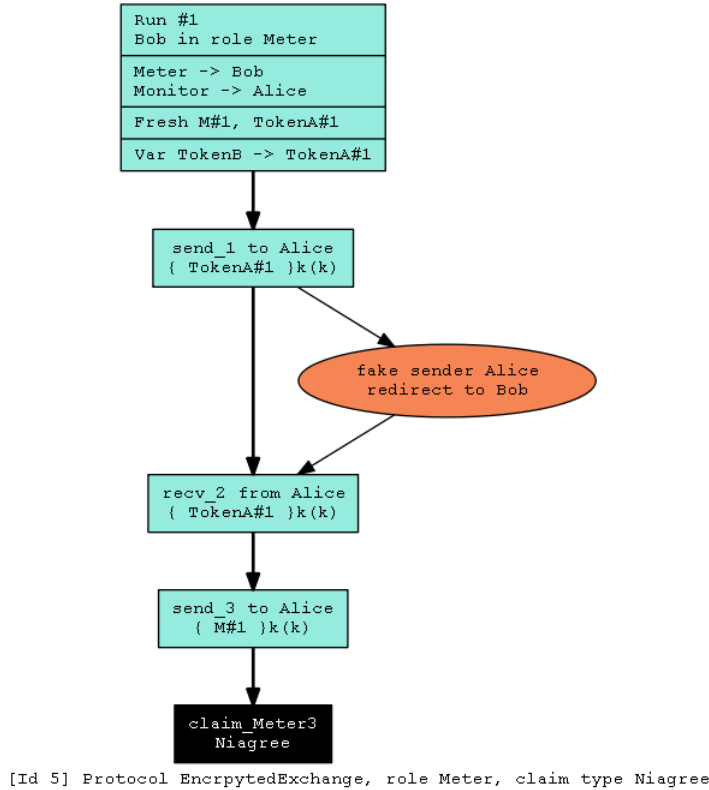


Figure 14: Niagree verification results before session key implementation

The results after the implementation show that whilst the simple impersonation attack

7.4 Mutual Authentication

8 Plan for remaining work

When referring back to NIST's [14] guidelines for the analysis and security implementation of a smart grid system, the first 3 phases defining use cases, risk assessment and specification of security requirements are reviewed in this report. Whilst the specification of security requirements will be further developed, the main focus of my remaining work is on the design and development of a security architecture and the assessment of the implementation of this architecture. This is reflected in my Gantt chart (fig:16) which shows how I plan to break down this work into tasks and my expected timings for each of these tasks.

8.1 Risk Analysis

I have identified 5 risks as key risks which could impact on my delivery of the rest of the work. The grid below shows my plan to mitigate these risks and my assessment of any residual impact that may linger.

Risk	Baseline	Mitigation	Residual
Scyther stops being supported on modern operating systems 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	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
My remaining work is larger or more difficult than I anticipated meaning I fail to complete parts of it	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.	Impact: 3 Likelihood: 2 Score: 6
Personal/family issue	Impact: 3 Likelihood: 3 Score: 9	Use the university support service when needed. Keep my supervisor informed	Impact: 2 Likelihood: 3 Score: 6

Table 5: Qualitative risk analysis and mitigation plan for the key risks

8.2 Gantt Chart

My Gantt chart details my time management plan for the progress report and future plan for the rest of the project.

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 Dumitras. 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] Chung-Wen Hung and Wen-Ting Hsu. Power consumption and calculation requirement analysis of aes for wsn iot. *Sensors*, 18(6):1675, 2018.

- [19] 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.
- [20] OWASP. Most common password list.
- [21] 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.
- [22] C.J.F. Cremers. *Scyther : semantics and verification of security protocols*. PhD thesis, Department of Mathematics and Computer Science, 2006.

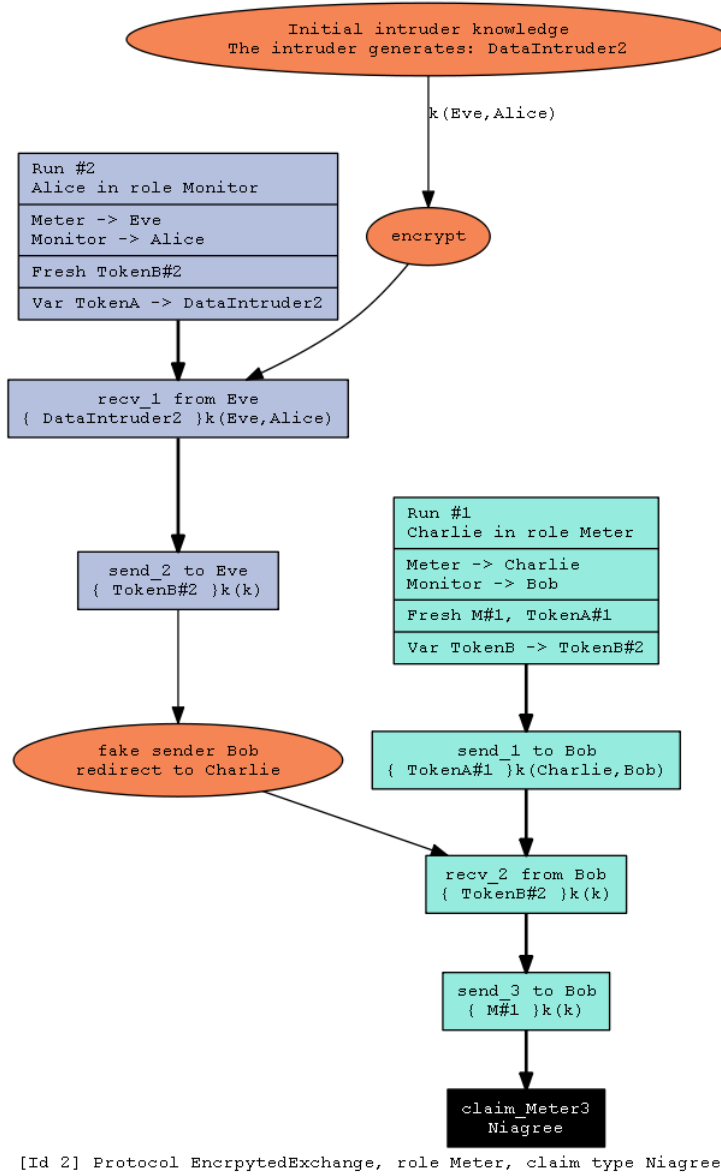


Figure 15: Niagree verification results after session key implementation

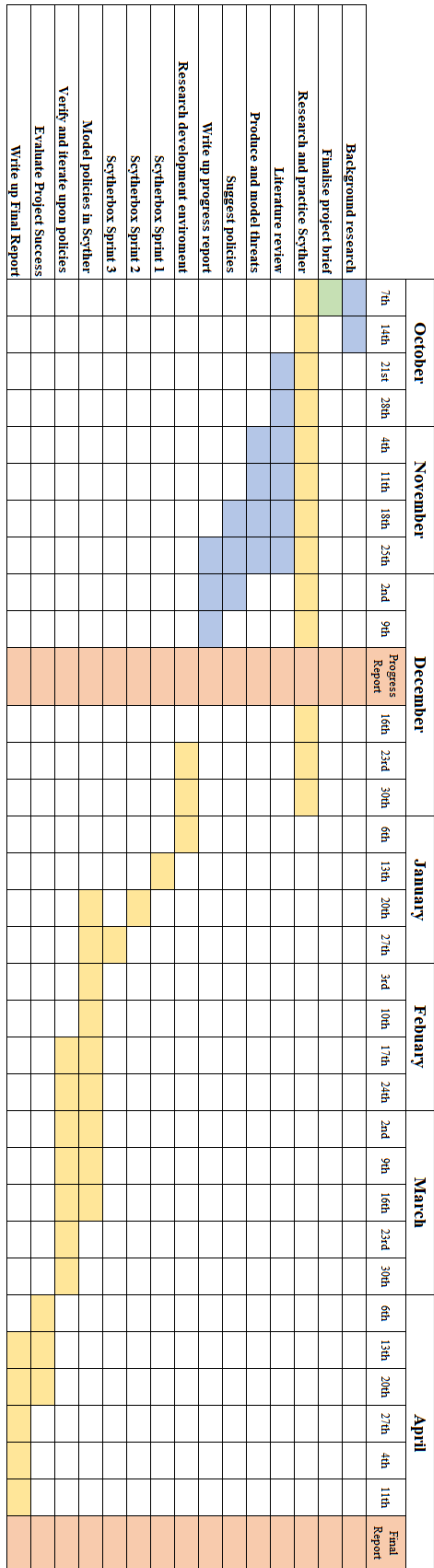


Figure 16: Gantt chart for the project