Analysing MIKEY-SAKKE:
A Cryptographic Protocol for Secure Multimedia Services

by

Chloe Bell

Submitted in partial fulfilment of the requirements for the MSc Degree in Computing Science of Imperial College London

September 2015
Abstract

MIKEY-SAKKE is a cryptographic protocol [54] used for the “provision of secure, cross-platform multimedia communications” [28], recommended by CESG (The Communications-Electronics Security Group) for use by government and enterprise organisations. The protocol uses the MIKEY key management scheme [9]; the SAKKE key exchange protocol [55]; and the ECCSI authentication protocol [53]. Even though there is security analysis of the different components there is currently no open security analysis of the complete MIKEY-SAKKE protocol.

In this project I aim to identify an approach to security analysis that is open, accurate and dynamic. Analysis by a wide range of security experts with a variety of skills enables more effective security analysis of any protocol as even if one person cannot break the protocol someone else may be able to. This thesis is intended as a starting point for the security analysis of MIKEY-SAKKE and for it to be reviewed and contributed to in order to improve the accuracy and completeness of the analysis which ultimately will either substantiate or contradict the level of security claimed. Quantitative analysis techniques are required in order to provide accuracy in any results; and ongoing, dynamic analysis is needed to capture changes in the capabilities of technology and advances in attack methods.

I derive the approach of Continuous Iterative Improvement to security analysis, incorporating both high level security analysis and low level exploit investigations, to identify improvements to the protocol design; implement bug fixes; and deploy defensive preventative measures.

The high level security analysis is conducted using the ADVISE formalism [72] and implemented with the Möbius tool [44, 35]. For this I incorporated a specification of the MIKEY-SAKKE system; descriptions of a range of potential adversaries; and the derivation of relevant metrics, which are produced as a result of the simulated attacks by each of the adversaries on the system.

For the low level exploit investigations I identified the possibility of a denial of service attack against the MIKEY protocol; I conducted a thought experiment for a transfer attack on the supersingular curve used by the SAKKE protocol; calculated the impact of a predictable pseudo-random number generator used with the ECCSI protocol; and researched the potential for timing attacks on AES, which is used as the symmetric encryption protocol after key exchange.

From my analysis, I was able to establish three potential attacks against MIKEY-SAKKE as characterised by the ADVISE model. However, I was also able to identify a number of further potential weakness, therefore future work would be able to establish a more complete security analysis of the MIKEY-SAKKE protocol.

The Continuous Iterative Improvement approach to the security analysis proved to be particularly effective and in theory is applicable to many other cryptosystems. Further implementation of this approach on different protocols will provide insight into the effectiveness of the approach as a general tool for open, accurate and dynamic security analysis. Multimedia Internet KEYing

1https://www.cesg.gov.uk/Pages/homepage.aspx
I would like to thank my supervisor, Dr. Jeremy Bradley, for his advice and guidance throughout this project; especially with direction for different security modelling techniques and support when my implementation attempt at an elliptic curve attack failed.

Thank you to Andy Lilley and Andy Kinney at Armour Communications for proposing this project; providing insight into the security industry; and reviewing my many draft report sections.

Finally, thank you to Dan Middlecote for his mathematical skills and dedicated help in my final (but unfortunately still unsuccessful) attempt at implementing that elliptic curve attack.
# Contents

List of Figures ................................................. 6

List of Tables .................................................. 7

Acronyms ............................................................ 8

1 Introduction .................................................... 10
   1.1 Motivation .................................................. 10
   1.2 Objectives .................................................. 11
   1.3 Contribution and Achievements .......................... 11
   1.4 Report Outline .............................................. 11

2 Background ..................................................... 13
   2.1 MIKEY-SAKKE Cryptosystem ............................... 13
       2.1.1 Overview ............................................. 13
       2.1.2 Trust Models ......................................... 14
       2.1.3 Attack Definitions ................................... 15
       2.1.4 MIKEY ............................................... 15
       2.1.5 SAKKE ............................................... 17
       2.1.6 ECCSI ............................................... 19
       2.1.7 AES .................................................. 21
       2.1.8 AES Key Wrap ....................................... 23
   2.2 Analysis Techniques ......................................... 25
       2.2.1 Attack Trees ......................................... 25
       2.2.2 Attack-Defense Trees ................................. 26
       2.2.3 ADVISE and Möbius ................................. 26
       2.2.4 Security Assurance Cases ............................ 28
   2.3 Vulnerabilities in Other Protocols ........................ 29
       2.3.1 General ............................................... 29
       2.3.2 SSL and TLS ............................................ 29
       2.3.3 ECDSA ............................................... 31

3 High level Security Analysis ................................ 32
   3.1 Theory ...................................................... 32
       3.1.1 Goals ................................................. 32
       3.1.2 Attacks ............................................... 32
       3.1.3 Adversaries .......................................... 33
   3.2 Implementation ............................................. 35
       3.2.1 Möbius Model Version 1 .............................. 35
       3.2.2 Möbius Model Version 5 .............................. 38
       3.2.3 Möbius Model Version 7 .............................. 39
       3.2.4 Möbius Model Version 8 .............................. 40
   3.3 Results ...................................................... 40
       3.3.1 Möbius Model Version 1 .............................. 40
### 3.3.2 Möbius Model Version 5 ........................................ 41
### 3.3.3 Möbius Model Version 7 ........................................ 43
### 3.3.4 Möbius Model Version 8 ........................................ 43
### 3.4 Evaluation ....................................................... 45

### 4 Low level Exploit Investigation .................................. 48
#### 4.1 Theory .......................................................... 48
##### 4.1.1 MIKEY ....................................................... 48
##### 4.1.2 SAKKE ....................................................... 48
##### 4.1.3 ECCSI ......................................................... 51
##### 4.1.4 AES .......................................................... 54
#### 4.2 Implementation .................................................. 54
#### 4.3 Evaluation ....................................................... 57

### 5 Evaluation Summary and Conclusion ............................. 59
#### 5.1 Overall Evaluation .............................................. 59
#### 5.2 Conclusion ....................................................... 60
#### 5.3 Future Work ..................................................... 61

Bibliography .......................................................... 62

Appendices ............................................................ 70

A Key Management Server Set Up .................................... 71
B Client Set Up .......................................................... 73
C Intel Threat Agent Library ........................................... 75
D CAPEC ................................................................. 76
E Elliptic Curves .......................................................... 79
F Rho Method Cost Calculation [90] ................................. 80
G MOV Attack - Find Elliptic Curve Point .......................... 82
H MOV Attack - Discrete Logarithm Problem [101] .................. 84
I Safe Curves ............................................................. 86
# List of Figures

2.1 Architecture ......................................................... 13  
2.2 I_MESSAGE Structure ............................................... 16  
2.3 Continuous Iterative Improvement ................................. 25  
2.4 Example Attack Tree [105] ........................................ 25  
2.5 Example Attack-Defense Tree [70] ............................... 26  
2.6 Example Attack Execution Graph [72] ........................... 27  
3.1 Read Message Möbius Model Version 1 ........................... 35  
3.2 KMS Denial of Service Möbius Model Version 1 ............... 36  
3.3 Impersonate User Möbius Model Version 1 ..................... 36  
3.4 Tamper with I_MESSAGE Möbius Model Version 1 ........... 36  
3.5 User Denial of Service Möbius Model Version 5 ............... 38  
3.6 Create Malicious User Möbius Model Version 5 ............... 38  
3.7 Read All Messages Möbius Model Version 7 .................... 39  
3.8 User Denial of Service Möbius Model Version 8 ............... 40  
3.9 Impersonate User Möbius Model Version 8 ..................... 40  
4.1 Example Section of Key Provision Response from KMS [113] 55  
5.1 Continuous Iterative Improvement for MIKEY-SAKKE ........ 59  
C.1 Intel Threat Agent Library Section 1 [25] ....................... 75  
C.2 Intel Threat Agent Library Section 2 [25] ....................... 75  
E.1 $E : y^2 = x^3 - x + 1$ [117] ........................................ 79  
E.2 $E : y^2 = x^3 - x + 1 \mod 97$ [117] ............................... 79
List of Tables

2.1 SAKKE Notation ......................................................... 18
2.2 ECCSI Notation .......................................................... 20
2.3 AES Notation ............................................................... 22
2.4 AES Key Wrap Notation .................................................. 23
3.1 STRIDE ........................................................................ 32
3.2 Intel Threat Agent Library [25] ........................................ 33
3.3 Möbius and Intel Threat Adversary Specifications .................. 34
3.4 Adversary Specifications for Möbius Model Version 1 .............. 37
3.5 Results from Möbius Model Version 1 .................................. 41
3.6 Results from Möbius Model Version 5 (1) ............................ 42
3.7 Results from Möbius Model Version 5 (2) ............................ 42
3.8 Results from Möbius Model Version 7 ................................ 43
3.9 Results from Möbius Model Version 8 (1) ............................ 44
3.10 Results from Möbius Model Version 8 (2) .......................... 44
3.11 Results from Möbius Model Version 8 (3) .......................... 47
4.1 Elliptic Curve Notation ................................................... 49
4.2 Elliptic Curve Equations .................................................. 49
4.3 SafeCurves Criteria [12] .................................................... 50
4.4 ECCSI Notation ............................................................. 52
4.5 NIST P-256 Analysis by SafeCurves [12] ............................. 52
4.6 MOV Attack Notation ...................................................... 55
5.1 Möbius Model Final Results .............................................. 59
I.1 Safe Curves [12] ............................................................ 86
Acronyms

**ADVISE**  ADversary V1ew Security Evaluation.

**AES**  Advanced Encrypted Standard.

**CAPEC**  Common Attack Pattern Enumeration and Classification.

**CESG**  The Communications-Electronics Security Group.

**DES**  Data Encryption Standard.

**DH**  Diffie-Hellman.

**DLP**  Discrete Logarithm Problem.

**DoS**  Denial of Service.

**DSA**  Digital Signature Algorithm.

**ECC**  Elliptic Curve Cryptography.

**ECCSI**  Elliptic Curve base Certificateless Signatures for ID-based Encryption.

**ECDLP**  Elliptic Curve Discrete Logarithm Problem.

**ECDSA**  Elliptic Curve Digital Signature Algorithm.

**GCHQ**  Government Communications Headquarters.

**HTTP**  HyperText Transfer Protocol.

**HTTPS**  HyperText Transfer Protocol Secure.

**ISP**  Internet Service Provider.

**ITAL**  Intel Threat Agent Library.

**KMS**  Key Management Server.

**MIKEY**  Multimedia Internet KEYing.

**MOV**  Menezes, Okamoto and Vanstone.

**NIST**  National Institute of Standards and Technology.

**NSA**  National Security Agency.

**PGP**  Pretty Good Privacy.

**PRNG**  Pseudo-Random Number Generator.
**RAM** Random Access Memory.

**RFC** Request For Comments.

**RSA** Rivest, Shamir, Adleman.

**SAKKE** Sakai-Kasahara Key Encryption.

**SHA-256** Secure Hash Algorithm 256 bit.

**SRTP** Secure Real-time Transport Protocol.

**SSL** Secure Sockets Layer.

**STRIDE** Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, and Elevation of Privilege.

**TCP** Transmission Control Protocol.

**TLS** Transport Layer Security.

**UDP** User Datagram Protocol.

**URI** Uniform Resource Identifier.

**URL** Uniform Resource Locator.

**Wi-Fi** Wireless Internet for Frequent Interface.
Chapter 1

Introduction

1.1 Motivation

In 2013 Edward Snowden leaked confidential NSA, National Security Agency,\(^1\) documents to The Guardian\(^2\) newspaper detailing the extent of the US and UK governments’ surveillance programs [118]. From these revelations there have been raised suspicions about the actual technical capabilities of the NSA and GCHQ, Government Communications Headquarters\(^3\): from having back doors in software that claims to be secure [40, 107]; to being able to break widely used encryption protocols [20, 108]. All these revelations have provoked a lack of trust in the security claims of cryptographic protocols, especially those designed by government agencies.

In order to overcome this lack of trust I believe we require open and effective security analysis in order to prove or disprove any claim made on the level of security provided by any cryptographic protocol. This substantiation eliminates any need for users to trust the protocol designers and instead bases the validity of the claimed level of security on wide ranging investigations undertaken by experts in the field.

In any real system there is no such thing as perfect security, as even if the specific protocol itself is provably secure [30], there are external interactions with people using the protocol; there can be flaws introduced by the implementation of the theoretical protocol [1]; and there may be vulnerabilities in the hardware or operating system the protocol is running on [39].

As such a binary approach to describing security is insufficient; we require clear quantitative measures capturing the level of security provided and the different areas of strength and weakness in a system. Cryptographic protocols that may be highly secure today could be broken tomorrow as new techniques are developed and the capabilities of technology advance. Therefore not only do we require clear quantitative measures but they must also be dynamic and able change over time as new information is acquired. For the security analysis of any cryptosystem to be most effective it must be open, accurate and dynamic.

The Communications-Electronics Security Group, also known as CESG,\(^4\) is the National Technical Authority for Information Assurance within the UK. This means they provide advice and define standards for UK information security; and are often referred to as the “information security arm of GCHQ” [26]. MIKEY-SAKKE is a cryptographic protocol for the “provision of secure, cross-platform multimedia communications” [28], recommended by CESG for use by government and enterprise organisations as a highly available, flexible and scalable protocol [28].

The Secure Chorus group [113], a collaboration of enterprise organisations, are working with CESG on evolving and prompting the MIKEY-SAKKE protocol theory and implementation. Armour Communications,\(^5\) a member of the Secure Chorus group, approached Imperial College London in order to conduct a thorough analysis of the security provided by MIKEY-SAKKE.

---

\(^1\)http://www.nsa.gov/
\(^2\)http://www.theguardian.com/uk
\(^3\)http://www.gchq.gov.uk/Pages/homepage.aspx
\(^4\)https://www.cesg.gov.uk/Pages/homepage.aspx
\(^5\)http://www.armourcomms.com/
1.2 Objectives

In this project I aim to fulfil the request of Armour Communications by initiating an open, accurate and dynamic security analysis of the MIKEY-SAKKE cryptographic protocol. The specific objectives are:

- Design an effective approach for open, accurate and dynamic security analysis of any cryptographic protocol
- Identify specific analysis techniques that can be used as part of the overall security analysis
- Conduct a security analysis of the MIKEY-SAKKE cryptographic protocol

1.3 Contribution and Achievements

From research of a range of analysis techniques, I was able to derive the Continuous Iterative Improvement approach to security analysis.

Continuous iterative improvement for security analysis prescribes the repeated application of quantitative analysis techniques to the cryptographic protocol, each time incorporating new information often gained from the previous iteration. Each new iteration aims to improve the accuracy of the security analysis as well as the completeness of the system and adversary descriptions. The analysis is centred around a high level security analysis, for which I use the ADVISE formalism, and low level exploit investigations each providing new information for the other for the next iteration.

The use of accurate analysis techniques for the high level and low level analyses ensures the accuracy of the overall security analysis and the iterations lend themselves to the dynamic requirement enabling new information to be incorporated as attack techniques or technologies advance.

I implement this Continuous Iterative Improvement security analysis on the MIKEY-SAKKE protocol using the M"obius tool \[44, 35\] with the ADVISE formalism \[72\] for the high level security analysis of the system and a range of potential adversaries. I also conducted some low level exploit investigations into specific areas of weakness of the protocol from which I was able to feed the results back into the M"obius model to improve the accuracy and completeness of my analysis.

For one of the low level exploits I implemented a thought experiment into the feasibility of the MOV attack on the SAKKE supersingular elliptic curve. This was proven to be infeasible, even though supersingular curves are known to be susceptible to this attack, due to the large parameter sizes used by the SAKKE protocol.

I was also able to identify a vulnerability in the ECCSI protocol in which if the pseudo-random number generator used is predictable then the user’s secret signing key can be derived from two signatures.

1.4 Report Outline

**Chapter 2: Background** describes the background research associated with this project including detail of the MIKEY-SAKKE protocol; discussion of different high level security analysis techniques; and examples of vulnerabilities in other protocols as a base for the identification of potential weaknesses in MIKEY-SAKKE.

**Chapter 3: High level Security Analysis** covers the theory, implementation and results of the high level security analysis using the ADVISE formalism with the M"obius tool; and ends with a detailed evaluation of the high level security analysis.

**Chapter 4: Low level Exploit Investigation** describes investigations conducted on the different specific areas of the MIKEY-SAKKE protocol including the MOV attack on the SAKKE supersingular curve and the impact of a predictable pseudo-random number generator on the ECCSI
protocol. A detailed evaluation of all findings from the investigations is given at the end of this chapter.

Chapter 5: Evaluation Summary and Conclusion provides an overall evaluation of the Continuous Iterative Improvement approach to security analysis as well as specific recommendations for the MIKEY-SAKKE protocol. A conclusion of the project describing the achievements and limitations is covered as well as suggestions for future work.
Chapter 2

Background

2.1 MIKEY-SAKKE Cryptosystem

MIKEY-SAKKE is a cryptographic protocol proposed by The Communications-Electronics Security Group (CESG\textsuperscript{1}), a group within the UK Government Communications Headquarters (GCHQ\textsuperscript{2}). It is intended for use across government and enterprise to enable “secure, cross-platform multimedia communications”.\textsuperscript{[54, 28, 27]}

MIKEY-SAKKE provides the key exchange element of a secure real-time protocol profile for transmitting encrypted media, such as voice over internet protocol \textsuperscript{[27]}. It aims to provide encryption, authentication, integrity and replay protection in both unicast and multicast applications. It is highly scalable; flexible; and maintains low latency for real time communications.

2.1.1 Overview

The overall architecture required for MIKEY-SAKKE is shown in figure 2.1. Each user is associated with a domain that is managed by a key management server. The three different elements required for secure communication are:

- Key delivery - Secure transfer of the user specific key information from the key management server to the user represented by a dashed line in figure 2.1.

- Key exchange - Secure establishment of a shared secret value between two (or more) authenticated users represented by a solid line in figure 2.1.

- Secure communication - Secure communication between two (or more) authenticated users once a shared secret value has been established. Represented by a dotted line in figure 2.1.

MIKEY-SAKKE provides the key exchange element of the secure communications and the key delivery and secure communication elements are not defined by the MIKEY-SAKKE RFC \textsuperscript{[54]} but left to the decision of the developers implementing the protocol.

CESG have developed and released a reference implementation of MIKEY-SAKKE as part of the Secure Chorus group \textsuperscript{[113]}. In the Secure Chorus implementation the key delivery is implemented using HTTP requests from the user to the key management server \textsuperscript{[112]}. These contain XML defining the required user information and returned key information. The keys are encrypted using the key wrap AES protocol, described in further detail in section 2.1.8 \textsuperscript{[112, 110]}. Secure communication is provided by AES encryption using the shared secret value as a generation key, described further in section 2.1.7 \textsuperscript{[111, 110]}.

The reference implementation does not use the SRTP protocol but instead uses UDP \textsuperscript{[113]}. This means that with this code stack MIKEY-SAKKE over SRTP cannot be analysed. However, with that caveat in mind, the Secure Chorus reference implementation can still be reviewed in detail in order to obtain a thorough analysis of the MIKEY-SAKKE protocol.

\textsuperscript{1}https://www.cesg.gov.uk/Pages/homepage.aspx
\textsuperscript{2}http://www.gchq.gov.uk/Pages/homepage.aspx
MIKEY-SAKKE uses unambiguously defined identities to provide authentication within a domain; and only requires the key management server root certificates for cross-domain communication. The private keys for a user are provisioned periodically by the key management server and in doing so the key management server provides the root-of-trust for domain users.

There are three main components to the MIKEY-SAKKE protocol:

- Multimedia Internet KEYing (MIKEY) [9], see section 2.1.4.
- Sakai-Kasahara Key Encryption (SAKKE) [55, 99], see section 2.1.5.
- Elliptic Curve based Certificateless Signatures for ID-based Encryption (ECCSI) [53], see section 2.1.6.

These are used together such that the MIKEY protocol allows the initiator of the communication to send a single simplex transmission to the intended recipient in the form of a MIKEY I MESSAGE containing SAKKE encapsulated data and a ECCSI signature. The recipient validates the ECCSI signature and processes the SAKKE encapsulated data to derive a shared secret value [54].

2.1.2 Trust Models

To authenticate something is “to prove that something is real, true or genuine : to prove that something is authentic” [120]. This is a key feature of many cryptosystems including MIKEY-SAKKE such that users are able to authenticate the other users with whom they are communicating. There are many different models for doing this each of which are differently suited to the different organisations that use them.

The Chain-of-Trust model is widely used, as it is a feature of the Transport Layer Security (TLS) protocol and its predecessor the Secure Sockets Layer (SSL) that provide security for HTTPS communications. It uses certification to provide authentication such that each user requires a certificate that is signed by another user higher up the chain. At the base of each chain is a certification authority, therefore a user only trusts another if they have a valid certificate that can be traced back up to a valid certification authority.

One weakness associated with this model is the certification authority. If a certification authority is hacked and even only a few of its certificates stolen this jeopardises the guarantee that all certificates from this authority are genuine and therefore breaks the chain-of-trust for all certificates associated with that authority. This makes the certification authority a single point of failure for the chain-of-trust model and one with a significant reward to an attacker.
Alternatively there is the **Web-of-Trust** model used by the Pretty Good Privacy (PGP\(^3\)) protocol. Like the *chain-of-trust* model this relies on certificates, however, each user’s certificate is signed by another user as opposed to an overarching authority. Therefore each user only trusts other users if they have a valid certificate signed by a peer that is already trusted.

This does remove the single point of failure as was the case with the *chain-of-trust* model, however, the structure of the model is not well suited to that of the average organisation, for example, a private company or a government department.

MIKEY-SAKKE utilises the model of a **Root-of-Trust**. The architecture described in figure 2.1 enables the key management server to act as a *root-of-trust* for the domain such that only the key management server needs to be certificated and not all of the users. This means that a user in a domain can trust any other user in that same domain without the need for certification. Certification is only required for inter-domain communications where the user requires a valid certificate for the key management server of the other domain.

The key management server certificates are provisioned using a *chain-of-trust* model which again introduces the single point of failure in the certification authority but the benefits of this model are well suited to many organisational structures for example allowing for authenticated communications within a domain or organisation without the need for certificates.

### 2.1.3 Attack Definitions

Within this chapter a number of different types of attack are referred to and these are described here to provide a clear explanation for the following sections.

A **Denial of Service Attack**\(^4,5\) can be achieved by a variety of methods depending on the specific system. It essentially refers to preventing valid service requests from being completed. This can be achieved by occupying the service provider with a flood of invalid or dummy requests [58, 42] or even triggering the worst case performance scenarios for certain data structures [33].

**Man-in-the-Middle Attacks**\(^6,7\) are conducted by the adversary intercepting the communications between two users. For example, Alice and Bob are two users communicating securely by using a cryptographic protocol, Mallory an adversary can communicate with Alice by pretending to be Bob and communicate with Bob by pretending to be Alice. Mallory is now in the middle of their communication and as such can read all the messages [6].

Denial of service and man-in-the-middle are both direct attacks on a protocol, whereas a **Timing Attack**\(^8\) is a side channel attack. This exploits information leaked through a timing side channel and can be particularly effective if optimisations are made to improve the performance of calculations used in the protocol [68]. For example, for elliptic curve cryptography, scalar multiplication is a key operation and any implementation that does not perform the calculations in standard time can be exploited to derive information about the values being used in the calculation [23].

**Replay Attacks**\(^9,10\) involve replaying messages to trick the recipient into thinking the same sender is contacting them again. These can be used to initiate a secure channel in which the adversary is now impersonating the original sender of the message.

A **Chosen Ciphertext Attack**\(^11\) can be implemented by an adversary when they are able to get the resulting plaintexts for one or more specifically chosen ciphertexts. The adversary can then use these decrypted plaintexts to recover the secret key that was used by the encryption protocol [95].

---

\(^3\)http://www.pgpi.org/
\(^4\)https://en.wikipedia.org/wiki/Denial-of-service_attack
\(^5\)https://capec.mitre.org/data/definitions/125.html
\(^6\)https://en.wikipedia.org/wiki/Man-in-the-middle_attack
\(^7\)https://capec.mitre.org/data/definitions/94.html
\(^8\)https://en.wikipedia.org/wiki/Timing_attack
\(^9\)https://en.wikipedia.org/wiki/Replay_attack
\(^10\)https://capec.mitre.org/data/definitions/60.html
\(^11\)https://en.wikipedia.org/wiki/Chosen-ciphertext_attack
An Adaptive Chosen Ciphertext Attack is an interactive version of the previously described chosen ciphertext attack. As with the non-adaptive version, the adversary chooses a ciphertext for which they can get the corresponding plaintext. The adaptive element is introduced by the adversary being able to use the information gained from the resultant plaintext to select the next ciphertext for decryption. Using this method the adversary can gradually learn information about the encryption key or other encrypted messages [95].

2.1.4 MIKEY

The Multimedia Internet KEYing Protocol (MIKEY) [9] is a key management scheme defined for real-time applications in heterogeneous (wired and wireless) networks. It provides the framework for key distribution between parties to set up a secure multimedia session.

MIKEY involves the party initiating the communications to send an I\_MESSAGE to the intended recipient. This is a single simplex-transmission as no verification or response message is required enabling a shared secret value to be established with minimal network communication. The I\_MESSAGE takes the form as shown in figure 2.2 [54] and the components of the I\_MESSAGE are defined as:

- HDR - Common Header Payload, describing the type of message
- T - Timestamp, used to prevent replay attacks.
- RAND - Random/psuedo-random byte string, used as a freshness value for key generation.
- IDRi - Identifier Role Initiator.
- IDRr - Identifier Role Responder.
- IDRkmsi - Identifier Role KMS of Initiator.
- IDRkmsr - Identifier Role KMS of Responder.
- CERT - Certificate, included only if an alternative to ECCSSI signature form is used, that requires certification.

Figure 2.2: I\_MESSAGE Structure

---

12https://en.wikipedia.org/wiki/Adaptive\_chosen\_ciphertext\_attack
Analysing MIKEY-SAKKE  
4th September, 2015

- SP - Security Policy(ies), specifying all security policies supported.
- SAKKE - SAKKE Encapsulated Data, including specification of the identity Scheme.
- SIGN - ECCSI Signature.
- square brackets [ ] - Optional parameter.
- braces {} - Multiple parameter.

The Identifier Role includes specification of the role of the identity as well as the identity itself. For example, for IDRi the role value is 1 for an initiator and would need to include the initiator’s identity. The definition of values for the identifier roles is given in [76] and [54].

The identities of all users are defined in MIKEY-SAKKE by the use of the "tel" URI scheme [109]. This scheme is of the form:

```
YYYY-MM\0tel:XXXXXXXXXXXXX\0
```

where:

- YYYY-MM denotes the timestamp including the year and month. This format effectively enables the monthly changing of a user’s private key(s).
- \0 denotes the null 8-bit ASCII character.
- tel:XXXXXXXXXXXXX denotes the user’s telephone number.

Once the initial I\_MESSAGE has been sent the responder can now process it to retrieve the shared secret value. To do this the responder completes the following steps [54]:

1. Check I\_MESSAGE is not already in the replay cache.
2. Extract the timestamp and check it is within the allowable clock skew.
3. Extract the initiator’s identity and authentication algorithm i.e. ECCSI.
4. Verify the signature (see section 2.1.6).
5. Process SAKKE message (see section 2.1.5) and add to replay cache.

Once a shared secret value has been established this is then used as a TEK Generation Key (TGK) where the TEK is the Traffic Encryption Key used to directly encrypt a crypto-session between the users. The exact method for generating the TEKs from the TGK is described in section 4.1 of [9].

The symmetric encryption protocol that uses the TEKs is AES which is described further in section 2.1.7.

### Positive Design Choices

As only a single simplex transmission is required to establish a shared secret value between users, MIKEY is a low latency protocol with the potential for very good performance.

MIKEY provides protection against replay attacks through the use of timestamps included in each I\_MESSAGE. This does require users to have synchronised clocks and the implementation of a dynamically sizing cache so as to manage denial of service attacks.

### Possible Weaknesses

MIKEY when used with Diffie-Hellman [9] has been shown to be vulnerable to man in the middle attacks [6] and denial of service attacks [58, 42]. Therefore it needs to be confirmed that MIKEY-SAKKE does not share the same weaknesses.

Both the positive and negative aspects highlighted here are covered in more detail in section 4.1.1.
2.1.5 SAKKE

The SAKKE protocol provides the method for key exchange between the two users such that they can establish a shared secret value that can be used in the symmetric encryption protocol, AES, for secure communications.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p$</td>
<td>a large prime integer</td>
</tr>
<tr>
<td>$F_p$</td>
<td>a finite field with $p$ elements</td>
</tr>
<tr>
<td>$E(F_p)$</td>
<td>an elliptic curve over the finite field $F_p$</td>
</tr>
<tr>
<td>$q$</td>
<td>a large odd prime integer that divides $p + 1$</td>
</tr>
<tr>
<td>$P$</td>
<td>a point on $E(F_p)$ that generates the cyclic subgroup of order $q$</td>
</tr>
<tr>
<td>$Q = [m]P$</td>
<td>elliptic curve point multiplication such that $Q$ is the result of adding $P$ to itself $m$ times</td>
</tr>
<tr>
<td>$KMS_T$</td>
<td>the key management server for domain $T$</td>
</tr>
<tr>
<td>$z_T$</td>
<td>the secret key for $KMS_T$</td>
</tr>
<tr>
<td>$Z_T$</td>
<td>the public key for $KMS_T$</td>
</tr>
<tr>
<td>$a$</td>
<td>a user’s identity</td>
</tr>
<tr>
<td>$RSK$</td>
<td>a user’s receiver secret key</td>
</tr>
<tr>
<td>$SSV$</td>
<td>a shared secret value</td>
</tr>
<tr>
<td>$n$</td>
<td>a security parameter defining the length in bits of $SSV$ and is set to 128</td>
</tr>
<tr>
<td>$\text{hash(one}</td>
<td></td>
</tr>
<tr>
<td>$R_{(b,S)}$</td>
<td>the SAKKE data intended for recipient $b$ who is provided by $KMS_S$</td>
</tr>
<tr>
<td>$(F_p)^*$</td>
<td>a multiplicative group of the finite field $F_p$</td>
</tr>
<tr>
<td>$PF_p$</td>
<td>the projectivisation of $F_p$ defined as $(F_p^<em>)/(F_p)^</em>$</td>
</tr>
<tr>
<td>$&lt; P, Q &gt;$</td>
<td>the Tate-Lichtenbaum pairing of elliptic curve points $P$ and $Q$ generating an element in $PF_p$</td>
</tr>
<tr>
<td>$g = &lt; P, P &gt; - g$</td>
<td>is the value of the Tate-Lichtenbaum [74] pairing of the generating point $P$ with itself</td>
</tr>
<tr>
<td>$g^r$</td>
<td>as $g$ is an element of $PF_p$ this is calculated using the algorithm defined in [55]</td>
</tr>
</tbody>
</table>

Table 2.1: SAKKE Notation

The SAKKE encryption uses a supersingular elliptic curve [55] $E$ defined over the prime field $F_p$ and having a subgroup of order $q$. This curve is of the form

$$y^2 = x^3 - 3x \mod p$$

(2.1)

This curve was chosen because of the efficiency and simplicity it offers [55]. A generating point, $P$, of order q is also chosen and is used to generate further points within the subgroup. This is done by multiplying the generating point by an integer in the range 2 to $q - 1$ for example:

$$Q = [m]P$$

(2.2)

where $Q$ is the result of adding $P$ to itself $m$ times.

Reversing the above scalar multiplication is essentially the elliptic curve discrete logarithm problem. In other words calculating $m$ from $P$ and $Q$ is a hard problem and forms the basis of the security of the MIKEY-SAKKE protocol.

To set up the protocol the key management server first needs to establish its own secret and public keys. For a key management server known as $KMS_T$ [55]:

1. $z_T$ is the master secret and is randomly chosen integer between 2 and $q - 1$ where $q$ is the order of the curve’s subgroup.
2. From the master secret, $z_T$, the public key, $Z_T$, can be derived as $Z_T = [z_T]P$.

The key management server then provisions each user with their key information [55]:

1. Each user already has their own unambiguous and unique identity, $a$, that is an integer in the range $2$ to $q - 1$.
2. The receiver secret key, $RSK$, is derived by the key management server as,
   
   $RSK = [(a + z_T)^{-1}]P$.

   If a user, Alice, wanted to establish a shared secret value, $SSV$ with another user, Bob, she would create the SAKKE encapsulated data using the following steps and send it to Bob [55].

   1. Alice has the user identity $a$ and is provisioned by $KMS_T$.
   2. Bob has the user identity $b$ and is provisioned by $KMS_S$, therefore this covers the scenario where Alice and Bob are in different domains.
   3. Choose the $SSV$ as a random integer in the range $2$ to $2^n - 1$ where $n = 128$ and represents the required number of bits for the symmetric encryption key.
   4. Calculate $r = hash(SSV || b)$ using the SHA-256 hash function, and $r$ is an integer in the range $0$ to $q$.
   5. Calculate $R_{(b,S)} = [r](bP + Z_S)$ to produce an elliptic curve point in the field $F_p$ on the elliptic curve.
   6. Calculate $g^r$ where $g = < P, P >$.
   7. Calculate $H = SSV \ XOR \ hash(g^r)$ to produce an integer in the range $0$ to $2^n$.
   8. Concatenate $R_{(b,S)}$ and $H$ to form the encapsulated data $(R_{(b,S)}, H)$.

On receiving the SAKKE encapsulated data from Alice, Bob can then use the below instructions to parse the data and derive the $SSV$ [55]:

   1. Extract $R_{(b,S)}$ and $H$ from the encapsulated data.
   2. Calculate $w$ as a Tate-Lichtenbaum pairing of the form $< R_{(b,S)}, RSK >$ where $RSK$ is Bob’s own receiver secret key.
   3. Calculate $hash(w)$ to produce an integer in the range $0$ to $2^n$.
   4. Calculate $SSV = hash(w) \ XOR \ H$.
   5. Check the $SSV$ value is correct:

      (a) Calculate $r = hash(SSV || b)$ giving an integer in the range $0$ to $q$.
      (b) Calculate $[r](bP + Z_S)$ and confirm the result is equal to $R_{(b,S)}$.

If the result of the last calculation does not match $R_{(b,S)}$ then the $SSV$ value must not be used and Alice must create a new value for the $SSV$ and repeat the process.

Positive Design Choices

The use of elliptic curves and so the elliptic curve discrete logarithm problem as the base of the security for the SAKKE protocol provides a higher level of security in comparison to other similar protocols not based on elliptic curves. This is because there are only exponential order algorithms for solving the elliptic curve discrete logarithm whereas sub-exponential algorithms exist for the standard discrete logarithm problem. [19, 60, 115]
SAKKE has been proven to be secure against adaptive chosen ciphertext attacks, when used as part of an identity based encryption scheme [30].

**Possible Weaknesses**

SAKKE uses a supersingular elliptic curve which is known to be vulnerable to transfer attacks that transfer the elliptic curve discrete logarithm problem into a simpler discrete logarithm problem over a finite field. [19, 60, 115]

The details of the security provided by different elliptic curves and the potential vulnerability of the SAKKE protocol to the attacks highlighted here are given in section 4.1.2.

### 2.1.6 ECCSI

Digital signatures provide a method for ensuring authenticity of a message. Elliptic Curve-Based Certificateless Signatures for Identity-based Encryption (ECCSI) [53] uses the key management server as the root-of-trust for authenticating users. The ECCSI scheme also builds on the commonly used Elliptic Curve Digital Signature Algorithm (ECDSA) [7].

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p$</td>
<td>a large prime integer</td>
</tr>
<tr>
<td>$F_p$</td>
<td>a finite field with $p$ elements</td>
</tr>
<tr>
<td>$E(F_p)$</td>
<td>an elliptic curve over the finite field $F_p$</td>
</tr>
<tr>
<td>$q$</td>
<td>a large odd prime integer that divides $p + 1$</td>
</tr>
<tr>
<td>$G$</td>
<td>a point on $E(F_p)$ that generates the cyclic subgroup of order $q$</td>
</tr>
<tr>
<td>$Q = [m]P$</td>
<td>elliptic curve point multiplication such that $Q$ is the result of adding $P$ to itself $m$ times</td>
</tr>
<tr>
<td>$KSAK$</td>
<td>the key management server secret authentication key</td>
</tr>
<tr>
<td>$KPAK$</td>
<td>the key management server public authentication key</td>
</tr>
<tr>
<td>$ID$</td>
<td>a user’s identity</td>
</tr>
<tr>
<td>$PVT$</td>
<td>a user’s public validation token</td>
</tr>
<tr>
<td>$SSK$</td>
<td>a user’s secret signing key</td>
</tr>
<tr>
<td>$hash(one</td>
<td></td>
</tr>
<tr>
<td>$(J_x, J_y)$</td>
<td>the affine coordinates for the elliptic curve point $J$</td>
</tr>
<tr>
<td>$v$</td>
<td>random variable</td>
</tr>
</tbody>
</table>

Table 2.2: ECCSI Notation

ECCSI uses the NIST P-256 [75] elliptic curve defined over the prime field $F_p$ and having a subgroup of order $q$. This curve is of the form

$$y^2 = x^3 - 3x + b \mod p$$  \hspace{1cm} (2.3)

where $b = 4105836372515212142129326129780047268409114441015993725554835256314039467401291$

A generating point, $G$, of order $q$ is also chosen and is used to generate further points within the subgroup. Like in SAKKE this is done by multiplying the generating point by an integer in the range 2 to $q - 1$ for example:

$$Q = [m]G$$  \hspace{1cm} (2.4)

where $Q$ is the result of adding $G$ to itself $m$ times.

To set up the key management server [53]:

1. Choose a random integer modulo $q$ as the KMS secret authentication key, $KSAK$. 

...
2. Derive the KMS public authentication key as $KPAK = [KSAK]G$.

To set up a user in the domain of the KMS [53]:

1. Choose $v$, as a random non-zero integer modulo $q$.
2. Compute the user’s public validation token as $PVT = [v]G$.
3. Compute the hash value $HS = \text{hash}(G||KPAK||ID||PVT)$ where $ID$ is the identity of the user.
4. Compute the user’s secret signing key as $SSK = (KSAK + HS \cdot v) \mod q$.
5. Confirm $SSK$ and $HS$ are non-zero modulo $q$, if not then repeat the above steps with a new value for $v$.
6. Send the $(SSK, PVT)$ pair to the user via an secure key provision mechanism.

On receipt of their $(SSK, PVT)$ pair, the user must validate these by [53]:

1. Checking the $PVT$ is a point on the elliptic curve $E$.
2. Computing $HS = \text{hash}(G||KPAK||ID||PVT)$.
3. Validating $KPAK = [SSK]G - [HS]PVT$. If this relationship is not valid then the keys cannot be used for authentication.

To sign a message $M$ the user [53]:

1. Chooses $j$, as a random non-zero integer modulo $q$.
2. Computes the elliptic curve point $J = [j]G$ where the affine coordinates of the point $J$ are $(J_x, J_y)$.
3. Assigns $r = J_x$.
4. Computes $HS = \text{hash}(G||KPAK||ID||PVT)$.
5. Computes $HE = \text{hash}(HS||r||M)$.
6. Confirms $HE + r \cdot SSK \mod q$ is non-zero otherwise repeat the above steps with a new value for $j$.
7. Computes $s = (((HE + r \cdot SSK)^{-1}) \cdot j) \mod q$.
8. Sends the message $M$ with the signature $(r||s||PVT)$ to intended recipient.

To verify a signature $(r||s||PVT)$ received with message $M$ from a user with identity $ID$, the recipient [53]:

1. Checks the $PVT$ is an elliptic curve point on curve $E$.
2. Computes $HS = \text{hash}(G||KPAK||ID||PVT)$.
3. Computes $HE = \text{hash}(HS||r||M)$.
5. Computes $J = [s]([HE]G + [r]Y)$ where the affine coordinates of the point $J$ are $(J_x, J_y)$.
6. If $J_x \mod p \neq 0$ then accepts signature as valid.
Positive Design Choices

The use of elliptic curves and so the elliptic curve discrete logarithm problem as the base of the security for the ECCSI protocol provides a higher level of security in comparison to other similar protocols not based on elliptic curves. This is because there are only exponential order algorithms for solving the elliptic curve discrete logarithm whereas sub-exponential algorithms exist for the standard discrete logarithm problem. [19, 60, 115]

Integrates randomness into the choice of the user $PVT$ and $SSK$ making it more difficult for an attacker to derive these values even if they have compromised the key management server.

Possible Weaknesses

ECCSI is based on the commonly used signature protocol ECDSA for which there are a number of successful attacks, including the high profile attack on Sony [4]. See section 2.3.3 for further details of attacks on the ECDSA protocol.

The SafeCurves website [12] has evaluated a wide range of elliptic curves and highlights a number of potential weaknesses for the NIST P-256 curve that could be exploited by attackers.

The specifics for how the NIST P-256 curve is categorised against the SafeCurves criteria is given in section 4.1.3 as well as further analysis of the different strengths and weaknesses identified.

2.1.7 AES

The Advanced Encryption Standard (AES), also known as Rijndael, [34] is a symmetric cryptographic protocol defined as part of a competition in 2001 to find an improved protocol to supersede the Data Encryption Standard (DES) [88]. In the MIKEY-SAKKE cryptosystem, AES is used to encrypt the symmetric communications between two (or more) authenticated users once they have established a shared secret value.

AES is an iterated block cipher for a set block length of 128 bits and variable key lengths of 128, 192 and 256 bits. This is actually a subset of the original Rijndael proposal that also allowed for a variable block length. In addition MIKEY-SAKKE further restricts the key length used to be 128 bits [54].

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N_b$</td>
<td>block length divided by 32</td>
</tr>
<tr>
<td>$N_k$</td>
<td>key length divided by 32</td>
</tr>
<tr>
<td>$N_r$</td>
<td>number of rounds or number of times the round transformation is applied to the message</td>
</tr>
<tr>
<td>$GF(2^8)$</td>
<td>binary finite field with $2^8 = 256$ elements</td>
</tr>
</tbody>
</table>

Table 2.3: AES Notation

The following defines the components needed for the AES cryptographic protocol [34]:

- State is the immediate cipher result and can be pictured as a rectangular array of bytes of size $(4 \times N_b)$ where $N_b = \text{blocklength}/32 = 128/32 = 4$.

- Cipher key is the key as mentioned above and can also be pictured as a rectangular array of bytes with the size $(4 \times N_k)$ where $N_k = \text{keylength}/32 = 128/32 = 4$.

- Number of rounds is the number of times the round transformation is applied to the message to obtain the cipher and is a function of $N_b$ and $N_k$ such that as $N_b = 4$ and $N_k = 4$ then $N_r = 10$. 

22
• The round key is derived from the cipher key via the key schedule.

• Key schedule has two components:
  – Key expansion expands the cipher key into an expanded key.
  – Round key selection chooses a round key from the expanded key.

The round transformation is applied to the above components and is defined by the pseudocode specified below [34]. The Round function is applied for the first \(N_r - 1\) round transformations and the FinalRound function is applied for the final \(N_r\) round.

```c
Round(State, RoundKey) {
    ByteSub(State);
    ShiftRow(State);
    MixColumn(State);
    AddRoundKey(State, RoundKey);
}

FinalRound(State, RoundKey) {
    ByteSub(State);
    ShiftRow(State);
    AddRoundKey(State, RoundKey);
}
```

The functions used in the round transformation functions complete the actions specified below [34].

• ByteSub transformation is a non-linear byte substitution operating on each of the state bytes independently.

• ShiftRow transformation cyclically rotates the rows of the state over different offsets.

• MixColumn transformation considers the columns of the state as polynomials over \(GF(2^8)\) and multiplies each a fixed polynomial \(c(x)\).

• AddRoundKey transformation applies the round key to the state by a simple bitwise EXOR.

The function for the Rijndael cipher, defined in pseudocode can now be given as [34]:

```c
Rijndael(State, CipherKey) {
    KeyExpansion(CipherKey, ExpandedKey);
    AddRoundKey(State, ExpandedKey);
    for (i = 1 ; i < N_r ; i++)
        Round(State, ExpandedKey + N_b * i);
    FinalRound(State, ExpandedKey + N_b * N_r);
}
```

where \(N_b\) is \(N_b\); and \(N_r\) is \(N_r\).

Using the Rijndael cipher function the two (or more) users can now establish secure communications between them using their shared secret value as the key.

### Positive Design Choices

AES is widely used protocol and as such is well studied and analysed. It shows good performances for multimedia applications [5, 38, 125] and as such is a good choice for the MIKEY-SAKKE use case.

A number of theoretical direct protocol attacks against AES have been designed [32, 46, 41, 85, 86, 17, 16, 62], however, none of these are deemed to be practically implementable [108].

### Possible Weaknesses

There are a number of different researchers that have shown examples of side-channel timing attacks on AES [122, 10, 57, 91].

These exploits are discussed further in section 4.1.4.
2.1.8 AES Key Wrap

The AES key wrap [89, 102] was defined to satisfy the requirement of the National Institute of Standards and Technology (NIST) [87]. This requirement was: “Design a cryptographic algorithm called a Key Wrap that uses the Advanced Encryption Standard (AES) as a primitive to securely encrypt a plaintext key(s) with any associated integrity information and data, such that the combination could be longer than the width of the AES blocksize (128-bits).” [89].

In order to avoid confusion when specifying the key wrap protocol the key that is to be wrapped by the protocol and sent as a cipher is referred to as the plaintext; and the key that is used to encrypt that plaintext key data is referred to as the key.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IV</td>
<td>initial value</td>
</tr>
<tr>
<td>n</td>
<td>number of plaintext blocks to be encrypted</td>
</tr>
<tr>
<td>P₁, P₂, P₃, ..., Pₙ</td>
<td>plaintext blocks</td>
</tr>
<tr>
<td>K</td>
<td>key to be used for encryption</td>
</tr>
<tr>
<td>C₀, C₁, ..., Cₙ</td>
<td>ciphertext blocks</td>
</tr>
</tbody>
</table>

Table 2.4: AES Key Wrap Notation

The AES key wrap protocol has a fixed block length of 64 bits and a variable key length. The plaintext to be wrapped is parsed into \( n \) 64 bit blocks, where \( n \geq 2 \). The Secure Chorus implementation restricts the AES key wrap encryption key lengths used to either 128 or 256 bits [112].

The key wrap algorithm [89, 102]:

1. Initialise the variables using the specified initial value (IV).

2. Encrypt the \( n \) 64-bit plaintext blocks \( \{P₁, P₂, ..., Pₙ\} \) using the AES codebook with key \( K \).

3. Output the \( (n + 1) \) 64-bit ciphertext blocks \( \{C₀, C₁, ..., Cₙ\} \).

The key unwrap algorithm [89, 102]:

1. Initialise the variables using the first 64-bit ciphertext block \( C₀ \).

2. Decrypt the \( (n + 1) \) 64-bit ciphertext blocks \( \{C₀, C₁, ..., Cₙ\} \) using the AES codebook with key \( K \).

3. Check the integrity value now held is an appropriate IV.

4. Output the \( n \) 64-bit plaintext blocks \( \{P₁, P₂, ..., Pₙ\} \).

The initial value (IV) referred to in the algorithms above is used to verify the data integrity. The default IV is the hexadecimal constant A6A6A6A6A6A6A6A6 [89, 102].

The Secure Chorus reference implementation [113] uses the AES key wrap protocol in order to encrypt key information sent to the user from the key management server. The key management server communicates with users in its domain using HTTP with the content formatted by XML. There are three different requests that a user can make to the key management server as defined by Secure Chorus [112]:

1. Initialisation - initialises the user with the key management server by verifying the user’s identity and associated boot key.

2. Key provision - provides the user with unique SAKKE and ECCSI keys as well as details of the KMS certificate including the SAKKE KMS public key and the ECCSI KMS public authentication key.
3. Certificate cache update - provides the user with certificates for any other trusted key management servers such that the user can communicate with users in other domains.

The AES key wrap algorithm also has the same strengths and weaknesses as covered for the AES protocol itself in section 4.1.4. There may be additional areas of strength or weakness to be evaluated for the key wrap specification but these have not been covered in this project.
2.2 Analysis Techniques

Perfect security is unattainable for most real systems [100] and so security analysts require methods and tools to make informed decisions about the level of security a system provides. Continuous Iterative Improvement is an approach to security analysis that incorporates both high level analysis techniques and low level exploit investigations in order to identify improvements to the protocol design; implementation bug fixes; and defensive preventative measures.

Figure 2.3, shows how information to trigger each of these different specific elements can be obtained from the high level security analysis and then the are results fed back into it. This cyclic process allows for the improvement of the accuracy and completeness of the security model as well as improvement of the effectiveness of the security provided by the protocol being analysed.

The high-level security analysis techniques aim to provide as broad an analysis as possible for a given protocol, taking into account the specific protocol design; the environment in which the protocol is used; and the different adversaries likely to attempt an attack.

There are qualitative security methods that provide effective models but they fail to provide the ability to directly compare specific prevention mechanisms and determine which specific alterations would provide the most added value. Quantitative security metrics, however, do allow for this direct comparison, at least in relative terms, which enables analysts to make effective decisions especially in balance with cost and performance trade-offs; and provides an accurate output from the overall Continuous Iterative Improvement which can be shared as substantiation or contradiction to the claimed level of security provided by the protocol.

There are a number of different techniques that can be used for broad system security analysis. In the following sections I detail Attack Trees [105]; Attack-Defense Trees [70]; ADVISE [72, 73] in conjunction with Möbius [44, 35]; and Security Assurance Cases [121] all as potentially useful techniques for this project.

2.2.1 Attack Trees

Attack trees [105] provide a formal and methodical way of describing the security of systems based on varying attacks. The root of the attack tree represents the security goal of the attacker, for example “Open Safe”. Figure 2.4 shows an example attack tree with this goal [105]. The child nodes to “Open Safe” represent attack steps that can achieve the goal. There are two types of node:
Analysing MIKEY-SAKKE

Figure 2.4: Example Attack Tree [105]

- OR node - for example, in figure 2.4 “Learn Combo” is an OR node as either “Find Written Combo” or “Get Combo From Target” need to be achieved to achieve “Learn Combo”.

- AND node - for example, in figure 2.4 “Eavesdrop” is an AND node as both “Listen to Conversation” and “Get Target to State Combo” need to be achieved to achieve “Eavesdrop”.

Values can also be assigned to the nodes on an attack tree [105]. These may be binary or continuous and may represent a variety of different attributes. For example, a binary attribute of “Possible” or “Impossible”; and a continuous attribute of “Cost to Attacker”. The analyst assigns specific values for these attributes to the leaf nodes and from these can use boolean logic to calculate the cost or possibility etc. of nodes higher up the tree and ultimately that of the security goal. From these calculations quantitative metrics can be obtained, such as the lowest cost attack or the most likely attack.

In addition, using attack trees in conjunction with adversary profiles allows metrics to be obtained for specific types of adversary. This is most useful for risk analysis of systems where there is a balance between the cost of preventing a specific attack and the risk of that attack actually occurring.

As a progression on [105]’s definition of attack trees, [124] expand on this to introduce a semi-adaptive model by including a temporal order of elementary attacks. This enables a more accurate representation of the adversary’s behaviour.

2.2.2 Attack-Defense Trees

Security systems are not static and defenders of a system will implement further security measures as they attempt to prevent attackers from achieving their goals. Attack-defense trees [70] were developed as a progression on attack trees to provide an expression of the actions of the defender of a system as well as the attack steps attempted by the adversary.

Similar to the attack tree the attack-defense tree has AND and OR nodes, also described as conjunctive and disjunctive nodes. These nodes are then further categorised as attack and defense nodes, as shown in the example in figure 2.5. There are two types of relation in an attack-defense tree [70]:

- Refinement: each node may have one or more child nodes, of the same type, that are sub-goals of itself and thus refine the current node. These relationships are represented by a solid line in the example shown in figure 2.5.
• Countermeasure: each node may also have one child node, of the opposite type, representing a countermeasure against the current node. For example, in figure 2.5, the attack node “Find Note” is countered by the defense node “Memorise”, which is in turn countered by the attack node “Force”. All countermeasure relations are shown with a dotted line.

The attack-defense tree represents an attack-defense scenario, which is essentially a game between two players: the proponent and the opponent [69]. If the root of the tree is an attack node then the proponent of the game is the attacker and the opponent is the defender; and vice versa if the root is a defense node. From this scenario description, the attack-defense tree can be converted into a binary zero-sum two-player extensive form game providing an alternative model under the well-studied methodology of game theory [69]. [18] also describes attack scenarios and game theory to detect the most promising actions for each the attack and the defender.

2.2.3 ADVISE and Möbius

ADversary View Security Evaluation (ADVISE) as first defined in [72] provides quantitative metrics based on system characteristics and adversary profiles to aid security decisions. It uses simulation algorithms to compute the attack decisions an adversary makes and the probabilities associated with these. From these different elements an executable model is produced that computes relevant and quantitative, security metrics.

Adversaries of the system are precisely characterised based on the following attributes [72]:

• Attack preference weights: describe the relative attractiveness to an adversary in each of four core criteria: cost, payoff, probability of success and probability of detection. These are each represented by a real number between 1 and 0.

• Attack skill levels: describe the proficiency of the adversary in executing specific types of attack and again are represented by a real number between 1 and 0.

• Attack goals: are specific to the system and describe the end goal of the adversary. For each system goal the adversary has a binary value that is 1 if it is a goal of this adversary and 0 if not.

• System knowledge: describes pieces of system specific knowledge that can aid the adversary. For each piece of information, the adversary has an assigned binary value: 1 if the information is known to the adversary and 0 if not.
System access: describes the system specific network domains or physical locations through which adversaries can attack the system. For each distinct type of access, the adversary has an assigned binary value: 1 if the adversary has this access and 0 if not.

Under the ADVISE formalism, the system is characterised by an attack execution graph. All attacks on a system are a chain of attack steps that lead to an attack goal, these are grouped into the attack execution graph and thus represent all the different ways an adversary can attack a system. An example of an attack execution graph is shown in figure 2.6 [72] and this incorporates the following nodes: attack goals; attack steps (defined further below); attack skills; system knowledge elements; and system access elements.

Each attack step in the attack execution graph is precisely defined to enable an effective simulation of the adversary’s decision process during an attack. The attack step definition includes [72]:

- Attack precondition: specifies the skill levels, system knowledge and system access elements the adversary must possess to attempt the attack step.
- Execution time: is the time needed for the adversary to execute the attack step.
- Cost: is the cost incurred if the adversary attempts the attack step irrespective of whether or not it is successful.
- Set of outcomes: are the possible results of an attempted step.
- Outcome distribution: is the probability distribution of the set of outcomes if the attack step is attempted by the adversary.
- Detection distribution: is the probability distribution indicating the likelihood of the adversary being detected when attempting an attack step.
- Payoff: is the value to the adversary of attempting the attack step and achieving a particular outcome.
- State variable updates: are the changes that occur to the overall state model due to a particular attack step outcome.
In order to acquire relevant metrics from the ADVISE model, these must first be specified as customised variables. These metrics are categorised into two types: system-focused and adversary-focused [72]. System-focused metrics compare the strength of different security configurations, for example, the most likely attack path or the average time for an adversary to achieve an attack goal. Adversary-focused metrics examine how changes in an adversary characterisation affect the security of a system, for example, for a selection of adversaries, which is able to achieve a particular attack goal in the shortest average time.

The above definitions provide the input to the executable simulation model that produces results for the specified security metrics. There are three phases to the adversary attack cycle that is simulated by the model [72]:

- Attack step precondition evaluation: a comparison of the attack step precondition with the adversary profile to see if the adversary has the skills, knowledge and access elements needed to attempt the attack step.

- Attack step attempt decision: choice of the most attractive attack step from those for which the precondition is met. This is based on a balance of the cost and payoff to the adversary as well as the likelihood of detection. There is always the option of the “Do Nothing” attack step, which may be the most attractive option to the adversary if, for example, all other available attack steps have high cost and low payoff values.

- Attack step attempt outcome: selection of the outcome that occurs based on the set of possible outcomes as well as the outcome distribution for the attempted attack step. This also specifies the payoff and state update variables as a consequence of the outcome.

This cycle is repeated to simulate an adversary or multiple adversaries attacking the system and measurements are made throughout the simulation to provide the resulting metrics.

The ADVISE model was further developed in [73] to define the state look-ahead tree. The state look-ahead tree is used in the adversary attack cycle at the attack step attempt decision stage. The description above only considers the attractiveness of the immediate attack steps to the adversary, however, in reality an adversary will be thinking about future attack steps and their attractiveness as well as the immediate ones. For example, an attractive, immediate attack step may subsequently lead to some very unattractive attack steps making that attack impossible to the adversary therefore the first attack step is not actually as attractive as it might first appear.

The state look-ahead tree [73] considers a long-range-planning adversary instead of a short-sighted one. It allows for computation of the relative attractiveness of the immediate attack steps based on the following N attack steps’ attractiveness, where N is defined as the planning horizon of the adversary i.e. the number of steps into the future the adversary can plan.

[43] extends the capabilities of ADVISE by using Markov chain analysis techniques instead of simulation. This introduces the efficient and accurate computation of a wider variety of security metrics thus expanding the available functionality of the ADVISE model.

Finally, the team who developed the ADVISE model have also created the tool, Möbius. Möbius is a discrete-event modelling environment and has been used for modelling system performance and dependability. The implementation of the ADVISE security modelling formalism is described in [44] and [35]. Möbius also provides a graphical front-end for creating and modifying ADVISE models.

2.2.4 Security Assurance Cases

A security assurance is a body of evidence organised into an argument demonstrating that some claim about a system holds. [121] uses the Goal Structuring Notation [65] to represent a security assurance case in graphical form.

To create a security case [121]:

- Identify top level claim: The high-level claim that is to be assured, for example, “System X is acceptably secure”.

30
• Identify strategy: The plan for how to substantiate the top level claim.
• Elaborate strategy: Define the sub-claims that refine the top level claim and fulfil the identified strategy.
• Evidence: Basic sub-claims, those that can no longer be refined and are directly supported by evidence.

Security assurance cases demonstrate the system’s security requirements and how they have been met. The case should evolve with the system throughout its lifetime to reflect changes and developments [121]. This evolution of the security assurance case fits in well with the approach of Continuous Iterative Improvement, as discussed at the start of section 2.2.

2.3 Vulnerabilities in Other Protocols

A system can be exploited in many different ways: through weaknesses in the implementation; vulnerabilities in the specification of the protocol itself; or even flaws introduced through regulations imposed on the system. Below I have covered a selection of these, which is by no means comprehensive but aims to provide interesting examples for which the root causes can be considered and related to specific points of investigation for MIKEY-SAKKE.

2.3.1 General

These general attacks can be applied to a wide range of systems and protocols; and all are potential vulnerabilities for MIKEY-SAKKE.

Generation of Random Numbers. Many cryptographic protocols rely on random numbers, however the effective generation of random numbers is difficult and relies on a significant source of entropy. Analysis of the software based random number generators provided in virtualised Linux machines indicated the numbers generated are not-so-random [39]. This not-so-random behaviour can often be exploited for resumption based attacks. Further articles describing attacks exploiting weaknesses in pseudo-random number generators include [103] and [66].

Denial of Service. Many data structures are used for implementing systems and protocols and these are often chosen for their efficient average running times. However, these data structures often have much less efficient worst case running times that can be exploited by adversaries in a denial of service attack. This principal is used by [33], for a denial of service attack mounted across a network exploiting the worst case of a selection of widely used hash table implementations. They show how an attacker can effectively compute a malicious client input and demonstrate significant degradation of server performance resulting in denial of service.

Timing Attacks exploit the different amounts of time it takes for a cryptosystem to process different inputs. This can be due to perform optimisations in the protocol itself or underlying performance optimisations such as RAM cache hits and processor instructions that run in non-fixed time. [68] describes how an attacker eavesdropping on an interactive protocol using Diffie-Hellman [36] or RSA [98] could determine the user’s secret key from public information and the response times. [68] also describes methods the defender can take to prevent such an attack by adapting techniques used for blindness signatures [29].

2.3.2 SSL and TLS

SSL, Secure Socket Layer,13 and TLS, Transport Layer Security,14 are widely used security protocols that provide secure communications over insecure infrastructure [97]. SSL was first released in November and was renamed and re-released as TLS in 1999 as it was migrated from Netscape to Microsoft [97]. This long standing protocol has evolved with many different versions and also many

---

13http://docs.oracle.com/cd/E19957-01/816-6156-10/contents.htm
weaknesses and vulnerabilities have been identified along the way. Just a few of these are described below.

**CRIME** [49] stands for Compression Ratio Info-leak Made Easy. This attack exploits server and browser communications where both support either SPDY\(^\text{15}\) (pronounced “speedy”) an open compression scheme or the TLS compression scheme known as Deflate.\(^\text{16}\) It is a chosen plaintext attack where the attacker mixes plaintext through JavaScript with the encrypted message to determine, letter by letter, the secret key contained in the authentication cookie for the session.

It has long been known that data compression provides a side channel, through which data can be leaked to adversaries [49]. It reduces the number of bytes in a file or data stream by removing redundant information. If a request using data compression returns fewer encrypted network packets then there must be more data in common between the attacker inserted data and the secret data. Providing the attack with a stronger base from which to perform a brute force attack.

The same researchers Juliano Rizzo and Thai Duong who identified CRIME in 2012 also identified **BEAST** [45] in 2011. The BEAST exploit targets the cipher block chaining encryption of SSL/TLS and builds on the previously known weakness of the predictable initialisation vector [82]. Using this the attacker can reduce the communication to electronic book mode, which is inherently insecure.

**Poodle** [22] is the Padding Oracle On Downgraded Legacy Encryption attack against SSL that allows a man-in-the-middle, such as a malicious Wi-Fi hotspot or a compromised ISP, to extract data from secure HTTP connections.

For example, this can be completed by an adversary injecting JavaScript into a session that repeatedly sends requests to the server. In each of these requests is the session cookie and the injected JavaScript is constructed in such a way as to ensure that one byte of the cookie is placed in a particular location within the message. The adversary then reorganises the message to put the portion of session cookie at the end of the message. Most of the time this fails when the server tries to decrypt it but occasionally (1 in every 256 attempts) it will be successfully decrypted and the adversary learns that portion of the session cookie. Repeatedly doing this allows the adversary to gain the entire session cookie which they can use to impersonate the victim and thus perform the man in the middle attack [22].

**Insecure SSL/TLS Renegotiation** is a protocol flaw that can be exploited such that the attacker is able to break the confidentiality of the secure communication and conduct a man-in-the-middle attack [15, 48, 96]. To do this the attacker:

- Intercepts the TCP connection request from client to server.
- Opens a new TLS connection to server and sends his attack payload.
- Continues to operate as a transparent proxy between the client and server.
- The client submits a new TLS handshake to establish a connection.
- The server has already seen the attacker’s TLS connection so interprets this as a renegotiation.
- Once renegotiated the client and server continue to exchange data and now the attacker’s payload and the client’s data are both part of the same data stream.

**Heartbleed**, reference CVE-2014-0160 [1, 52], is a specific implementation error in the heartbeat extension of the TLS protocol. The implementation mistake is a simple missing bounds check, such that there is no check that the requested size of data is within the allocated buffer size so the function returns contents of the main memory up to 64KB in length. This attack can be repeated as many times as the adversary wishes, for example, until it returns confidential data like password or session ticket keys. This bug is present in OpenSSL\(^\text{17}\) versions 1.0.1 to 1.0.1f and upgrading to

\(^{15}\)https://developers.google.com/speed/spdy/?hl=en

\(^{16}\)http://www.w3.org/Graphics/PNG/RFC-1951

\(^{17}\)https://www.openssl.org/
version 1.0.2 provides a resolution to this exploitation [52]. However, it is a clear example of how a small oversight in implementation can undermine the security of the entire cryptosystem.

In the 1990s the United States government mandated a rigorous regime of export controls for encryption systems in order to deny current and potential enemies access to their cryptographic systems [50]. Therefore US companies were required to deliberately weaken the strength of their encryption key to 512 bits, when communicating outside of the US. To support these export ciphers SSL provided a negotiation mechanism to identify the capabilities of both parties and use the best cipher that both could support. The US government has since lifted the export controls, however, the SSL functionality still remains and is supported by a significant number of servers. The following attack describe how the export ciphers can be exploited to the benefit of an adversary.

**FREAK** or factoring Attack on RSA-Export Keys [83, 50, 51] involves the adversary conducting a man-in-the-middle attack and forcing down the security of the connection so that the decryption key can be factored. All the adversary needs to do is intercept the clients message and change it to request an export RSA cipher, the server then responds with a 512-bit key, which the adversary can factor to recover the corresponding decryption key. The adversary can now read the communication in plaintext and inject in additional information.

Even though these SSL/TLS attacks are not directly related to the MIKEY-SAKKE protocol they provide insight into the different aspects of a protocol that can be exploited in order to attack the system. Protocol vulnerabilities can occur due to ineffective protocol design; implementation mistakes; side channels; and even regulations as shown in the case of the export cipher.

### 2.3.3 ECDSA

ECDSA is the Elliptic Curve Digital Signature Algorithm [64]. ECDSA was developed as an analogue to the Digital Signature Algorithm, DSA, [71]. DSA relies on the discrete logarithm problem for its security whereas ECDSA relies on the elliptic curve discrete logarithm problem. There are a number of sub-exponential algorithms known for solving the discrete logarithm problem but there are only exponential solutions for the elliptic curve discrete logarithm problem [19, 60, 115]. Therefore ECDSA was developed to provide a higher level of security per bit in comparison to DSA [64].

ECDSA is now a commonly used authentication protocol and so there are also a number of attacks that exploit vulnerabilities associated with it.

In 2010, there was a particularly high profile attack on Sony in which a group called fail0verflow were able to recover the ECDSA private key used to sign Sony’s PlayStation 3 software. To do this they exploited a weakness in the ECDSA protocol implementation as used by Sony, where the private random variable, m, was static when it should have had a different value for each signature. fail0verflow were successfully able to derive the private key from two different signatures with the same random variable value [4].

As described in section 2.3.1 timing attacks can be an effective method for exploiting a side channel to gain information. [23] demonstrate a timing attack that exploits the implementation of elliptic curve scalar multiplication. Scalar multiplication is a critical component when working with elliptic curves in cryptography as is discussed in more detail in section 4.1.3. Using the information leaked from the timing of the OpenSSL elliptic curve scalar multiplication [23] are able to retrieve the ECDSA private key.

ECDSA is also used as the authentication protocol for Android apps. These apps use the SecureRandom Java class [63], which due to an implementation error the pseudo random number generator creates collisions in the private random variable, m. As in the case of fail0verflow’s attack on Sony, once two signatures with the same random variable value have been identified then the ECDSA private key can be derived and used by the attacker to steal bitcoins from the user [31].

---

18http://www.sony.co.uk/
19https://fail0verflow.com/about.html
20https://www.android.com/
21https://bitcoin.org/en/
The ECCSI authentication protocol, described in section 2.1.6, is based on the ECDSA protocol therefore there is the possibility that some of the attacks described in this section for ECDSA could also be applied to ECCSI. This is investigated in more detail in section 4.1.3.
Chapter 3

High level Security Analysis

3.1 Theory

For the broad protocol analysis I focused on the ADVISE formalism and used the Möbius tool to capture and model the system and potential adversaries.

3.1.1 Goals

In order to define the attack goals of a system using MIKEY-SAKKE, I started by researching different threat models to find an accurate categorisation of the different goals and ensure I was covering all possible types of threat. The STRIDE threat model [78] specifies six different and clearly defined categories each relating to a security property of a protocol.

For each of the STRIDE categories I identified goals specific to the MIKEY-SAKKE protocol as shown in table 3.1.

<table>
<thead>
<tr>
<th>Threat</th>
<th>Property Violated</th>
<th>Specific Attack Goals</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spoofing</td>
<td>Authentication</td>
<td>Impersonate user</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Take over domain</td>
</tr>
<tr>
<td>Tampering</td>
<td>Integrity</td>
<td>Tamper with _MESSAGE</td>
</tr>
<tr>
<td>Repudiation</td>
<td>Non-repudiation</td>
<td></td>
</tr>
<tr>
<td>Information Disclosure</td>
<td>Confidentiality</td>
<td>Read message</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Read all messages</td>
</tr>
<tr>
<td>Denial of Service</td>
<td>Availability</td>
<td>User denial of service</td>
</tr>
<tr>
<td></td>
<td></td>
<td>KMS denial of service</td>
</tr>
<tr>
<td>Elevation of Privilege</td>
<td>Authorisation</td>
<td>Create malicious user</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Create malicious domain</td>
</tr>
</tbody>
</table>

Table 3.1: STRIDE

3.1.2 Attacks

From the definition of the potential goals of the system I then defined potential attack paths to reach each of the goals. To do this I used the different skills of the adversaries along with a dictionary of known attacks, called CAPEC [81]. CAPEC stands for Common Attack Pattern Enumeration and Classification and was devised by MITRE\(^1\) to provide a comprehensive dictionary of known attacks as well as clear classification of these attacks for use by analysts, developers, testers and any other security roles [81].

\(^1\)http://www.mitre.org/
From these known attacks I was able to apply them to the MIKEY-SAKKE protocol to produce the attack execution graph as a specification of the system in the ADVISE formalism. Section 3.2 shows the exact attack paths for the different goals and explains the detail of what is required to achieve each of the attack steps as specified in M"obius.

3.1.3 Adversaries

For the adversaries I wanted to capture a wide range as well as accurately describe individual attributes to ensure accurate results for which type of adversary would attempt which attack. I found the Intel Threat Agent Library (ITAL) [25], which is a comprehensive library of 22 distinct adversaries covering a complete range based on Intel’s experience of system attacks. The specification used to define the different attributes of each adversary is shown in table 3.2.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access</td>
<td>An adversary’s access to the company’s assets using two categories: Internal or External.</td>
</tr>
<tr>
<td>Outcome</td>
<td>An adversary’s primary goal, from the options: Acquisition/Theft; Business Advantage; Damage; Embarrassment; and Tech Advantage.</td>
</tr>
<tr>
<td>Limits</td>
<td>The legal and ethical limits of the adversary i.e. what rules/laws the adversary is prepared to break in order to achieve an outcome from: Code of Conduct; Legal; Extra-legal, minor; and Extra-legal, major.</td>
</tr>
<tr>
<td>Resources</td>
<td>The resources available for the adversary to use in an attack and is linked to the Skills attribute, categorised by: Individual; Club; Contest; Team; Organisation; and Government.</td>
</tr>
<tr>
<td>Skills</td>
<td>The abilities of the adversary that can be utilised in an attack, defined as: None; Minimal; Operational; and Adept.</td>
</tr>
<tr>
<td>Objective</td>
<td>The attack steps that an adversary will take to achieve an outcome: Copy; Deny; Destroy; Damage; Take; All of the Above/Don’t Care.</td>
</tr>
<tr>
<td>Visibility</td>
<td>The extent to which the adversary intends to conceal their identity, categorised as: Overt; Covert; Clandestine; Multiple/Don’t Care.</td>
</tr>
</tbody>
</table>

Table 3.2: Intel Threat Agent Library [25]

The Intel Threat Agent Library uses different classifications to the ADVISE formalism and M"obius so I defined relationships between the attributes in each of the models and translated the ITAL adversary definitions into specifications using the ADVISE attributes. The relationships I defined are described in table 3.3.

There is no direct relation for the planning horizon attribute but it can be inferred from the skill and access levels. The level of skill for each adversary is given a value in the range 1 to 4 where 1 is None and 4 is Adept. Similarly, the level of access is defined as a numeric value where 1 is Internal and 0 is external. Adding these two values together gives a value in the range 1 to 5 where a low skilled, external adversary will have a low planning horizon and a high skilled, internal adversary will have a high planning horizon.

For the cost an adversary is willing to accept when executing an attack step, this can be directly related to the resources available to that adversary. Such that if an adversary has greater resources then they are able to accept larger costs when attacking at system. Resources are given a value in the range 6 to 1 where 6 corresponds to an Individual and 1 to a Government. For the Cost, Detection and Payoff attribute these are relative values summing to a value of 1 representing the adversary’s relative prioritisation of each, which therefore dividing by the sum of all three gives a real number between 0 and 1;

Detection is inversely related to the adversary’s visibility such that a clandestine adversary will try to avoid detection at all costs. Visibility is rated in the range 1 to 3 such that 3 is Clandestine.
and 1 is Overt or Multiple/Don’t Care. As with cost this is then divided by the sum of resources, visibility and outcomes to get a real number between 0 and 1;

There is no direct relation in the Intel Threat Agent Library for the payoff attribute but it can be inferred from the outcomes attribute and common sense interpretation based on the overall adversary description. Outcomes is given a value in the range 1 to 3 defined as

1. Fewer outcomes usually Damage and/or Embarrassment.

2. Fewer high profile outcomes namely Acquisition/Theft, Business Advantage or Tech Advantage.

3. 2 or more high profile outcomes.

Again these are divided to get a real number between 0 and 1.

In the ADVISE formalism access elements are associated with each adversary therefore even though there is a direct relation with the Intel Threat Agent Library access attribute this needs to be broken down further for the M¨ obius tool. As each specific access element is identified then it can be categorised as Internal and/or External. Each adversary is then given the specific access elements associated with its access level as defined in the Intel Threat Agent Library.

Similarly, in ADVISE specific knowledge elements are defined instead of general knowledge levels. There is no direct relation for knowledge but this can be inferred through a combination of the adversary access and general common sense of the adversary description.

There is a direct relationship for the skills attribute and as with the access attribute, the individual M¨ obius skill elements can be associated with the overall Intel skill categories. Therefore the skill elements associated with the skill level the adversary is defined as having in the Intel Threat Agent Library can be associated with the adversary in the M¨ obius model.

For the goals I went back to the STRIDE threat model, that I initially used to define the potential goals for MIKEY-SAKKE. I then allocated the STRIDE threats based on the overall adversary specifications and assigned the relevant goals for each.

Attack steps can be seen as clearly related to objectives however, M¨ obius does not restrict the adversaries for which attack steps they attempt. So even though this is not something that can be input into the simulation it could be used in analysis of the results, such that if an adversary is executing a specific attack in the simulation that uses many objective that are not assigned to it in the Intel specification then maybe overall specification of the model and the adversary should be reconsidered.

The limits attribute included in the Intel threat agent library specification does not appear to clearly relate to any of the ADVISE attributes therefore it cannot be captured in this representation of the adversaries.

<table>
<thead>
<tr>
<th>Möbius</th>
<th>Intel</th>
<th>Relationship</th>
</tr>
</thead>
<tbody>
<tr>
<td>Planning</td>
<td>Skills</td>
<td>Planning Horizon = Skills + Access (1-5)</td>
</tr>
<tr>
<td>Horizon</td>
<td>Access</td>
<td>Cost = Resources/(Resources + Visibility + Outcomes)</td>
</tr>
<tr>
<td>Cost</td>
<td>Resources (1-6)</td>
<td>Detection = Visibility/(Resources + Visibility + Outcomes)</td>
</tr>
<tr>
<td>Detection</td>
<td>Visibility (1-3)</td>
<td>Payoff = Outcomes/(Resources + Visibility + Outcomes)</td>
</tr>
<tr>
<td>Payoff</td>
<td>Outcomes (1-3)</td>
<td>Access Specific to each Möbius access element.</td>
</tr>
<tr>
<td>Access</td>
<td>Access</td>
<td>Knowledge Specific to each Möbius knowledge element.</td>
</tr>
<tr>
<td>Knowledge</td>
<td>Access</td>
<td>Skills Specific to each Möbius skill element.</td>
</tr>
<tr>
<td>Skills</td>
<td>-</td>
<td>Goals Organise goals into STRIDE categories.</td>
</tr>
<tr>
<td>Goals</td>
<td>Objective</td>
<td>Attack Step Specific to each Möbius attack step.</td>
</tr>
<tr>
<td>Attack Step</td>
<td>Limits</td>
<td>- Not captured by Möbius.</td>
</tr>
</tbody>
</table>

Table 3.3: Möbius and Intel Threat Adversary Specifications
The Intel Threat Agent Library defines 22 distinct adversaries, however, there is an element in these classifications that isn’t captured for the MIKEY-SAKKE system. MIKEY-SAKKE is a cryptosystem that will be implemented by security developers and then subsequently used by organisations or other entities for secure communications. The three employee categories: Employee Reckless; Employee Untrained; and Employee Disgruntled didn’t seem to provide this level of distinction between an employee involved in the development and implementation of MIKEY-SAKKE and an employee who is using MIKEY-SAKKE to communicate.

In addition the MIKEY-SAKKE protocol defines the role of a domain manager, who manages the key management server and therefore has a higher level of access than a regular employee (of either kind previously described).

Therefore, I expanded these categories to include employee, user and domain manager each for reckless, untrained and disgruntled. With the definitions for each being:

- **Employee** - An employee of the company/organisation implementing the MIKEY-SAKKE protocol.
- **User** - A user of the MIKEY-SAKKE protocol, as part of their company/organisation communications system.
- **Domain Manager** - A domain manager for a company/organisation using MIKEY-SAKKE as part of their communications system.

The exact adversary definitions evolved with the overall model as new elements were introduced and are described in the next section, 3.2.

### 3.2 Implementation

In this section the implementation of the Möbius model is described including the different goals and attacks added to the model as well as the adversary specifications. I have highlighted the most significant versions to describe in detail but also refer to those in between to show clearly the different steps taken in developing an accurate model of the MIKEY-SAKKE system.

#### 3.2.1 Möbius Model Version 1

The first version of the MIKEY-SAKKE system specification included 4 goals: Read Message; KMS Denial of Service; Impersonate User; and Tamper with IMESSAGE.

In order to achieve the Read Message goal there are a number of different attacks that can be attempted. Figure 3.1 shows an excerpt from the overall Möbius model including the Read Message goal and all attacks that lead to that goal.

The Read Message goal is an information disclosure threat as defined by the STRIDE model. To read the message or communications between the two users then one needs to decrypt the AES communications of an encrypted SRTP stream intercepted from the public network. In order to do this the adversary either needs to possess the traffic encryption key or have the ability to break the AES communications. The traffic encryption key is generated from the TEK generation key (TGK) which is encrypted in the SAKKE data in an IMESSAGE. Therefore the adversary must either break the SAKKE encryption or possess the user’s receiver secret key, (RSK). Figure 3.1 shows a variety of different ways to acquire the user’s RSK including logging into the user’s device; compromising the key management server; or decrypting the HTTP key provision response when sent by the KMS to the user.

These attack paths are not necessarily all achievable but the Möbius model provides a simulation of the adversary behaviour and therefore the adversaries may not behave as may be expected. It is the unexpected behaviour of the adversaries in the simulation that is likely to provide the most insight into the security of the cryptosystem.

The KMS Denial of Service goal can be categorised as a Denial of Service threat. Figure 3.2 shows possible attack paths that an adversary could take and the required access, knowledge and/or
Figure 3.1: Read Message Möbius Model Version 1

Figure 3.2: KMS Denial of Service Möbius Model Version 1
skill elements needed for each attack step. In this version of the model three possible attack paths have been defined:

1. Redirection of communications such that any user trying to contact the KMS would be redirected to a different URL and therefore would be unable to obtain new keys or update certificates.

2. Repeatedly sending HTTP requests such that the KMS would need to process an excessive number of dummy requests and therefore would be unable to effectively process valid requests from users.

3. Destroying the KMS by gaining physical access and destroying the physical server on which the key management server is running.

For the Impersonate User goal, a spoofing threat, the aim of the adversary is to send an I_MESSAGE to a target user in the domain that implies another user in the domain sent it. Therefore the adversary is able to communicate with the target under false pretences. Figure 3.3 shows this requires all the user’s key information as well as the target’s ID; and as described for the Read Message goal the user key information can be acquired in a variety of ways.
The Tamper with I\textsubscript{MESSAGE} goal is a tampering threat and could be achieved by intercepting an encrypted I\textsubscript{MESSAGE} from the public network and changing it in some way before sending it on to the user. Figure 3.4 shows this section of the model.

The adversary specifications were completed using the relationships specified in section 3.1.3. The calculated values for the planning horizon, cost detection and payoff are shown in table 3.4.

<table>
<thead>
<tr>
<th>Number</th>
<th>Adversary</th>
<th>Planning Horizon</th>
<th>Cost</th>
<th>Detection</th>
<th>Payoff</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Employee Reckless</td>
<td>5</td>
<td>0.5</td>
<td>0.3</td>
<td>0.2</td>
</tr>
<tr>
<td>2</td>
<td>User Reckless</td>
<td>3</td>
<td>0.5</td>
<td>0.3</td>
<td>0.2</td>
</tr>
<tr>
<td>3</td>
<td>Employee Untrained</td>
<td>3</td>
<td>0.6</td>
<td>0.2</td>
<td>0.2</td>
</tr>
<tr>
<td>4</td>
<td>User Untrained</td>
<td>2</td>
<td>0.6</td>
<td>0.2</td>
<td>0.2</td>
</tr>
<tr>
<td>5</td>
<td>Info Partner</td>
<td>4</td>
<td>0.4</td>
<td>0.4</td>
<td>0.2</td>
</tr>
<tr>
<td>6</td>
<td>Anarchist</td>
<td>1</td>
<td>0.6</td>
<td>0.2</td>
<td>0.2</td>
</tr>
<tr>
<td>7</td>
<td>Civil Activist</td>
<td>4</td>
<td>0.3</td>
<td>0.5</td>
<td>0.3</td>
</tr>
<tr>
<td>8</td>
<td>Competitor</td>
<td>4</td>
<td>0.2</td>
<td>0.4</td>
<td>0.4</td>
</tr>
<tr>
<td>9</td>
<td>Corrupt Government Official</td>
<td>4</td>
<td>0.1</td>
<td>0.2</td>
<td>0.7</td>
</tr>
<tr>
<td>10</td>
<td>Data Miner</td>
<td>4</td>
<td>0.2</td>
<td>0.4</td>
<td>0.4</td>
</tr>
<tr>
<td>11</td>
<td>Employee Disgruntled</td>
<td>4</td>
<td>0.6</td>
<td>0.2</td>
<td>0.2</td>
</tr>
<tr>
<td>12</td>
<td>User Disgruntled</td>
<td>3</td>
<td>0.6</td>
<td>0.2</td>
<td>0.2</td>
</tr>
<tr>
<td>13</td>
<td>Government Cyberwarrior</td>
<td>4</td>
<td>0.2</td>
<td>0.4</td>
<td>0.4</td>
</tr>
<tr>
<td>14</td>
<td>Government Spy</td>
<td>5</td>
<td>0.0</td>
<td>0.5</td>
<td>0.5</td>
</tr>
<tr>
<td>15</td>
<td>Internal Spy</td>
<td>5</td>
<td>0.2</td>
<td>0.5</td>
<td>0.3</td>
</tr>
<tr>
<td>16</td>
<td>Irrational Individual</td>
<td>1</td>
<td>0.6</td>
<td>0.2</td>
<td>0.2</td>
</tr>
<tr>
<td>17</td>
<td>Legal Adversary</td>
<td>4</td>
<td>0.4</td>
<td>0.3</td>
<td>0.3</td>
</tr>
<tr>
<td>18</td>
<td>Mobster</td>
<td>4</td>
<td>0.2</td>
<td>0.4</td>
<td>0.4</td>
</tr>
<tr>
<td>19</td>
<td>Radical Activist</td>
<td>4</td>
<td>0.4</td>
<td>0.3</td>
<td>0.3</td>
</tr>
<tr>
<td>20</td>
<td>Sensationalist</td>
<td>2</td>
<td>0.4</td>
<td>0.3</td>
<td>0.3</td>
</tr>
<tr>
<td>21</td>
<td>Terrorist</td>
<td>4</td>
<td>0.2</td>
<td>0.5</td>
<td>0.3</td>
</tr>
<tr>
<td>22</td>
<td>Thief</td>
<td>2</td>
<td>0.4</td>
<td>0.4</td>
<td>0.2</td>
</tr>
<tr>
<td>23</td>
<td>Vandal</td>
<td>3</td>
<td>0.4</td>
<td>0.4</td>
<td>0.2</td>
</tr>
<tr>
<td>24</td>
<td>Vendor</td>
<td>4</td>
<td>0.2</td>
<td>0.4</td>
<td>0.4</td>
</tr>
<tr>
<td>25</td>
<td>Domain Manager Reckless</td>
<td>5</td>
<td>0.5</td>
<td>0.3</td>
<td>0.2</td>
</tr>
<tr>
<td>26</td>
<td>Domain Manager Untrained</td>
<td>3</td>
<td>0.6</td>
<td>0.2</td>
<td>0.2</td>
</tr>
<tr>
<td>27</td>
<td>Domain Manager Disgruntled</td>
<td>4</td>
<td>0.6</td>
<td>0.2</td>
<td>0.2</td>
</tr>
</tbody>
</table>

Table 3.4: Adversary Specifications for Möbius Model Version 1

All adversaries are given the public knowledge and access elements listed below:

- Access to Network (the public network over which the MIKEY-SAKKE communications take place).
• KMS Public Key.

• User URI (Uniform Resource Identifier - in the case of MIKEY-SAKKE this is the user’s telephone number, see section 2.1.4).

• KPAK (KMS Public Authentication Key).

• Target URI (Target is another user in the domain for which any gained information, for example the User Secret Signing Key, could be used against).

The KMS Public Key and the KPAK are described as public knowledge but require some action on the part of the adversary in order to acquire them. Specifically the adversary must intercept a HTTP key provision response from the key management server to any user in the domain. This contains both the KMS Public Key and the KPAK in plaintext. These additional actions are not captured as part of the Möbius model in this version, however, the actual implementation is covered in section 4.2.

In addition users, domain managers and the internal spy are given the domain internal information of the KMS URL. This piece of information may be more widely known but for the first version I used a cautious approach.

None of the adversaries were given any of the skill elements, which is unrealistic but I wanted to evaluate the results that could be achieved first before providing the adversaries with more abilities.

Finally the goals were associated with each adversary based on the above categorisations and the outcomes assigned to the adversary in the Intel Threat Agent Library, as described in section 3.1.3.

Once the system and the adversaries were defined I then created a reward model linked to the system model. The reward model defines a number of performance variables which are tracked throughout the simulation and are the metrics used to describe the results. In version 1 I defined 4 performance variables, one for each of the goals, such that the variable returns the probability of the goal being achieved at each point during the simulation.

I defined the simulation to run over an incremental range of 0 to 60 in 5 unit steps and specified the execution time of all the attack steps as 5 units. This was chosen as an initial approach based on the example provided for demonstration of the Möbius tool in using the ADVISE modelling [2, 3].

3.2.2 Möbius Model Version 5

After version 1 I continued to expand and improve the model by including more goals and attack paths as well as introducing specific values for attack step costs and goal payoffs.

Version 2 introduced the User Denial of Service goal, which also comes under the denial of service threat category. The attacks associated with this goal are shown in figure 3.5.

In version 3 I introduced specific cost values to all the attack steps. Within the range 1 to 5 I assigned a value to the attack step based on my judgement of the difficulty and resources required to execute it. In addition I assigned skills to the adversaries by relating each specific skill element to a general skill level. I then gave each adversary the specific skills associated with its skill level and all lower levels.

In version 4 I continued to develop the detail of the model by introducing payoff values for each of the goals specific to each adversary. The payoff values are within the range of 100 to 500 in 100 multiples depending on how important a given goal was to each adversary. I used my judgement of the overall adversary specification to assign these values.

Version 5 expanded the system specification again by including a Create Malicious User goal as shown in figure 3.6. An adversary can create a malicious user if they can acquire a boot key and know the KMS URL; then unique keys can be requested from the KMS and the adversary now is a valid user within the domain. The boot key is provided to the adversary by the domain manager so that the user can be initialised within the domain. This is referred to as bootstrapping however, the exact process is out of scope for both the MIKEY-SAKKE specification and the SecureChorus implementation [112].
Figure 3.5: User Denial of Service Möbius Model Version 5

Figure 3.6: Create Malicious User Möbius Model Version 5
In order to capture these additional goals and knowledge elements in the resultant metrics, I added the performance variables listed below to the reward model. The performance variables are named with either a ‘g’ or a ‘k’ at the start to indicate either a goal or a knowledge element respectively, followed by the name of the element as one word all in lower case.

- g_userdenialofservice
- k_trafficencryptionkey
- k_ksak
- k_userpvt
- k_userdevicepasscode
- k_tekgenerationkey
- k_userssk
- k_kmssecretkey
- k_ranomelement
- k_transportkey
- k_userrsk
- g_createmalicioususer
- k_domainmanagercontact
- k_attackerbootkey
- k_targeturl

I introduced performance variables that tracked the probability of a knowledge element being known at each step in the simulation in order to provide another level of detail to the results and show at what point knowledge elements are achieved that enable the achievement of a goal.

### 3.2.3 Möbius Model Version 7

In conjunction with the low level exploit investigations, described in chapter 4, as specific protocol vulnerabilities were identified I was then able to feed these into the Möbius model.

I was able to identify a possible point of vulnerability in the use of a supersingular curve for the SAKKE encryption, see section 4.1.2. Supersingular curves are particularly susceptible to the MOV attack [77] from which the elliptic curve discrete logarithm problem can be solved.
In versions 6 and 7, I created the Read All Messages goal, see figure 3.7, for which the adversary must solve the SAKKE KMS elliptic curve discrete logarithm problem to obtain the KMS secret key. Using the KMS secret key the adversary can then derive the receiver secret key for any user in that domain and therefore is capable of reading all messages to users in the domain.

This MOV attack, to solve the elliptic curve discrete logarithm problem, requires a high level of skill and significant resources in order to complete the calculation, section 4.2 details the exact results I obtained in trying to implement this attack. So I assigned the Ability to Solve the SAKKE ECDLP to the two skilled government agents: the Government Cyberwarrior, and the Government Spy. I also gave the Read All Messages goal to all adversaries, whom already had the Read Message goal and specified a payoff value of 20 times that of the Read Message payoff value as the adversary can now read the messages for all users.

To capture the Read All Messages goal in the reward model I added another performance variable that returns the probability of the goal being achieved at each point in the simulation. The specifications provided in this section were used in the Möbius model to generate results for each of the different versions. These are given in section 3.3.

### 3.2.4 Möbius Model Version 8

For version 8 of the Möbius model I was able to take information from the low-level exploit investigations and feed those findings back into the high-level analysis model.

I adjusted the Repeatedly Send I_MESSAGES attack step implemented against a specific user, to also require the Ability to Fake From and Create Random Signatures skill element, as shown in figure 3.8. This is due to my research of denial of service attacks against the MIKEY protocol as described in section 4.1.1. I then assigned this skill to specific adversaries with a skill level of 4 or 5. This allows for a more accurate representation of which type of adversary might undertake this attack as well as a more accurate description of exactly how it could be done.

I also included a more accurate representation for how the KMS Public Key and KPAK could be acquired by an adversary. Following discussions with my stakeholders the general feedback
established that even though these knowledge elements are described as public that only means they are transmitted in plaintext as opposed to being displayed on a website for all users etc. Therefore I linked the Link Intercept HTTP Key Prov Response attack step with the KMS Public Key and KPAK knowledge elements, as they are included in plaintext in this response message from the key management server. I also removed the KMS Public Key and KPAK knowledge elements from all the adversaries so that they had to follow the prescribed attack path in order to acquire that information.

Sections 4.1.2 and 4.2 describe the thought experiment conducted for the MOV attack of the SAKKE supersingular elliptic curve. This experiment concludes the infeasibility of this attack against this curve due to large parameter sizes even though as a supersingular curve it is more susceptible to this attack. To represent this new information in the Möbius model I removed the Ability to Break SAKKE ECDLP skill element from all the adversaries it was previously assigned to, however, the goal and the attack steps still stand as if an attack that allowed an adversary to break the SAKKE ECDLP was discovered then this is a valid attack path.

Finally, I introduced the attack described in section 4.1.3 for exploiting the use of a predictable random number generator or even a static value in order to derive the user’s secret signing key. Figure 3.9 shows the introduction of this additional attack path in order to achieve the impersonate user goal. Once a predictable pseudo-random number generator (PRNG) has been identified the difficulty of the attack is low and has a high probability of success. However, identifying a predictable PRNG has a much lower probability of success. Further analysis of the PRNG used in the Secure Chorus implementation could improve the accuracy of these probabilities and is discussed as part of the potential future work in section 5.2.
3.3 Results

3.3.1 Möbius Model Version 1

In version 1 of the Möbius model almost all of the 27 different adversaries failed to achieve any of the goals as they did not have sufficient skills or knowledge to complete the defined attack steps.

However, three adversaries who were able to achieve a goal, namely the KMS Denial of Service goal, were the three variations of the domain manager. These results are shown in table 3.5. This shows that the domain manager adversaries all converge on a mean probability of 0.999 with a confidence interval of 0.002 of achieving the KMS Denial of Service goal by the final stage of the simulation.

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Time</th>
<th>DM Reckless Mean</th>
<th>DM Reckless CI</th>
<th>DM Untrained Mean</th>
<th>DM Untrained CI</th>
<th>DM Disgruntled Mean</th>
<th>DM Disgruntled CI</th>
</tr>
</thead>
<tbody>
<tr>
<td>g_kmsdenialofservice</td>
<td>0</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>g_kmsdenialofservice</td>
<td>5</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>g_kmsdenialofservice</td>
<td>10</td>
<td>0.340</td>
<td>0.029</td>
<td>0.340</td>
<td>0.029</td>
<td>0.340</td>
<td>0.029</td>
</tr>
<tr>
<td>g_kmsdenialofservice</td>
<td>15</td>
<td>0.629</td>
<td>0.030</td>
<td>0.629</td>
<td>0.030</td>
<td>0.629</td>
<td>0.030</td>
</tr>
<tr>
<td>g_kmsdenialofservice</td>
<td>20</td>
<td>0.810</td>
<td>0.024</td>
<td>0.810</td>
<td>0.024</td>
<td>0.810</td>
<td>0.024</td>
</tr>
<tr>
<td>g_kmsdenialofservice</td>
<td>25</td>
<td>0.898</td>
<td>0.019</td>
<td>0.898</td>
<td>0.019</td>
<td>0.898</td>
<td>0.019</td>
</tr>
<tr>
<td>g_kmsdenialofservice</td>
<td>30</td>
<td>0.957</td>
<td>0.013</td>
<td>0.957</td>
<td>0.013</td>
<td>0.957</td>
<td>0.013</td>
</tr>
<tr>
<td>g_kmsdenialofservice</td>
<td>35</td>
<td>0.979</td>
<td>0.009</td>
<td>0.979</td>
<td>0.009</td>
<td>0.979</td>
<td>0.009</td>
</tr>
<tr>
<td>g_kmsdenialofservice</td>
<td>40</td>
<td>0.992</td>
<td>0.006</td>
<td>0.992</td>
<td>0.006</td>
<td>0.992</td>
<td>0.006</td>
</tr>
<tr>
<td>g_kmsdenialofservice</td>
<td>45</td>
<td>0.995</td>
<td>0.004</td>
<td>0.995</td>
<td>0.004</td>
<td>0.995</td>
<td>0.004</td>
</tr>
<tr>
<td>g_kmsdenialofservice</td>
<td>50</td>
<td>0.999</td>
<td>0.002</td>
<td>0.999</td>
<td>0.002</td>
<td>0.999</td>
<td>0.002</td>
</tr>
<tr>
<td>g_kmsdenialofservice</td>
<td>55</td>
<td>0.999</td>
<td>0.002</td>
<td>0.999</td>
<td>0.002</td>
<td>0.999</td>
<td>0.002</td>
</tr>
<tr>
<td>g_kmsdenialofservice</td>
<td>60</td>
<td>0.999</td>
<td>0.002</td>
<td>0.999</td>
<td>0.002</td>
<td>0.999</td>
<td>0.002</td>
</tr>
</tbody>
</table>

Note: DM - Domain Manager; CI - Confidence Interval; g_kmsdenialofservice is the denial of service goal for the KMS

Table 3.5: Results from Möbius Model Version 1

Analysis of the simulation traces for the domain manager adversaries showed that all three used the fact that they had access to the key management server to implement the attack step to destroy the key management server.

3.3.2 Möbius Model Version 5

The introduction of cost values to the attack steps in version 3 resulted in none of the adversaries attempting any attacks as the cost to payoff ratio was too small, so the cost averse domain manager roles now had no incentive to attempt the KMS Denial of Service goal as they had previously. Version 4 significantly increased the cost to payoff ratio and with the introduction of a number of new attacks in version 5 some interesting results were produced.

Table 3.6 shows the Corrupt Government Official adversary achieving the KMS denial of service goal with a 0.999 mean probability and a 0.002 confidence interval. Analysis of the simulation traces shows the Corrupt Government Official adversary uses the Redirect HTTP Communications attack to achieve its goal.

In addition, table 3.7 shows the Government Spy adversary having a 0.563 probability of being able to read the encrypted message after 60 time units within a confidence interval of 0.010. The knowledge items achieved leading up to this final goal are also shown. The simulation trace for the Government Spy shows he takes the following route of attack:

1. DeriveUserID - from User URI, which is public.
2. DeriveTargetID - from Target URI, which again is public.
<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Time</th>
<th>Corrupt Government Official Mean</th>
<th>Confidence Interval</th>
</tr>
</thead>
<tbody>
<tr>
<td>g_kmsdenialofservice</td>
<td>0</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>g_kmsdenialofservice</td>
<td>5</td>
<td>0.491</td>
<td>0.031</td>
</tr>
<tr>
<td>g_kmsdenialofservice</td>
<td>10</td>
<td>0.741</td>
<td>0.027</td>
</tr>
<tr>
<td>g_kmsdenialofservice</td>
<td>15</td>
<td>0.863</td>
<td>0.021</td>
</tr>
<tr>
<td>g_kmsdenialofservice</td>
<td>20</td>
<td>0.938</td>
<td>0.015</td>
</tr>
<tr>
<td>g_kmsdenialofservice</td>
<td>25</td>
<td>0.964</td>
<td>0.012</td>
</tr>
<tr>
<td>g_kmsdenialofservice</td>
<td>30</td>
<td>0.985</td>
<td>0.008</td>
</tr>
<tr>
<td>g_kmsdenialofservice</td>
<td>35</td>
<td>0.996</td>
<td>0.004</td>
</tr>
<tr>
<td>g_kmsdenialofservice</td>
<td>40</td>
<td>0.997</td>
<td>0.003</td>
</tr>
<tr>
<td>g_kmsdenialofservice</td>
<td>45</td>
<td>0.999</td>
<td>0.002</td>
</tr>
<tr>
<td>g_kmsdenialofservice</td>
<td>50</td>
<td>0.999</td>
<td>0.002</td>
</tr>
<tr>
<td>g_kmsdenialofservice</td>
<td>55</td>
<td>0.999</td>
<td>0.002</td>
</tr>
<tr>
<td>g_kmsdenialofservice</td>
<td>60</td>
<td>0.999</td>
<td>0.002</td>
</tr>
</tbody>
</table>

Note: g_kmsdenialofservice is the denial of service goal for the KMS.

Table 3.6: Results from Möbius Model Version 5 (1)

<table>
<thead>
<tr>
<th>Time</th>
<th>g_readmessage</th>
<th>k_tek</th>
<th>k_userpvt</th>
<th>k_tgk</th>
<th>k_userssk</th>
<th>k_userrsk</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>5</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>10</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>15</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>20</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>25</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>30</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>35</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>40</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>45</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>50</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>55</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>60</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
</tbody>
</table>

Note: CI - Confidence Interval; g_readmessage is the Read Message goal; k represents a knowledge item; tek - traffic encryption key; userpvt - user PVT; tgk - TEK generation key; userssk - user SSK; userrsk - user RSK

Table 3.7: Results from Möbius Model Version 5 (2)

4. InterceptMESSAGE - from the public network.
5. DecryptSAKKEData - from the intercepted MESSAGE, using the user SSK obtained when the user device was hacked.
6. GenerateTEK - from SSV also known as TGK obtained from the decrypted SAKKE data.
7. InterceptSRTPStream - from the public network.
8. DecryptAESCommunication - in the SRTP stream using the previously generated TEK.

The initial steps, such as deriving IDs, are relatively easy and so have a high probability of success and a low probability of detection. However, the later steps are more difficult and therefore are often repeated during the simulation until the adversary achieves a successful result. This explains the low final mean value for the g_readmessage variable as with the limited time of the simulation the adversary will not always achieve its final goal.

### 3.3.3 Möbius Model Version 7

As in Version 5 the Corrupt Government Official adversary still achieves the KMS denial of service goal with a 0.999 mean probability and a 0.002 confidence interval.

The Government Spy now has the ability to break the SAKKE elliptic curve discrete logarithm problem and a goal to read all the messages in the domain with a very large pay off value. So the adversary no longer attempts the read message goal but instead chooses to attack the key management server SAKKE elliptic curve discrete logarithm problem. The Government Cyberwarrior has the same skill and associated goal, in addition the cost, detection and payoff priorities have also been adjusted such that the adversary is less cost averse.

Both adversaries execute the attack to solve the SAKKE elliptic curve discrete logarithm problem and derive the KMS secret key and the results are shown in table 3.8.

<table>
<thead>
<tr>
<th>Time</th>
<th>g_readallmessages Mean</th>
<th>g_readallmessages CI</th>
<th>k_kmssecretkey Mean</th>
<th>k_kmssecretkey CI</th>
<th>k_randomelement Mean</th>
<th>k_randomelement CI</th>
<th>k_userrsk Mean</th>
<th>k_userrsk CI</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>5</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>10</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.464</td>
<td>0.018</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>15</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.739</td>
<td>0.016</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>20</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.869</td>
<td>0.012</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>25</td>
<td>0.197</td>
<td>0.014</td>
<td>0.197</td>
<td>0.014</td>
<td>0.935</td>
<td>0.009</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>30</td>
<td>0.473</td>
<td>0.018</td>
<td>0.473</td>
<td>0.018</td>
<td>0.969</td>
<td>0.006</td>
<td>0.140</td>
<td>0.012</td>
</tr>
<tr>
<td>35</td>
<td>0.684</td>
<td>0.017</td>
<td>0.684</td>
<td>0.017</td>
<td>0.985</td>
<td>0.004</td>
<td>0.372</td>
<td>0.017</td>
</tr>
<tr>
<td>40</td>
<td>0.820</td>
<td>0.014</td>
<td>0.820</td>
<td>0.014</td>
<td>0.993</td>
<td>0.003</td>
<td>0.588</td>
<td>0.018</td>
</tr>
<tr>
<td>45</td>
<td>0.908</td>
<td>0.010</td>
<td>0.908</td>
<td>0.010</td>
<td>0.998</td>
<td>0.002</td>
<td>0.752</td>
<td>0.015</td>
</tr>
<tr>
<td>50</td>
<td>0.953</td>
<td>0.008</td>
<td>0.953</td>
<td>0.008</td>
<td>0.999</td>
<td>0.001</td>
<td>0.865</td>
<td>0.012</td>
</tr>
<tr>
<td>55</td>
<td>0.974</td>
<td>0.006</td>
<td>0.974</td>
<td>0.006</td>
<td>1.000</td>
<td>0.001</td>
<td>0.927</td>
<td>0.009</td>
</tr>
<tr>
<td>60</td>
<td>0.987</td>
<td>0.004</td>
<td>0.987</td>
<td>0.004</td>
<td>1.000</td>
<td>0.001</td>
<td>0.961</td>
<td>0.007</td>
</tr>
</tbody>
</table>

Note: CI - Confidence Interval; g_readallmessages is the Read All Messages goal; k represents a knowledge item; kmssecretkey - KMS secret key; randomelement - random element; userrsk - user RSK

Table 3.8: Results from Möbius Model Version 7

At the end of the simulation both government agents have the probability of achieving the Read All Messages goal of 0.987 with a 0.004 confidence interval. The other columns in table 3.8 show how the adversary also attempts to derive the specific user RSK using the KMS secret key. Also the adversaries guess the random element which can be used in the derivation of the user secret signing key, although that derivation is not attempted in this simulation.

### 3.3.4 Möbius Model Version 8

As described in section 3.2.4, version 8 of the Möbius model removes the Ability to Break SAKKE ECDLP skill element from all adversaries and as such the results show that none of the adversaries now attempt this attack as they had done in version 7.
In addition, version 8 introduces an attack involving the exploitation of a predictable pseudo-random number generator to derive the user’s secret signing key and thus impersonate the user. However, due to the very low probability of success assigned to the Identify Predictable PRNG attack step, no adversary attempts this attack either.

As in previous versions, the Corrupt Government Official adversary achieves the KMS denial of service goal by redirecting the HTTP communications. The Government Cyberwarrior now also achieves this goal with a 1.000 probability and 0.000 confidence interval by the same method of redirecting the HTTP communications. The results for both of these attacks are shown in Table 3.9.

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Time</th>
<th>Corrupt Government Official</th>
<th>Government Cyberwarrior</th>
</tr>
</thead>
<tbody>
<tr>
<td>g_kmsdenialofservice</td>
<td>0</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>g_kmsdenialofservice</td>
<td>5</td>
<td>0.491</td>
<td>0.489</td>
</tr>
<tr>
<td>g_kmsdenialofservice</td>
<td>10</td>
<td>0.741</td>
<td>0.744</td>
</tr>
<tr>
<td>g_kmsdenialofservice</td>
<td>15</td>
<td>0.863</td>
<td>0.868</td>
</tr>
<tr>
<td>g_kmsdenialofservice</td>
<td>20</td>
<td>0.938</td>
<td>0.935</td>
</tr>
<tr>
<td>g_kmsdenialofservice</td>
<td>25</td>
<td>0.964</td>
<td>0.966</td>
</tr>
<tr>
<td>g_kmsdenialofservice</td>
<td>30</td>
<td>0.985</td>
<td>0.983</td>
</tr>
<tr>
<td>g_kmsdenialofservice</td>
<td>35</td>
<td>0.996</td>
<td>0.992</td>
</tr>
<tr>
<td>g_kmsdenialofservice</td>
<td>40</td>
<td>0.997</td>
<td>0.996</td>
</tr>
<tr>
<td>g_kmsdenialofservice</td>
<td>45</td>
<td>0.999</td>
<td>0.998</td>
</tr>
<tr>
<td>g_kmsdenialofservice</td>
<td>50</td>
<td>0.999</td>
<td>0.999</td>
</tr>
<tr>
<td>g_kmsdenialofservice</td>
<td>55</td>
<td>0.999</td>
<td>1.000</td>
</tr>
<tr>
<td>g_kmsdenialofservice</td>
<td>60</td>
<td>0.999</td>
<td>1.000</td>
</tr>
</tbody>
</table>

Note: g_kmsdenialofservice is the denial of service goal for the KMS

Table 3.9: Results from Möbius Model Version 8 (1)

The Government Cyberwarrior also attempts some further attack steps in order to achieve knowledge elements but is unable to achieve any further goals. Table 3.10 shows the Government Cyberwarrior acquiring the random element through guesswork; and the KPAK and the KMS Public Key by interception of a HTTP key provision response from the key management server.

The Government Spy is the third and final adversary to attempt any attacks in version 8 of the model. Table 3.11 shows the adversary acquiring different knowledge elements in order to ultimately achieve the read message goal with a probability of 0.353 and a confidence interval of 0.009. The Government Spy uses the same attack path as described in version 5 of the model. As the adversary no longer has the more beneficial attack of solving the SAKKE elliptic curve discrete logarithm problem, he has reverted back to hacking the user’s device in order to acquire the user’s key information. The adversary also intercepts a HTTP key provision response from the key management server in order to gain the KPAK and the KMS Public Key.

### 3.4 Evaluation

From versions 1 to 8 of the Möbius ADVISE model, the representation of the MIKEY-SAKKE protocol; its environment; and the adversaries evolve to show the likelihood of different attack paths and factors that facilitate different attacks.

In version 1 only the domain manager adversaries attempted and achieved any attacks as they had a higher level of knowledge and access. However, the introduction of values for the cost of each of the attack steps quickly showed the domain managers as cost averse characters and were actually very unlikely to destroy the key management server even though they have access to it.

Version 5 showed two new attack attempts. The Corrupt Government Official redirecting the HTTP communications of the key management server in order to achieve denial of service; and the
Government Spy attempting to read the secure communications between users by hacking into the user’s device.

The Corrupt Government Official is able to achieve the KMS denial of service goal with a 0.999 probability and 0.002 confidence interval consistently through the remaining versions. However, if we consider the nature of the MIKEY-SAKKE protocol this actually is unlikely to have any significant impact on users as long as sufficient detection steps are in place. Users are able to establish secure communications with each other without any communication with the KMS once they have had their keys provided. The recommendation is for key provision to occur on a monthly basis so the Corrupt Government Official would need to deny service at the exact point of change over for this to have a significant impact on users. Furthermore, sharing next month’s keys with users in the domain a few days in advance would again limit the impact of this attack. Finally, a good detection mechanism to ensure communications to the key management server are being directed correctly would also be a good preventative measure against this attack.

The Government Spy, however, as a highly skilled adversary with large resources attempts to read secure communications between the users and potentially is undetected. There is a low probability of success for this attack, however, repeated attempts would increase the overall likelihood of success and as this adversary is apathetic to cost then he is very likely to continue trying until he succeeds. However, this is only potentially undetectable as the exact method for hacking into a user’s device and acquiring the key information is not specified. Further investigation into the exact methods that could be used is required as well as the possible preventative measures that could be used on the device in order to determine the likelihood of detection and overall success.

In version 7, an attack based on the MOV attack against the SAKKE elliptic curve, see section 4.1.2, is captured and with the ability to break the SAKKE elliptic curve discrete logarithm problem then the Government Cyberwarrior and the Government Spy are both able to execute this attack and read all the secure communications to users in the domain. This attack has very serious consequences for the users of the MIKEY-SAKKE protocol as the adversary can now read all the ‘secure’ communications and remain completely undetected. However, as established in section 4.2, this attack is not practical to implement on the SAKKE elliptic curve and is not likely to be any time soon. Therefore in version 8 this skill was removed from both of these adversaries whom reverted to attempting different attacks.

The iterative nature in which I have described the development of the Möbius ADVISE model
for MIKEY-SAKKE shows the iterative nature of the approach I took to the security analysis. Using the findings from the low level exploit investigations, described in chapter 4, each iteration of the model can be improved in terms of accuracy of the probabilities of the different attack steps; and in terms of completeness in the overall description of the MIKEY-SAKKE protocol and all possible attack paths.
<table>
<thead>
<tr>
<th>Time</th>
<th>$g_{readmessage}$ Mean</th>
<th>$k_{tek}$ Mean</th>
<th>$k_{userpvt}$ Mean</th>
<th>$k_{tgk}$ Mean</th>
<th>$k_{userssk}$ Mean</th>
<th>$k_{random}$ Mean</th>
<th>$k_{userrsk}$ Mean</th>
<th>$k_{kpak}$ Mean</th>
<th>$k_{kmspublickey}$ Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>5</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>10</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>15</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>20</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>25</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>30</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>35</td>
<td>0.000</td>
<td>0.000</td>
<td>0.069</td>
<td>0.005</td>
<td>0.955</td>
<td>0.004</td>
<td>0.362</td>
<td>0.009</td>
<td>0.955</td>
</tr>
<tr>
<td>40</td>
<td>0.000</td>
<td>0.000</td>
<td>0.216</td>
<td>0.008</td>
<td>0.984</td>
<td>0.002</td>
<td>0.574</td>
<td>0.010</td>
<td>0.984</td>
</tr>
<tr>
<td>45</td>
<td>0.025</td>
<td>0.003</td>
<td>0.396</td>
<td>0.010</td>
<td>0.995</td>
<td>0.001</td>
<td>0.737</td>
<td>0.009</td>
<td>0.995</td>
</tr>
<tr>
<td>50</td>
<td>0.096</td>
<td>0.006</td>
<td>0.565</td>
<td>0.010</td>
<td>0.998</td>
<td>0.001</td>
<td>0.850</td>
<td>0.007</td>
<td>0.998</td>
</tr>
<tr>
<td>55</td>
<td>0.214</td>
<td>0.008</td>
<td>0.711</td>
<td>0.009</td>
<td>0.999</td>
<td>0.000</td>
<td>0.917</td>
<td>0.005</td>
<td>0.999</td>
</tr>
<tr>
<td>60</td>
<td>0.353</td>
<td>0.009</td>
<td>0.814</td>
<td>0.008</td>
<td>1.000</td>
<td>0.000</td>
<td>0.953</td>
<td>0.004</td>
<td>1.000</td>
</tr>
</tbody>
</table>

Note: CI - Confidence Interval; $g_{readmessage}$ is the Read Message goal; $k$ represents a knowledge item; $tek$ - traffic encryption key; $userpvt$ - user PVT; $tgk$ - TEK generation key; $userssk$ - user SSK; $random$ - random element; $userrsk$ - user RSK; $kpak$ - KPAK; $kmspublickey$ - KMS Public Key

Table 3.11: Results from Möbius Model Version 8 (3)
Chapter 4

Low level Exploit Investigation

4.1 Theory

4.1.1 MIKEY

Analysis of the MIKEY protocol has highlighted a number of vulnerabilities [6, 58, 42]. I have reviewed these with reference to the MIKEY-SAKKE protocol to provide a theoretical analysis of whether or not MIKEY-SAKKE may be vulnerable to these weaknesses as well.

The man-in-the-middle attack involves an adversary, Eve, intercepting the communications between two users, Alice and Bob. Eve pretends to be Alice when communicating with Bob; and pretends to be Bob to Alice, therefore she can read all the communications between Alice and Bob. MIKEY-DH, the MIKEY key management protocol with the Diffie-Hellman key exchange [36], is vulnerable to a man-in-the-middle attack as there is no authentication provided by this protocol. Therefore there is no way for Alice to verify that she is actually talking to Bob and vice-versa. The MIKEY-SAKKE protocol uses the ECCSI authentication protocol and so is not exposed to this same vulnerability.

Denial of service attacks against MIKEY in DH mode or RSA mode are described by [42]. This can be done by flooding the victim with session invite requests with a faked From header and random data that looks like a signature in the place of a valid one. The victim will need to process each signature to verify whether or not it is valid and therefore performing a large number of operations denying the valid service.

MIKEY-SAKKE uses the ECCSI algorithm as its digital signature and so in theory is also vulnerable to this user denial of service attack. Section 3.2.2 shows the high-level analysis of this type of attack incorporated into the Möbius model for the MIKEY-SAKKE protocol.

An evaluation based on this theory is given in section 4.3.

4.1.2 SAKKE

The use of elliptic curves and so the elliptic curve discrete logarithm problem is the base of the security for the SAKKE protocol. As part of my specific investigation into the SAKKE protocol I first started with a review of the basics of elliptic curve mathematics. Table 4.1 details the notation used in this section.

An elliptic curve is defined over a finite field, examples of which are given in appendix I. For this section I focus on those defined over a prime finite field, \( F_p \), where \( p \) is a large prime number. There are a number of different ways an elliptic curve equation can be defined, which are shown in table 4.2.

For use in cryptography the following additional parameters are defined for an elliptic curve:

- \( q \) is a second large prime number, which gives the order of the generating point or base point
- \( P \) is the generating point or base point of the elliptic curve, which has an order \( q \).
Table 4.1: Elliptic Curve Notation

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( p )</td>
<td>a large prime integer</td>
</tr>
<tr>
<td>( F_p )</td>
<td>a finite field with ( p ) elements</td>
</tr>
<tr>
<td>( F_p^* )</td>
<td>a multiplicative group of the finite field ( F_p )</td>
</tr>
<tr>
<td>( E(F_p) )</td>
<td>an elliptic curve over the finite field ( F_p ), where ( E : y^2 = x^3 + ax + b )</td>
</tr>
<tr>
<td>( #E(F_p) )</td>
<td>the number of points of the elliptic curve ( E(F_p) )</td>
</tr>
<tr>
<td>( q )</td>
<td>a large odd prime integer that divides ( p + 1 )</td>
</tr>
<tr>
<td>( P )</td>
<td>a point on ( E(F_p) ) that generates the cyclic subgroup of order ( q )</td>
</tr>
<tr>
<td>( Q = [m]P )</td>
<td>elliptic curve point multiplication such that ( Q ) is the result of adding ( P ) to itself ( m ) times</td>
</tr>
<tr>
<td>( O )</td>
<td>the infinite point on the elliptic curve</td>
</tr>
<tr>
<td>point order</td>
<td>point ( X ) is of order ( y ) if ( y ) is the smallest value such that ( [y]X = O )</td>
</tr>
<tr>
<td>( j(E) )</td>
<td>the ( j )-invariant of the the elliptic curve ( E ), where ( j = 1728(4a^3/(4a^3 + 27b^2)) )</td>
</tr>
<tr>
<td>( k )</td>
<td>the embedding degree, where ( p^k = 1 \mod q )</td>
</tr>
<tr>
<td>( t )</td>
<td>the trace of the curve such that ( t = p + 1 - #E(F_p) )</td>
</tr>
<tr>
<td>( D )</td>
<td>the complex-multiplication field discriminant defined as ((t^2 - 4p)/s^2) if ((t^2 - 4p)/s^2) mod 4 = 1, otherwise as (4(t^2 - 4p)/s^2).</td>
</tr>
</tbody>
</table>

Table 4.2: Elliptic Curve Equations

<table>
<thead>
<tr>
<th>Name</th>
<th>Equation</th>
<th>Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>short-Weierstrass equation</td>
<td>( E : y^2 = x^3 + ax + b )</td>
<td>( 4a^3 + 27b^2 \neq 0 ) in ( F_p )</td>
</tr>
<tr>
<td>Montgomery equation</td>
<td>( E : y^2 = x^3 + Ax^2 + x )</td>
<td>( B(A^2 - 4) \neq 0 ) in ( F_p )</td>
</tr>
<tr>
<td>Edwards equation</td>
<td>( E : x^2 + y^2 = 1 + dx^2y^2 )</td>
<td>( d(1-d) \neq 0 ) in ( F_p )</td>
</tr>
</tbody>
</table>

The SAKKE protocol [54], specifies the use of a supersingular elliptic curve, \( E \) over a finite field \( F_p \) of the form
\[
E : y^2 = x^3 - 3x
\] (4.1)
where \( p \) and \( q \) are 1024 bit prime numbers and \( P \) is a point on the curve, \( E \) of order \( q \). The exact values for these parameters are given in [54].

A curve is described as supersingular if either:
\[
p = 2 \text{ or } 3, \quad \text{and} \quad j(E) = 0
\] (4.2)
or
\[
p \geq 5, \quad \text{and} \quad t = 0.
\] (4.3)

The SAKKE curve is supersingular as it meets the second set of criteria, eqn. 4.3, as \( p \geq 5 \) and \( \#E(F_p) = p + 1 \) such that the trace, \( t = 0 \).

The elliptic curve discrete logarithm problem is an alternative hard problem on which the security of a cryptographic protocol can be based. In contrast, the security of the RSA (Rivest-Shamir-Adleman) protocol relies on the discrete logarithm problem as the base of its security. However, there are a number of algorithms that have been developed including the Number Field Sieve [94] and Index Calculus [123] that increase the efficiency in solving this problem to a sub-exponential order. Therefore, the security of cryptographic protocols relying on the factorisation problem to guarantee security is impacted as the gap between the difficulty of factoring and the difficulty of multiplying diminishes. There are currently no algorithms that improve on the exponential approach to solve the elliptic curve discrete logarithm problem, thus providing a much more difficult problem on which to base the security of a system.
The level of security provided by a curve depends on a number of different factors in order to ensure that the elliptic curve discrete logarithm problem is sufficiently difficult as well as to prevent against attacks that use certain elliptic curve properties in order to bypass the elliptic curve discrete logarithm problem altogether.

“SafeCurves: choosing safe curves for elliptic-curve cryptography” [12] is a website by Daniel J. Bernstein and Tanja Lange that provides a list of criteria specifying the conditions they have identified in order for an elliptic curve to be safe. As quoted on the SafeCurves website “The SafeCurves criteria are designed to ensure ECC security, not just ECDLP security.”[12] These criteria are captured in table 4.3

SafeCurves applies the specified criteria to a range of different curves designed by government agencies as well as by independent researchers. The SAKKE elliptic curve is not included in this analysis but I have selected a few specific areas to investigate for this curve.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Curve Parameters</strong></td>
<td></td>
</tr>
<tr>
<td>Field</td>
<td>The finite field $F_p$ must be prime.</td>
</tr>
<tr>
<td>Equation</td>
<td>Elliptic curves, $E$ over $F_p$ must be expressed by one of the equation forms given in table 4.2.</td>
</tr>
<tr>
<td>Base Point</td>
<td>The base point $P$ must be on the curve, $E$.</td>
</tr>
<tr>
<td><strong>ECDLP Security</strong></td>
<td></td>
</tr>
<tr>
<td>Rho</td>
<td>The Rho attack [92], takes $0.886\sqrt{q}$ additions; and this value must be greater than $2^{100}$ to be safe.</td>
</tr>
<tr>
<td>Transfer</td>
<td>A transfer converts the ECDLP into a DLP over a finite field as shown by the MOV [77] and Smart-ASS [114, 116] attacks. To be safe the embedding degree, $k$ must be greater than $(q - 1)/100$.</td>
</tr>
<tr>
<td>Discriminant</td>
<td>The absolute value for the complex-multiplication field discriminant $</td>
</tr>
</tbody>
</table>
| Rigidity        | Rigidity is a feature of the curve generation process, categorised by [12]:  

- **Fully rigid** - Completely explained.  
- **Somewhat rigid** - Not completely explained, but unexplained parts do not give the curve generators many bits of control.  
- **Manipulatable** - Large unexplained input, giving the curve generator a large space of curves to choose from.  
- **Trivially manipulatable** - Large unexplained input, giving the curve generator a large space of curves to choose from, and there is an efficient method to work backwards from a specified curve in this space to the input. |
| **ECC Security** |             |
| Ladders         | Scalar multiplication is a key operation used in ECC and is implemented with ladders. The Montgomery ladder [84] runs in constant time and is required to be safe. |
| Twists          | Twist secure curves prevent against small-subgroup attacks and invalid curve attacks. |
| Completeness    | To be safe a curve must be complete for single and multi scalar multiplication formulas such that there are no failure cases that can be exploited. |
| Indistinguishable| Cryptographic protocols require keys that are indistinguishable from random data [24]. Curves must support a bijective map that represents elliptic curve points as indistinguishable random strings. |

Table 4.3: SafeCurves Criteria [12]

The rho method [93], as mentioned in table 4.3, can be used to solve the elliptic curve discrete logarithm problem. For the elliptic curve cyclic subgroup $E(F_p)[q]$ the rho method runs in order
O(√q) time. The SafeCurve’s requirement for an elliptic curve to be safe from this attack is for the cost of the attack, defined as 0.886√q, to be greater than 2^{100}. The cost of the rho method on the SAKKE elliptic curve is at least 2^{511}, see appendix F, which surpasses the SafeCurve’s requirement.

The SAKKE curve is supersingular and this type of curve is widely described as weak and unsuitable for use in cryptography [19, 60]. Supersingular elliptic curves are particularly vulnerable to the MOV attack [77].

The MOV attack [77] transfers the elliptic curve discrete logarithm problem onto a discrete logarithm problem in the multiplicative group $\mathbb{F}_{p^k}^*$. This discrete logarithm problem can then be solved using one of the sub-exponential algorithms described in [94], [123] and [101]. A supersingular curve is particular vulnerable to the MOV attack as the embedding degree is limited to $k \leq 6$ thus limiting the size of the multiplicative group.

In order to implement the MOV attack, the following steps can be followed:

1. $Q = [m]S$ is the ECDLP over the curve $E(\mathbb{F}_p)$ where $Q$ and $S$ are points on the curve and $m$ is an integer value
2. Choose a random point $T$ in $E(\mathbb{F}_p)$, that is not equal to $S$, $Q$ or $O$
3. Check $T$ has order $q$, otherwise choose a different random point
4. Compute the Weil pairing values using Miller’s Algorithm [79, 80]:
   - $\alpha = e_q(S, T)$ in $\mathbb{F}_{p^k}^*$
   - $\beta = e_q(Q, T)$ in $\mathbb{F}_{p^k}^*$
5. Solve the DLP for $\alpha$ and $\beta$ in $\mathbb{F}_{p^k}^*$ as $\beta = \alpha^m$
6. The value for $m$ now also solves the ECDLP for $Q = [m]S$

Section 4.2 describes the detailed thought experiment I implemented for the MOV attack on the SAKKE supersingular elliptic curve.

Elliptic curve points are also easily distinguishable from uniform random strings and with some analysis the exact curve being used for the protocol can often be identified [14]. This provides difficulties in the case where censorship circumvention is required by the protocol.

Section 4.3 provides an evaluation of the security provided by the SAKKE protocol.

### 4.1.3 ECCSI

ECCSI uses a different curve to the SAKKE protocol, namely the NIST P-256 curve [75]. The notation used to describe this curve and the related ECCSI investigations is given in table 4.4.

ECCSI uses the NIST P-256 elliptic curve, $E$, defined over the prime field $\mathbb{F}_p$ and having a subgroup of order $q$. $E$ is defined using the short-Weierstrauss equation as [75]

$$y^2 = x^3 - 3x + b$$  \hspace{1cm} (4.4)

where $b = 41058363725152142129326129780047268409114441015993725554835256314039467401291$

SafeCurves [12], as discussed in section 4.1.2, provides criteria that define safe elliptic curves for use in cryptography. The NIST P-256 curve is analysed with these criteria by the SafeCurves team and table 4.5 shows the results.

From the analysis of NIST P-256 by the SafeCurves, there are four different areas of weakness that have been identified: rigidity, ladders, completeness, and indistinguishability.

SafeCurves defines rigidity as “a feature of a curve-generation process, limiting the number of curves that can be generated by the process. The attacker succeeds only if some curve in this limited set is vulnerable to the secret attack. For comparison, without rigidity, the attacker can freely generate curves until finding a curve vulnerable to the secret attack” [12]. The NIST P-256 curve is described as manipulatable as there is a large unexplained seed value used in the definition of the curve. [75] in section A.3.3.2 describes the algorithm for how to generate an elliptic curve over
Symbol | Description
--- | ---
p | a large prime integer

$F_p$ | a finite field with $p$ elements

$E(F_p)$ | an elliptic curve over the finite field $F_p$

$q$ | a large odd prime integer that divides $p + 1$

$P$ | a point on $E(F_p)$ that generates the cyclic subgroup of order $q$

$Q = [m]P$ | elliptic curve point multiplication such that $Q$ is the result of adding $P$ to itself $m$ times

$k$ | the embedding degree, where $p^k = 1 \mod q$

t | the trace of the curve such that $t = p + 1 - \#E(F_p)$

$D$ | the complex-multiplication field discriminant defined as $(t^2 - 4p)/s^2$ if $(t^2 - 4p)/s^2 \mod 4 = 1$, otherwise as $4(t^2 - 4p)/s^2$.

$RSK$ | a user’s receiver secret key

$(J_x, J_y)$ | the affine coordinates for the elliptic curve point $J$

$j$ | a random variable

$(r, s)$ | a signature

e | a hash value computed over the message being sent (see section 2.1.6 for more details)

$(X)_x$ | is the $x$ affine coordinate of the elliptic curve point $X$

---

Table 4.4: ECCSI Notation

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Curve Parameters</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Field</td>
<td>True</td>
<td>Finite field $F_p$ is prime.</td>
</tr>
<tr>
<td>Equation</td>
<td>True</td>
<td>$E(F_p)$ is expressed by the short-Weierstrauss equation.</td>
</tr>
<tr>
<td>Base Point</td>
<td>True</td>
<td>The Base Point $P$ is on the curve, $E$.</td>
</tr>
<tr>
<td><strong>ECDLP Security</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rho</td>
<td>True</td>
<td>The cost of the Rho attack, $0.886\sqrt{q}$ is greater than $2^{100}$.</td>
</tr>
<tr>
<td>Transfer</td>
<td>True</td>
<td>The embedding degree, $k$ is greater than $(q - 1)/100$.</td>
</tr>
<tr>
<td>Discriminant</td>
<td>True</td>
<td>The absolute value for the complex-multiplication field discriminant $</td>
</tr>
<tr>
<td>Rigidity</td>
<td>False</td>
<td><em>Manipulatable</em> as there is a large unexplained seed input value.</td>
</tr>
<tr>
<td><strong>ECC Security</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ladders</td>
<td>False</td>
<td>The Montgomery ladder [84] is not supported.</td>
</tr>
<tr>
<td>Twists</td>
<td>True</td>
<td><em>stuff here that makes better sense</em></td>
</tr>
<tr>
<td>Completeness</td>
<td>False</td>
<td>Not complete for either single or multi scalar multiplication.</td>
</tr>
<tr>
<td>Indistinguishable</td>
<td>False</td>
<td>Does not support a bijective map.</td>
</tr>
</tbody>
</table>

Table 4.5: NIST P-256 Analysis by SafeCurves [12]

$F_p$, where the seed value is chosen arbitrarily and used to calculate the parameters for the elliptic curve. Without further details for the unexplained seed value there is little further investigation that can be done.

Scalar multiplication, i.e. $Q = [m]P$, is one of the key operations when using elliptic curves in cryptography. Ladders are used to implement these operations and must be simple, fast and run in constant time in order to avoid conflicts between simplicity, efficiency and security [12]. SafeCurves specifies the requirement to support the Montgomery ladder which the NIST P-256 curve does not. SecureChorus uses the OpenSSL implementation of elliptic curves. OpenSSL’s ladder implementation for the NIST B-163 curve over a binary finite field $F_{2^{w}}$ was shown to be
Analysing MIKEY-SAKKE 4

September, 2015

vulnerable to timing attacks [23].

Completeness requires there to be no failure cases in the ladder implementation for the elliptic curve single or multi scalar implementation. The possible failure cases often arise for addition formulas when the mathematical definition for elliptic curve point scalar multiplication are implemented directly. By triggering these failure cases an adversary can gain information about the protocol as demonstrated by [61].

As discussed in section 4.1.2 for the SAKKE supersingular curve, elliptic curve points are easily distinguishable and therefore cause difficulties if censorship circumvention is required for the protocol.

Finally, in section 2.3.3 I described two attacks on the ECDSA protocol that exploited a lack of randomness in order to derive the user’s secret signing key.

For the ECDSA algorithm a user signing a message calculates the signature \((r, s)\) by

\[
\begin{align*}
    r &= ((jP)_x, \quad (4.5) \\
    s &= \frac{e + zr}{j} \quad (4.6)
\end{align*}
\]

where in this case the user’s RSK is represented by the letter \(z\).

The failoverflow team exploited the use of a static random variable, \(j\), in order to derive a value for the user’s receiver secret key \(z\) from public information [4], as shown here

1. For two signatures \((r_1, s_1)\) and \((r_2, s_2)\) using the same random variable \(j\)
2. \(r_1 = r_2 = ((jP)_x\) and so can be described as \(r\) for both
3. \(s_1 = \frac{e_1 + zr}{j}\)
4. \(s_2 = \frac{e_2 + zr}{j}\)
5. \(s_1 - s_2 = \frac{(e_1 + zr) - (e_2 + zr)}{j} = \frac{e_1 - e_2}{j}\)
6. \(j = \frac{e_1 - e_2}{s_1 - s_2}\)
7. \(z = \frac{js_1 - e_1}{r} = \frac{e_1s_2 - e_2s_1}{r(s_1 - s_2)}\)

The ECCSI algorithm uses the same equation, eqn. 4.5, as ECDSA to calculate the \(r\) value of the signature; and for \(s\) the equation is inverted as shown here

\[
s = \frac{j}{e + zr} \quad (4.7)
\]

Therefore, inverting the values for \(s\) allows for a similar relationship to be derived when the same value of \(j\) is used for two different signatures from the same user

\[
z = \frac{e_1s_2^{-1} - e_2s_1^{-1}}{r(s_1^{-1} - s_2^{-1})} \quad (4.8)
\]

Therefore the ECCSI signature algorithm is vulnerable to the same attack, if the value of \(j\) is predictable. This is also noted as a security concern in the original RFC describing the ECCSI protocol [53].

An evaluation of the security provided by the ECCSI protocol in MIKEY-SAKKE is given in section 4.3.
4.1.4 AES

AES is the symmetric encryption protocol used for encrypting the communications between users once a shared secret value has been established. It is also used in the key wrap encryption protocol for sending key information between the key management server and the user. For AES I researched any known protocol weaknesses and discuss these briefly below for their impact on the overall security provided by the MIKEY-SAKKE protocol.

Timing attacks are a general weakness of many different protocols as discussed in 2.3.1. Specifically for AES there have been a number of examples for cache timing attacks [122, 10, 57, 91]. These exploit the use of a shared system cache of separate isolated virtual machines. Using this technique [122] are able to extract the AES encryption key.

In contrast to the side channel attack that exploits timing leaks from the AES protocol, there are currently no known practical attacks for directly breaking the AES encryption. After the release of Rijndael as the Advanced Encryption Standard there have been a number of theoretical attacks discussed but none of these are deemed practical for actual implementation.

One of these theoretical attacks is the XSL attack [32, 46, 41, 85, 86]. This exploits the simple algebraic description of the AES S-box which “can be described by an overdefined system of algebraic equations”[32] and then minimised and ultimately solved [106].

The related key attack, another theoretical attack, involves observation of ciphers encrypted with different keys for which there is some known relationship that can be exploited. [17], [16] and [62] all demonstrate related key attacks on AES using 192 or 256 bit keys that are faster than exhaustive searches but still remain impractical for actual attacks.

An evaluation of the impact of these attacks is given in section 4.3 as well as discussion on the impact to the security of the overall MIKEY-SAKKE protocol.

4.2 Implementation

For the low-level exploit investigation I chose to focus on the MOV attack on the SAKKE supersingular curve.

I used the Secure Chorus reference implementation of MIKEY-SAKKE [113] on which to base my attack. This implementation is split into three libraries: libms, libmksms and libcrypto, for which the functionality is described in detail in [110]. These client side libraries are written in C++ and the Visual Studio\(^1\) solution also includes a test application as well as an external key store library and a miscellaneous library.

The key management server implementation is written mainly in Perl and I installed the key management server code onto an Ubuntu\(^2\) virtual machine running in the Imperial College London cloud. The client code I installed on my Windows laptop for multiple clients, the exact instructions for these installations are documented in appendices A and B.

Once the key management server was running, I was able to use the test application provided in the Secure Chorus client code to initiate two separate clients, Jim and Bob; provide them with keys; and update their certificates from the key management server. In addition a secure communication could be established between Jim and Bob using the command line interface and the UDP communication implementation provided by the Secure Chorus test application.

Using Wireshark,\(^3\) a network protocol analyser, I was able to capture and analyse the packets sent between the key management server and the clients, Jim and Bob. These included:

- The HTTP requests and responses between each of the clients and the key management server for initialisation; key provision and certificate cache updates.
- The MIKEY-SAKKE I_MESSAGE sent from one client to the other in order to establish a secure communication channel.

---

\(^1\)https://www.visualstudio.com/
\(^2\)http://releases.ubuntu.com/14.04/
\(^3\)https://www.wireshark.org/
Analysing MIKEY-SAKKE

4th September, 2015

Figure 4.1: Example Section of Key Provision Response from KMS [113]

- The AES encrypted UDP messages for the secure communication between the two clients.

By intercepting an HTTP response from the key management server to a key provision request an attacker can easily obtain the key management server public keys. The XML content, for which an example section is shown in figure 4.1, identifies the SAKKE public key as $<\text{PubEncKey}>$. From this point on I conducted a thought experiment into the feasibility of the MOV attack on the SAKKE supersingular curve. The aim is for the attacker to derive the key management server secret key from the public key in order to be able to derive the receiver secret keys for all users in the domain and therefore have the ability to decrypt all secure communications to users in the domain.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p$</td>
<td>a large prime integer</td>
</tr>
<tr>
<td>$F_p$</td>
<td>a finite field with $p$ elements, also described as order $p$</td>
</tr>
<tr>
<td>$F_p^*$</td>
<td>a multiplicative group of the finite field $F_p$</td>
</tr>
<tr>
<td>$E(F_p)$</td>
<td>an elliptic curve over the finite field $F_p$, where $E : y^2 = x^3 + ax + b$</td>
</tr>
<tr>
<td>$q$</td>
<td>a large odd prime integer that divides $p + 1$</td>
</tr>
<tr>
<td>$E[q]$</td>
<td>the cyclic subgroup of the elliptic curve $E$ of order $q$</td>
</tr>
<tr>
<td>$P$</td>
<td>a point on $E(F_p)$ that generates the cyclic subgroup $E[q]$</td>
</tr>
<tr>
<td>$Q = [m]P$</td>
<td>elliptic curve point multiplication such that $Q$ is the result of adding $P$ to itself $m$ times</td>
</tr>
<tr>
<td>$k$</td>
<td>the embedding degree, where $p^k = 1 \mod q$</td>
</tr>
<tr>
<td>$e_q(S,T)$</td>
<td>the Weil pairing of elliptic curve points $S$ and $T$ in order $q$</td>
</tr>
<tr>
<td>$f_P(Q)$</td>
<td>is a function of the elliptic curve point $P$ applied to the point $Q$ computed using Miller’s algorithm [79]</td>
</tr>
<tr>
<td>$number_of_bits(x)$</td>
<td>specifies the number of bits for integer $x$</td>
</tr>
<tr>
<td>$h_{X,Y}$</td>
<td>defined as $l_{X,Y}/v_{X,Y}$ where $l$ is the line through elliptic curve points $X$ and $Y$ and $v$ is the vertical line through $X + Y$ and $-(X + Y)$</td>
</tr>
</tbody>
</table>

Table 4.6: MOV Attack Notation

The MOV attack requires the following steps to be implemented as previously detailed in section 4.1.2 and repeated below:

1. $Q = [m]S$ is the ECDLP over the curve $E(F_p)$ where $Q$ and $S$ are points on the curve and $m$ is an integer value
2. Choose a random point $T$ in $E(F_p)$, that is not equal to $S$, $Q$ or $O$
3. Check $T$ has order $q$, otherwise choose a different random point
4. Compute the Weil pairing values using Miller’s Algorithm [79, 80]:
   \[\alpha = e_q(S, T) \text{ in } F_{p^k}^*\]
   \[\beta = e_q(Q, T) \text{ in } F_{p^k}^*\]

5. Solve the DLP for \(\alpha\) and \(\beta\) in \(F_{p^k}^*\) as \(\beta = \alpha^m\)

6. The value for \(m\) now also solves the ECDLP for \(Q = [m]S\)

When applying the MOV attack to the SAKKE key management server elliptic curve discrete logarithm problem: \(Q\) described in the steps above represents the key management server public key; \(S\) is the generating point or base point for the elliptic curve; and \(m\) is the key management server secret key. This is the ECDLP as described in step 1 of the MOV attack.

Steps 2 and 3 can be implemented by the find_EC_POINT function as defined in appendix G. This chooses an elliptic curve point at random; checks it is not infinity; and checks the order of the point. While the order of the point does not equal \(q\) another point is chosen to be checked. This function along with the simple checks to ensure the chosen point is not equal to \(S\) or \(Q\) completes these steps successfully.

Step 4 involves computing the Weil pairing for each of the points with the third randomly chosen point. The Weil pairing can be defined as [115]

\[
e_n(P, Q) = \frac{f_P(Q + S)}{f_P(S)} \div \frac{f_Q(P - S)}{f_Q(-S)}
\] (4.9)

where \(P\) and \(Q\) are elliptic curve points on the curve \(E(F_p)\) to be paired order \(n\); \(S\) is a third elliptic curve point on the same curve that is not in the subgroup \(E[n]\); and \(f\) denotes a function that can be computed using Miller’s algorithm [79].

Miller’s algorithm [79] computes a function \(f_P\) for the Weil pairing by following the steps defined below [115, 119]:

1. Set \(T = P\) and \(f = 1\)

2. Loop \(i = \text{number\_of\_bits}(n - 1)\) down to 0
   (a) Set \(f = f^2 \cdot h_{T,T}\)
   (b) Set \(T = 2T\)
   (c) If \(n_i = 1\)
      i. Set \(f = f \cdot h_{T,P}\)
      ii. Set \(T = T + P\)
   (d) End if

3. End loop

4. Return the value \(f\)

where \(n_i\) is the \(i^{th}\) bit of \(n\); and \(h_{X,Y} = l_{X,Y}/v_{X,Y}\) such that \(l\) is the line through the two elliptic curve points and \(v\) is the vertical line through \(X + Y\) and \(-(X + Y)\).

For the calculation of \(l_{X,Y}\), the coordinates of \(X\) are defined as \((x, y)\) and the equation of the line is

\[-\lambda X + Y + (-y + \lambda x)\]

where \(\lambda\) is the gradient of the line. If \(X = Y\) then we require the tangent to the elliptic curve at point \(X\) so \(\lambda = \frac{3x^2 + a}{2y}\) where \(a\) is the parameter from the original definition of the elliptic curve, \(E\). Otherwise we require the line through both points \(X = (x_1, y_1)\) and \(Y = (x_2, y_2)\) therefore \(\lambda = \frac{y_2 - y_1}{x_2 - x_1}\).
For the vertical line through $X + Y$ and $-(X + Y)$, such that the point generated by $X + Y$ can be assigned the coordinates $(x, y)$ then the vertical line $v_{X,Y}$ is calculated simply as

$$X - x$$

(4.11)

These line equations allow for the definition of the $f$ function which is then applied to the elliptic curve point(s) specified in the brackets. For example, $f_P(Q)$ is the function for $P$ applied to $Q$. Once calculated these functions can then be used to derive the Weil pairing as defined in equation 4.9.

Step 5 of the MOV attack requires the discrete logarithm problem to be solved for $\alpha$ and $\beta$ in $F_{p^k}^*$ as $\beta = \alpha^m$. There are a number of different methods that can be used to do this. [101] describes a simple $O(n^3)$ to solve the discrete logarithm problem, which I implemented in C++ and demonstrated as successful as shown in appendix H. However, this algorithm is not particularly efficient especially with the large parameter sizes used in cryptography.

Alternatively, index calculus [123] is a sub-exponential algorithm for solving the discrete logarithm problem for the multiplicative group of a finite field. This method can solve the discrete logarithm problem in

$$\exp \left( c^3 \sqrt{(\log q)(\log \log q)^2} \right) \text{ steps}$$

(4.12)

where $q$ is the order of the finite field $F_q$ where $F_q^*$ is the multiplicative group; and $c$ is a small absolute constant.

It is the embedding degree that makes supersingular elliptic curves particularly susceptible to the MOV attack as for these curves the embedding degree is limited to $k \leq 6$. The embedding degree can be calculated by $p^k = 1 \mod q$. In the case of the SAKKE supersingular curve $k = 2$, which a particularly low value and could imply that this attack may be effective. However, MIKEY-SAKKE uses a 1024 bit value for $p$ therefore $p^2$ gives a 2048 bit value meaning the MOV attack transfers the elliptic curve discrete logarithm problem into a discrete logarithm problem for a multiplicative group of a finite field of order $2^{2048}$ [56].

This is equivalent to an RSA protocol using 2048 bit lengths and the longest known key length to be successfully factorized for RSA is 768 bits [67] and “factoring a 1024-bit RSA modulus would be about a thousand times harder” [67].

Therefore the MOV attack cannot be practically implemented on the SAKKE supersingular elliptic curve as the large parameters ensure the size of the multiplicative field is infeasibly large in which to solve a discrete logarithm even if the value of the embedding degree is only 2.

### 4.3 Evaluation

From the low level exploit investigations conducted there are a range of results as for some areas, initially thought to be weak, are now shown as secure; some routes of attack through exploitation of a weakness have been identified; and some further possible areas of weakness have been identified but require further investigation.

The MIKEY protocol, when used as MIKEY-SAKKE, was found to be secure against the man-in-the-middle attacks to which MIKEY-DH was previously found to be vulnerable [6, 58]. However, it is found to be vulnerable to user denial of service attacks [42].

It is shown that the SAKKE supersingular elliptic curve is not vulnerable to the MOV attack or the rho attack due to the large parameter sizes used [56, 67].

Elliptic curve points in general are known to be easily distinguishable from uniform random strings so both SAKKE and ECCSI are potentially vulnerable to attacks that exploit this weakness. Further investigation into the exact exploitation that could be used to attack the protocol from this are required as well as investigation into possible methods that could be used to disguise the elliptic curve points as random data [14].

The NIST P-256 elliptic curve [75] used for the ECCSI protocol is identified as having potential vulnerabilities in the implementation of the ladder used for scalar multiplication [12]. The exact implementation of the ladder used by the Secure Chorus reference implementation [113] is required
to determine if this is a vulnerability for this implementation. In addition recommendations for the
ladders [84, 21] that should be used with this protocol in the future would provide a better basis
for the security of the MIKEY-SAKKE protocol in all implementations.

The issue of the lack of rigidity in the NIST P-256 curve [75] is raised [12] and again further
investigation needs to be done to identify how this could be exploited. An alternative would also
be to select an alternative elliptic curve for use with this protocol. Appendix I lists the elliptic
curves defined as safe by the SafeCurves team [12], however, again, further investigation into the
suitability of any of these curves for the MIKEY-SAKKE protocol would need to be conducted
before use. In addition the MIKEY-SAKKE protocol aims to provide a standard to enable secure
communications between different domains. Therefore if a different curve were to be chosen then
these restrictions would also need to be considered.

The attack exploiting a predictable pseudo-random number generator or even a static variable
[4], that was previously conducted against the ECDSA signature protocol [64] is also found to be
applicable to the ECCSI protocol. Therefore, further investigation to ensure the use of a secure
random number generator [104] is required for the MIKEY-SAKKE protocol in order to prevent
an attacker from breaking the authentication protocol and being able to impersonate a user in the
domain.

Finally, even though there are no practical direct attacks against the AES protocol there are a
number of side channel attacks, specifically exploiting cache timing [122, 10, 57, 91]. Therefore this
is an area of weakness that needs to be investigated further to ensure all preventative measures can
be taken to avoid these attacks.

Overall, no fundamental vulnerabilities in the MIKEY-SAKKE protocol have been found, how-
ever, there are a number of areas requiring further investigation; and it is recommended that
this be continued throughout the lifetime of the protocol as technologies and attack techniques
advance then new vulnerabilities can be introduced. This all relates to the Continuous Iterative
Improvement approach to security analysis, as discussed further in section 5.1.
Chapter 5

Evaluation Summary and Conclusion

5.1 Overall Evaluation

For this project there are two threads of analysis: high level security analysis; and low level exploit investigation. Using these two approaches together enables an approach of Continuous Iterative Improvement, which is applied to the security modelling as well as the design, implementation and use of the cryptographic protocol itself.

Figure 5.1 shows the Continuous Iterative Improvement cycle, as first discussed in section 2.2, applied to the analysis of MIKEY-SAKKE, including the specific tasks undertaken as part of this project and resulting suggestions and recommendations.

<table>
<thead>
<tr>
<th>Adversary</th>
<th>Attack</th>
<th>Mean Probability of Success</th>
<th>Confidence Interval</th>
</tr>
</thead>
<tbody>
<tr>
<td>Corrupt Government Official</td>
<td>Key Management Server Denial of Service</td>
<td>0.999</td>
<td>0.002</td>
</tr>
<tr>
<td>Government Cyberwarrior</td>
<td>Key Management Server Denial of Service</td>
<td>1.000</td>
<td>0.000</td>
</tr>
<tr>
<td>Government Spy</td>
<td>Read Message</td>
<td>0.353</td>
<td>0.009</td>
</tr>
</tbody>
</table>

Table 5.1: Möbius Model Final Results

The automated simulations of the Möbius tool with the accurate capture of the system and the adversaries through the ADVISE formalism enabled the provision of relevant metrics in the form of probabilities of each attack goal for each adversary. The final results from the Möbius ADVISE security model are given in table 5.1.

The low level exploits conducted, as shown in the red circles on figure 5.1, enabled the Continuous Iterative Improvement of the Möbius model. Hence the results from the investigations in chapter 4 and the new vulnerabilities identified were fed back into the Möbius model in chapter 3 to improve the accuracy and completeness of the high level security model. For example, understanding the ability to exploit a predictable pseudo-random number generator enabling the derivation of the user’s secret signing key allowed this attack to be introduced into the model as another route to achieve the impersonate user goal [4].

Protocol design improvements are shown by orange circles on figure 5.1 for which here only one is described. Following the evaluation of the SafeCurves analysis [12] for the NIST P-256 elliptic curve [75] there is the possibility that the protocol could be improved by the use of a different elliptic curve, see appendix I. As discussed in section 4.3, further investigation of alternative curves would be required but this is included as a potential protocol design improvement derived from the low level analysis of the elliptic curves.

The yellow circles on figure 5.1 represent implementation bug fixes. Investigation of the elliptic curves used for the SAKKE and ECCSIE protocols raised concerns about the implementation of
the ladders used for scalar multiplication [12], which can be exploited by timing attacks to derive key information [23]. Use of a secure ladder [84, 21] in the protocol implementation would protect against these attacks and is recommended as an implementation bug fix.

My original objective was to design an open, accurate and dynamic method for security analysis protocols and use that to conduct a thorough security analysis of the MIKEY-SAKKE protocol. The Continuous Iterative Improvement approach enables ongoing evaluation and development of the protocol; the clear quantitative ADVISE formalism with automated simulations provided by the Möbius tool provide a high level of accuracy in specifying different areas of vulnerability; and the open nature of this report enables review and progression by other researchers, designers and analysts.

The security analysis of the MIKEY-SAKKE protocol established the infeasibility of certain attacks and the potential for exploitation of other weaknesses. Overall, the Möbius model in the last version identified three feasible attacks, as shown in table 5.1. However, due to time limitations this is by no means a complete analysis; as identified by the low level exploit investigations there are a number of areas that require further analysis. Therefore in order to obtain a more complete security analysis of the MIKEY-SAKKE protocol further iterations of the high level security analysis need to be conducted taking into account results from further low level exploit investigations.

<table>
<thead>
<tr>
<th>MIKEY-SAKKE</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Vulnerabilities and Attacks</strong></td>
</tr>
<tr>
<td>MIKEY User Denial of Service</td>
</tr>
<tr>
<td>ECCSI User Impersonation, if the random variable is predictable or static</td>
</tr>
<tr>
<td>Distinguishable Elliptic Curve Points</td>
</tr>
<tr>
<td>ECCSI Insecure Ladder Implementation</td>
</tr>
<tr>
<td>ECCSI Manipulatable Curve Generation</td>
</tr>
</tbody>
</table>

Figure 5.1: Continuous Iterative Improvement for MIKEY-SAKKE
5.2 Conclusion

In conclusion I have been able to derive an effective approach to security analysis of the MIKEY-SAKKE cryptographic protocol. I have been able to produce an accurate analysis of the protocol highlighting specific areas of vulnerability by the probability of a successful attack by an adversary.

Throughout this project I have been able to continuously improve the high level security analysis of the protocol by incorporating new information determined from a range of low level exploit investigations. However, as covered below in section 5.3, there are a number of areas that require further investigation in order to provide a more complete security analysis of the MIKEY-SAKKE protocol.

The Continuous Iterative Improvement approach to security analysis incorporating both high level security analysis and low level exploit investigation is in theory a valid approach for the security analysis of a wide range of cryptographic protocols. Further implementation of this approach on different protocols will provide further insight into the effectiveness of the approach more generally and identify if any improvements can be made.

5.3 Future Work

Specific areas of further investigation identified for the MIKEY-SAKKE protocol during the low level exploit investigations include:

- Investigation into the exact implementation method for conducting denial of service attacks against specific users and possible countermeasures that could be used to prevent such an attack.
- Evaluation of the SAKKE supersingular elliptic curve against the criteria defined by SafeCurves as only the rho method and the transfer attack (MOV attack) were covered in detail in this project.
- Investigation into the impact of the distinguishability of elliptic curve points when used in cryptography and review of potential mechanisms that could be used to represent elliptic curve points with uniform random strings.
- Investigation into the vulnerabilities associated with the NIST P-256 curve being rated as Manipulatable by the SafeCurve criteria and how this could be exploited.
- Analysis of the safety of the ladder implementation for the elliptic curve scalar multiplication used by the Secure Chorus reference implementation in terms of vulnerability to a timing attack and vulnerability to specific failure cases.
- Investigation into the security of the pseudo-random number generator used by the Secure Chorus reference implementation and if it is predictable in any way, as there is an exploitation for the ECCSI authentication protocol based on a predictable pseudo-random number generator.
- Investigation into timing attacks for the AES protocol especially focusing on the type of devices that are likely to be used by users to run the MIKEY-SAKKE protocol, for example, smart phones.

Using the results obtained from the low level exploit investigations described above further iterations of the Möbius model for MIKEY-SAKKE can be implemented incorporating the new information and improving the completeness of the MIKEY-SAKKE security analysis. Also as a dynamic approach this analysis should continue to be conducted throughout the lifetime of the protocol and can be used to identify new weaknesses or balance the risks and requirements of different attacks based on developments in attack techniques and/or technological capabilities.

Further use of the Continuous Iterative Improvement approach to security analysis on different cryptographic protocols would also enable a broad evaluation of its effectiveness as an open, accurate and dynamic security approach for any cryptographic protocol.
Bibliography


doi: 10.1109/MSP.2014.31.


Appendices
Appendix A

Key Management Server Set Up

To run the Key Management Server (KMS) I used a virtual machine on the Imperial College London Department of Computing’s cloudstack [37]. This was set up using the following steps [113]:

1. Log into the cloudstack web interface using college credentials

2. Add an instance of the following spec:
   (a) Ubuntu v14.04 - 20Gb Community Template
   (b) CPU: 2 cores 2GHz & RAM: 2Gb Local Storage
   (c) Local Storage: 10Gb

3. Once the instance is created and running, open the console view from the web interface

4. Install the prerequisite libraries
   (a) apt-get -y install git
   (b) apt-get -y install build-essential
   (c) apt-get -y install git libssl-dev libgmp-dev
   (d) apt-get -y install libmojolicious-perl libxml-perl libxml-libxml-perl

5. Manually set up CPAN
   (a) Entering the command: cpan
   (b) Choosing the auto configuration when prompted

6. Use CPAN to install the further dependent libraries
   (a) cpan install Crypt::OpenSSL::AES
   (b) Crypt::OpenSSL::ECDSA Crypt::Blowfish Crypt::CBC Crypt::DES_EDE3 Crypt::Rijndael
       Data::Compare Encoding::BER Math::GMPz Crypt::OpenSSL::Random DBI YAML::XS
       DBD::SQLite Digest::SHA1 UUID::Tiny

7. Use git to clone the Secure Chorus MIKEY-SAKKE KMS libraries
   (a) git clone https://bitbucket.org/securechorus/sc_kms
   (b) git clone https://bitbucket.org/abbutcher/crypt-ecdsa-gmpz.git

8. Set up and install Crypt::ECDSA

1https://www.imperial.ac.uk/
2http://releases.ubuntu.com/14.04/
3http://www.cpan.org/
(a) pushd crypt-ecdsa-gmpz
(b) perl Makefile.PL
(c) make
(d) make install
(e) popd
(f) cd sc_kms
(g) morbo sckms.pl
Appendix B

Client Set Up

To run the Client I installed the Secure Chorus code on my Windows 8\(^1\) Sony Vaio\(^2\) laptop. To do this [113]:

1. Download the Secure Chorus MIKEY-SAKKE client libraries (sc_crypt) [113]
2. Download Win32 OpenSSL version 1.0.2c (openssl-1.0.2c.tar.gz)\(^3\)
3. Download Apache Xerces version 3.1.2 (xerces-c-3.1.2.zip)\(^4\)
4. Download Apache Santuario version 1.7.2 (xml-security-c-1.7.2.tar.gz)\(^5\)
5. Download Curl (curl-7.40.0.zip)\(^6\)
6. Extract each of the above zipped files into the external directory within the Secure Chorus folder
7. Rename the folders in the external directory as shown here:
   (a) curl
   (b) OpenSSL
   (c) xerces
   (d) xml-security-c-1.7.2
8. Build Apache Xerces:
   (a) Navigate to “external\xerces\projects\Win32\VC10\xerces-all”
   (b) Open the “xerces-all.sln” solution in Visual Studio\(^7\) and accept any prompts to update to VS2012
   (c) Change build to “Release” and build solution
9. Build Apache Santuario:
   (a) Navigate to “external\xml-security-c-1.7.2\Projects\VC10.0\xsec”
   (b) Open the “xsec.sln” solution in Visual Studio and accept any prompts to update to VS2012
   (c) Change build to “Release No Xalan”

\(^1\)http://windows.microsoft.com/en-gb/windows-8/upgrade-to-windows-8
\(^2\)http://www.sony.co.uk/support/en/hub/prd-comp-vaio
\(^3\)https://slproweb.com/products/Win32OpenSSL.html
\(^4\)http://xerces.apache.org/xerces-c/download.cgi
\(^5\)http://santuario.apache.org/download.html
\(^6\)http://curl.haxx.se/download.html
\(^7\)https://www.visualstudio.com/
(d) In the “xsec_lib” project, edit the file “version.rc” and replace the “afxres.h” with “windows.h”

(e) Build the solution

10. Build Curl:

(a) Open a the Visual Studio command prompt. This can be found by navigating to “All Programs”; then “Visual Studio”; and then “Visual Studio Tools”

(b) Within the command prompt navigate to “external\curl\install\”

(c) Enter the command “nmake /f Makefile.vc mode=dll” to build

11. Build MIKEY-SAKKE Client:

(a) Open the “SecureChorus.sln” solution in Visual Studio and accept any prompts to upgrade to VS2012

(b) Change build to “Release”

(c) In the project properties for each project set C++ “Additional Include Directories” and the Linker “Additional Library Directories” to the include and library paths of Curl, OpenSSL, Xerces and Santuario.

(d) Build the solution

In order to create a second client to analyse the client to client communications I created a virtual machine on my laptop.

1. Download and install Oracle VirtualBox\(^8\)

2. Download IE10 Windows 7 virtual machine image from Microsoft\(^9\)

3. Download and install Visual Studio\(^10\)

4. Repeat previously described steps to install MIKEY-SAKKE

\(^8\)https://www.virtualbox.org/

\(^9\)http://dev.modern.ie/tools/vms/linux/

\(^10\)https://www.visualstudio.com/
Appendix C

Intel Threat Agent Library

<table>
<thead>
<tr>
<th>Intent</th>
<th>NON-HOSTILE</th>
<th>Anarchist</th>
<th>Civil Activist</th>
<th>Competitor</th>
<th>Corrupt Government Official</th>
<th>Data Miner</th>
<th>Employee Disgruntled</th>
<th>Government Cyberwarrior</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access (f)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Internal</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>External</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Acquisition/Heist</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Business Advantage</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Damage</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Embarrassment</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tech Advantage</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Code of Conduct</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Legal</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Extra-legal minor</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Extra-legal major</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Individual</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Club</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cooperative</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Team</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Organization</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Government</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>None</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Minimal</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Operational</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Adopt</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Copy</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Deny</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Destroy</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Damage</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Take</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Goal of the Above</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Don’t Care</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Overt</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Covert</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Clandestine</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Multiple/Don’t Care</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure C.1: Intel Threat Agent Library Section 1 [25]
Figure C.2: Intel Threat Agent Library Section 2 [25]
Appendix D

CAPEC

CAPEC [81] is MITRE’s Common Attack Pattern Enumeration and Classification. Attack groups in terms of the mechanism used for the attack:

- **Gather information**
  - Excavation
  - Interception
  - Footprinting
  - Fingerprinting
  - Social Information Gathering Attacks/Information Elicitation via Social Engineering

- **Deplete resources**
  - Flooding
  - Excessive Allocation
  - Resource leak exposure
  - Sustained client engagement
  - Amplification

- **Injection**
  - Parameter injection
  - Code inclusion
  - Resource injection
  - Code injection
  - Command injection

- **Deceptive interactions**
  - Path traversal
  - Content spoofing
  - Identity spoofing
  - Resource location spoofing
  - Action spoofing

- **Manipulate timing and state**
  - Forced deadlock
  - Leveraging race conditions
- Leveraging time-of-check and time-of-use (TOCTOU) race conditions
- Cross-domain search timing
- Manipulating user state

**Abuse of functionality**
- API abuse/misuse
- Try all common application switches and options
- Cache poisoning
- Functionality misuse
- Directory traversal
- Abuse of communication channels
- Socket capable browser plugins result in transparent proxy abuse
- Forceful browsing
- WSDL scanning

**Probabilistic techniques**
- Brute force
- Screen temporary files for sensitive information
- Fuzzing
- Manipulating opaque client-based data tokens

**Exploitation of authentication**
- Authentication abuse
- Authentication bypass
- Exploitation of session variables, resource IDs and other trusted credentials

**Exploitation of authorisation**
- Privilege escalation
- Privilege abuse
- Exploiting trust in a client (AKA making the client invisible)
- Hijacking a privileged process
- Subvert code-signing facilities
- Target programs with elevated privileges

**Manipulate data structures**
- Buffer manipulation
- Attack through shared data
- Integer attacks
- Pointer attack
- Accessing/intercepting/modifying HTTP cookies

**Manipulate resources**
- Input data manipulation
- Resource location spoofing
- Infrastructure manipulation
- File manipulation
- Variable manipulation
- Configuration/environment manipulation
- Abuse of transaction data structure
- Audit log manipulation
- Schema poisoning
- Protocol manipulation
- Accessing/intercepting/modifying HTTP cookies
- Contaminate resource

• Analyse target
  - Reverse engineering
  - Cryptanalysis

• Gain physical access
  - Bypassing physical security
  - Physical theft

• Malicious code execution
  - Targeted malware

• Alter system components
  - Software integrity attacks
  - Hacking hardware devices or components
  - Malicious logic inserted into product
  - Physical destruction of device or component

• Manipulate system users
  - Target influence via social engineering
Appendix E

Elliptic Curves

Examples of an elliptic curve shown as a continuous curve over no field in figure E.1, and then as integer only points over a finite field in figure E.2.

Figure E.1: $E : y^2 = x^3 - x + 1$ [117]

Figure E.2: $E : y^2 = x^3 - x + 1 \mod 97$ [117]
Appendix F

Rho Method Cost Calculation [90]

```cpp
#include <iostream>
#include <openssl/bn.h>
using namespace std;

void BN_sqrt (BIGNUM *r, BIGNUM *n);

int main () {

    // q
    string str = "265EAEC7C2958FF6997184663684195E"
        "905B0338672D20986FA6B8D62CF8068B"
        "BD02A9C9F8BF03C68A1CC354C69672C"
        "39E46CE7FDFF2228645B49FD2999A9B4"
        "389B1921CC9AD335144AB173595A0738"
        "6DABFD2A0C614AA0A9F3CF1487DF026A"
        "A7E535ADB5A5C7C7FF38FA08E2615F6C"
        "203177C42B1EB331D99B601EBFAA17FB";

    if (!BN_hex2bn (&q, str ))
        cout << "Failed to set hexadecimal string to big number for q

        
"
;
    cout << "finding cost of rho" << endl;
    BIGNUM *rho = BN_new ();

    // compute square root of q
    BN_sqrt (rho, q);

    cout << "q " << BN_bn2dec (q) << endl;
    cout << "ans " << BN_bn2dec (rho) << endl;
    cout << "num of bits " << BN_num_bits (rho) << endl;

    // check answer
    BN_sqr (rho, rho, ctx);

    cout << "rho squared " << BN_bn2dec (rho) << endl;

    BIGNUM *test = BN_new ();
    BN_sub (test, q, rho);

    cout << "diff " << BN_bn2dec (test) << endl;

    BN_CTX_free (ctx);
    BN_free (q);
    BN_free (test);
    BN_free (rho);

    return 1;
}
```
// Approximately calculate squareroot of BIGNUM
void BN_sqrt(BIGNUM *r, BIGNUM *n) {
    // Method for function defined by:
    BN_CTX *ctx = BN_CTX_new();
    BIGNUM *M = BN_new();
    BIGNUM *P = BN_new();
    BIGNUM *two = BN_new();
    BIGNUM *calc = BN_new();

    BN_copy(calc, n);
    BN_copy(P, n);
    BN_set_word(two, 2);
    do {
        BN_copy(M, calc);
        BN_div(calc, NULL, P, M, ctx);
        BN_add(calc, M, calc);
        BN_div(calc, NULL, calc, two, ctx);
    } while (BN_cmp(M, calc) > 0);

    BN_copy(r, M);
    return;
}

RESULTS

q = 26944228087941870931184994387151710996385124719660355776394
    42892295267830208675210753112185681137312586415371617825734
    12789292204998623568761620647893217943801178969088565787142
    32966625977581336766070642571955951438455214662457037787017
    9788638950689760844363920624055575636349830198143607503012
    7754953234427

ans = 5190782937247300543873613573176206988299395865428665446359
    12494335647439415925303477622308179800588680624827829234912
    045661404734982782330566599527604893

number of bits in answer 511
Appendix G

MOV Attack - Find Elliptic Curve Point

```c
#include <openssl/bn.h>
#include <openssl/ec.h>

using namespace std;

int find_EC_POINT(EC_GROUP *curve, EC_POINT *T, BIGNUM *m) {
    BIGNUM *rand = BN_new();
    BIGNUM *ord = BN_new();
    BN_CTX *ctx = BN_CTX_new();
    EC_POINT *check = EC_POINT_new(curve);

    // loop until find point with order m
    do {
        if (!BN_rand_range(rand, m)) {
            // Failed to generate random number
            return 0;
        }

        if (!EC_POINT_mul(curve, T, rand, NULL, NULL, ctx)) {
            // Failed to generate point T
            return 0;
        }

        if (!EC_POINT_mul(curve, check, NULL, T, m, ctx)) {
            // Failed to multiply elliptic curve points
            return 0;
        }

        find_order(curve, T, ord, ctx);
    } while (EC_POINT_is_at_infinity(curve, T) || BN_cmp(ord, m) != 0);

    BN_free(rand);
    BN_free(ord);
    BN_CTX_free(ctx);
    EC_POINT_free(check);

    return 1;
}

void find_order(EC_GROUP *curve, EC_POINT *T, BIGNUM *m, BN_CTX *ctx) {
    EC_POINT *test = EC_POINT_new(curve);
    EC_POINT_copy(test, T);
    BIGNUM *one = BN_new();
    BN_one(one);
    BN_one(m);

    // loop until point is at infinity
    // increment m each time to give order of point T
    while (!EC_POINT_is_at_infinity(curve, test)) {
        EC_POINT_add(curve, test, test, T, ctx);
    }
}
```
BN_add(m, m, one);
}

EC_POINT_free(test);
BN_free(one);

return;
#include <openssl/bn.h>

using namespace std;

// Method for function defined by:
// http://arxiv.org/pdf/0912.2269v1.pdf?e=afqzCNFg2VU5AFh01bD8z89FTF8suJD0g6v92=PsKJFFeyJ9oAN3MFUJtSzA8cad=rja

bool dlp(BIGNUM *x, BIGNUM *y, BIGNUM *p, BIGNUM *k) {
    BIGNUM *lv_x1 = BN_new();
    BIGNUM *lv_x2 = BN_new();
    BIGNUM *lv_y1 = BN_new();
    BIGNUM *lv_y2 = BN_new();
    BIGNUM *lv_e1 = BN_new();

    BN_copy(lv_x1, x);
    BN_zero(lv_x2);
    BN_copy(lv_y1, y);
    BN_copy(lv_y2, lv_y1);
    BN_one(lv_e1);
    BN_zero(k);

    BIGNUM *i = BN_new();
    BIGNUM *j = BN_new();
    BIGNUM *one = BN_new();
    BN_one(one);

    for (BN_one(i); BN_cmp(i, p) <= 0; BN_add(i, i, one)) {
        BN_zero(lv_x2);
        for (BN_one(j); BN_cmp(j, x) <= 0; BN_add(j, j, one)) {
            BN_add(lv_x2, lv_x2, lv_x1);
        }
        BN_copy(lv_x1, lv_x2);
        BN_add(lv_e1, lv_e1, one);
        while (BN_cmp(lv_x1, p) > 0) {
            BN_sub(lv_x1, lv_x1, p);
        }
        BN_copy(lv_x1, lv_x2);
        BN_add(lv_e1, lv_e1, one);
        while (BN_cmp(lv_x1, p) > 0) {
            BN_sub(lv_x1, lv_x1, p);
        }
        if (BN_cmp(lv_x1, lv_y2) == 0) {
            BN_copy(k, lv_e1);
            break;
        }
    }
    BN_free(lv_x1);
BN_free(lv_x2);
BN_free(lv_y1);
BN_free(lv_y2);
BN_free(lv_e1);
BN_free(i);
BN_free(j);
BN_free(one);

if (!BN_is_zero(k))
    return true;

return false;
}
Appendix I

Safe Curves

The NIST P-256 [75] elliptic curve classification is therefore determined not to be a safe choice for elliptic curve cryptography [12]. The curves shown in table I.1 are alternatives that could be used instead of NIST P-256, however, this would be contradicting the RFC specification for MIKEY-SAKKE [54].

<table>
<thead>
<tr>
<th>Name</th>
<th>Definition</th>
<th>Reference(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>M-221</td>
<td>$y^2 = x^3 + 117050x^2 + x$ and modulo $p = 2^{221} - 3$</td>
<td>[8]</td>
</tr>
<tr>
<td>E-222</td>
<td>$x^2 + y^2 = 1 + 160102x^2y^2$ and modulo $p = 2^{222} - 117$</td>
<td>[8]</td>
</tr>
<tr>
<td>Curve1174</td>
<td>$x^2 + y^2 = 1 - 1174x^2y^2$ and modulo $p = 2^{251} - 9$</td>
<td>[14]</td>
</tr>
<tr>
<td>Curve25519</td>
<td>$y^2 = x^3 + 486662x^2 + x$ and modulo $p = 2^{255} - 19$</td>
<td>[11]</td>
</tr>
<tr>
<td>E-382</td>
<td>$x^2 + y^2 = 1 - 67254x^2y^2$ and modulo $p = 2^{382} - 105$</td>
<td>[8]</td>
</tr>
<tr>
<td>M-383</td>
<td>$y^2 = x^3 + 2065150x^2 + x$ and modulo $p = 2^{383} - 187$</td>
<td>[8]</td>
</tr>
<tr>
<td>Curve383187</td>
<td>$y^2 = x^3 + 229969x^2 + x$ and modulo $p = 2^{383} - 187$</td>
<td>[8]</td>
</tr>
<tr>
<td>Curve41417</td>
<td>$x^2 + y^2 = 1 + 3617x^2y^2$ and modulo $p = 2^{414} - 17$</td>
<td>[13]</td>
</tr>
<tr>
<td>Ed448-Goldilocks</td>
<td>$x^2 + y^2 = 1 - 39081x^2y^2$ and modulo $p = 2^{448} - 2^{224} - 1$</td>
<td>[59]</td>
</tr>
<tr>
<td>M-511</td>
<td>$y^2 = x^3 + 530438x^2 + x$ and modulo $p = 2^{521} - 187$</td>
<td>[8]</td>
</tr>
<tr>
<td>E-521</td>
<td>$x^2 + y^2 = 1 - 376014x^2y^2$ and modulo $p = 2^{521} - 1$</td>
<td>[13, 8]</td>
</tr>
</tbody>
</table>

Table I.1: Safe Curves [12]