Towards inconspicuous security

Goh, Weihan

2013

Goh, W. (2013). Towards inconspicuous security. Doctoral thesis, Nanyang Technological University, Singapore.

<https://hdl.handle.net/10356/54997>

<https://doi.org/10.32657/10356/54997>

1	one inch + <code>\hoffset</code>	2	one inch + <code>\voffset</code>
3	<code>\oddsidemargin = 36pt</code>	4	<code>\topmargin = 18pt</code>
5	<code>\headheight = 0pt</code>	6	<code>\headsep = 0pt</code>
7	<code>\textheight = 648pt</code>	8	<code>\textwidth = 417pt</code>
9	<code>\marginparsep = 10pt</code>	10	<code>\marginparwidth = 85pt</code>
11	<code>\footskip = 30pt</code>		<code>\marginparpush = 7pt</code> (not shown)
	<code>\hoffset = 0pt</code>		<code>\voffset = 0pt</code>
	<code>\paperwidth = 597pt</code>		<code>\paperheight = 845pt</code>

1	one inch + \hoffset	2	one inch + \voffset
3	\evensidemargin = 0pt	4	\topmargin = 18pt
5	\headheight = 0pt	6	\headsep = 0pt
7	\textheight = 648pt	8	\textwidth = 417pt
9	\marginparsep = 10pt	10	\marginparwidth = 85pt
11	\footskip = 30pt		\marginparpush = 7pt (not shown)
	\hoffset = 0pt		\voffset = 0pt
	\paperwidth = 597pt		\paperheight = 845pt

**NANYANG
TECHNOLOGICAL
UNIVERSITY**

TOWARDS INCONSPICUOUS SECURITY

GOH WEIHAN

SCHOOL OF COMPUTER ENGINEERING

2013

TOWARDS INCONSPICUOUS SECURITY

GOH WEIHAN

School of Computer Engineering

A thesis submitted to the Nanyang Technological University
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy

2013

Statement of Originality

I hereby certify that the work embodied in this thesis is the result of original research done by me and has not been submitted for a higher degree to any other University or Institute.

October 30, 2013

.....

Date

.....

Goh Weihan

Abstract

This dissertation presents novel ideas on the design and analysis of techniques to provide security while being minimally conspicuous. The author identifies three key areas in which such techniques can be used, and evaluates the performance of the techniques when used. The three areas identified are the areas of information anti-forensics, mobile system anti-forensics, and threat mitigation in disruption tolerant networks. Additionally, the author proposes a technique to utilize a Trusted Platform Module to enhance the security of the key generating authority in a Shamir identity-based signature scheme.

For the first area, a system designed to hinder forensics against information is presented. The system is designed from the perspective of a criminal; the idea being that a criminal would more likely find itself in a situation where forensics retrieval of information could result in severe consequences. A Trusted Platform Module and the Linux Integrity Measurement Architecture are used in its construction to enable the establishment of a system's trustworthiness.

Although designed from a criminal's point-of-view, ideas are presented on how such a system could be used instead for good and legitimate purposes.

Next, a method of deploying Android in RAM to hinder forensics in mobile devices is presented. The method entails having the operating system copy itself from persistent storage to RAM and continues execution from there, while deleting sensitive components and certain other entities from persistent storage, including traces of the deletion. It is shown how such a method would be disadvantageous to first responders in particular and forensics in general, as it is almost unlikely for first responders to know if an acquired device is actually engaging in measures to remove sensitive components.

For threat mitigation in disruption tolerant networks, an authentication protocol for beacon message information using the Shamir identity-based signature (IBS) scheme is presented. The signature generated contains a hidden code that can be used to signal if the signer is compromised. On top of that, it is shown how the authentication information, once obtained by the verifier, can be further used

by a routing protocol to secure the network from attacks. Flooding and spoofing attacks are shown to severely impact delivery rates in routing protocols that are not protected, however with protection to the protocols, successful mitigation of the attacks is shown to be possible.

Finally, an implementation of a key generating authority (KGA) for the Shamir IBS scheme using a TPM is presented. It demonstrates how TPMs can be used for alternative, and sometimes more lightweight causes that extend beyond the documented, specification-outlined uses. As the earlier authentication protocol for beacon message information requires a KGA for the Shamir IBS scheme, the implementation can be used there as well.

Acknowledgment

The pursuit over the past five years culminating in this dissertation would not be possible without the support and assistance of the following individuals and organizations:

Associate Professor Yeo Chai Kiat My supervisor, for her strong guidance and insight, unwavering support, and constant motivation in the technical aspects of my research and in advising me on coping as a PhD candidate.

Senior Research Scientist Leong Peng Chor For his insight regarding security, especially for Chapter 1, and the eye-opening discussions.

Associate Professor Lee Bu Sung, Francis For his constant and invaluable advice on research.

Center for Multimedia and Network Technology (CeMNet) Ms Chua Poo Hua, Ms Cindy Teo, and Ms Siom Siew Ling for assisting me in laboratory issues, and keeping the laboratory a conducive place to conduct research. Fellow students in CeMNet, past and present, for the discussions and lively environment.

Friends and Family For being there for me all the time.

Contents

Abstract	i
Acknowledgment	iii
List of Tables	xi
List of Figures	xiii
List of Abbreviations	xv
1 Introduction	1
1.1 Research Objective	1
1.2 Motivation and Scope	2
1.2.1 Hindering Forensics against Information	3
1.2.2 Hindering Forensics in Mobile Systems	5
1.2.3 Threat Mitigation in Disruption Tolerant Networks	6
1.3 Contributions	9
1.3.1 Hindering Forensics against Information	10
1.3.2 Hindering Forensics in Mobile Systems	10
1.3.3 Threat Mitigation in Disruption Tolerant Networks	11
1.3.4 Using a Trusted Platform Module as a Key Generating Authority	12
1.4 Theme of Research	13
1.5 Dissertation Organization	15

2	Literature Review	17
2.1	On Hindering Forensics	17
2.1.1	Electronic Evidence and Digital Forensics	17
2.1.1.1	Digital Forensics	17
2.1.2	Hindering Digital Forensics	19
2.1.2.1	Concealing Information	19
2.1.2.2	Data and Data Source Tampering	20
2.1.3	Mobile Operating System	20
2.2	Trusted Platform Module and Chain of Trust	21
2.2.1	Remote Attestation	25
2.2.2	Data Binding and Data Sealing	26
2.2.3	Integrity Measurement Architecture	26
2.3	On Threat Mitigation in Disruption Tolerant Networks	27
2.3.1	Disruption Tolerant Networks	27
2.3.2	Routing in Disruption Tolerant Networks	28
2.3.3	Threat Model	29
2.3.4	Bundle Protocol Security	31
2.3.5	Routing Protocol Security	33
2.3.6	Disruption-Tolerant Network Neighbor Discovery	35
2.4	Identity-Based Signature Scheme	37
3	Hindering Forensics against Information	39
3.1	System Objectives and Requirements	40
3.1.1	Anti-Forensic Measures	40
3.1.1.1	Information Confidentiality	42
3.1.1.2	Information and Action Hiding	44
3.1.1.3	Plausible Deniability	45
3.1.2	Legal Consideration	46
3.2	System Design	48
3.2.1	High-Level Architecture	48

3.2.2	Design Considerations	50
3.2.3	Attestation Mechanism and Design	52
3.2.4	Client-Server Transaction / Communications Protocol	54
3.2.5	The Extend Ordered Measurement List Process	57
3.2.6	System Data Sets	59
3.2.7	Data Encryption	62
3.2.8	Data Decryption	63
3.3	Security Analysis of Design	63
3.3.1	Authentication and Attestation	64
3.3.2	Data Read Transaction	67
3.3.3	Data Write Transaction	68
3.3.4	Information and Action Hiding	69
3.3.4.1	Information Hiding	69
3.3.4.2	Action Hiding	73
3.3.5	Plausible Deniability	73
3.3.6	Role of the Trusted Platform Module	76
3.4	Prototype Implementation and Anti-Forensics Capabilities Test	77
3.5	Positive Uses	82
3.6	Conclusion	82
4	Hindering Forensics in Mobile Systems	85
4.1	Challenges for Enabling Anti-Forensics in Mobile Devices	86
4.2	Considerations for Anti-Forensics in A Mobile Operating System	87
4.2.1	Anti-Forensic Measures	89
4.2.2	RAM-Based Method to Hinder Forensics	90
4.3	The Android Operating System and Forensics	91
4.4	Executing Android in RAM	92
4.4.1	Managing the Communication Channel	94
4.5	Evaluating a RAM-Based Android Instance	95
4.5.1	Discussion	98

4.6	Conclusion	99
5	Threat Mitigation in Disruption Tolerant Networks	101
5.1	The Case for Authentication and Authorization	102
5.1.1	Tactical Level Disruption Tolerant Networks	103
5.1.2	Attacks Against Disruption Tolerant Networks	104
5.1.2.1	Flooding	104
5.1.2.2	Impersonation	107
5.1.2.3	Hijacking	107
5.2	System Objectives and Requirements	109
5.2.1	High-Level Adversary Model	109
5.2.2	Threat Mitigation Aims and Measures	110
5.2.2.1	Non-interactive Authentication	111
5.2.2.2	Covert Communication of Node State	111
5.2.2.3	Plausible Deniability of Covert Communication	111
5.3	Proposed Protocol	112
5.3.1	Network Pre-Deployment Phase	113
5.3.2	Signature Key Generation	113
5.3.3	Beacon Message Construction	114
5.3.4	Beacon Message Verification	116
5.4	Covert Signaling and Plausible Deniability	118
5.4.1	Plausible Deniability and Disinformation	121
5.5	Security Comparison to Bundle Authentication Block	122
5.6	Improving the Protocol	123
5.6.1	A More Efficient Construction for the Translucent Set Mem- ber x_t	123
5.6.2	Implementing the Security Protocol at the Routing Protocol	124
5.7	Utilizing the Information from the Security Protocol	126
5.7.1	Using the Information to Prevent Flooding Attacks	126

5.7.2	Using the Authentication Information to Prevent Source Spoofing Attacks	133
5.8	Conclusion	140
6	Using a TPM as Key Generating Authority for the Shamir IBS Scheme	141
6.1	The Shamir Identity-Based Signature Scheme	142
6.2	The Fundamental Mathematics	143
6.2.1	The Initialization Phase	143
6.2.2	The Key Generation Operation	143
6.2.2.1	Using the Signer Private Key	144
6.2.2.2	Verifying a Signature	144
6.3	Why Use a TPM as KGA?	144
6.3.1	Low Overall Code Complexity	144
6.3.2	Security of the TPM Key Generation Process	145
6.3.3	Security from the Use of TPM Keys	145
6.3.4	Assurance to External Parties	148
6.4	Using a TPM as KGA for Shamir's IBS Scheme	149
6.4.1	Programming for the TPM and Key Generation Primer . . .	149
6.4.2	Generating the Master Key Pair using a TPM	150
6.4.3	Generating the Signer Private Key using a TPM	153
6.4.4	Using the Signer Private Key	155
6.4.5	Verifying a Signature Generated using a Signer Private Key	155
6.5	Real-World Implementation and Performance	155
6.6	Conclusion	157
7	Future Works and Conclusion	159
7.1	Future Works	159
7.1.1	On Hindering Forensics against Information	159
7.1.2	On Hindering Forensics in Mobile Systems	159
7.1.3	On Threat Mitigation in Disruption Tolerant Networks . . .	160

7.2	Conclusion	160
7.2.1	Hindering Forensics against Information	160
7.2.2	Hindering Forensics in Mobile Systems	161
7.2.3	Threat Mitigation in Disruption-Tolerant Networks	162
7.2.4	TPM as KGA for the Shamir IBS Scheme	162
Author's Publications		163
Bibliography		165

List of Tables

3.1	Notations Used and Basic Statements in GNY Logic	41
3.2	Entities and Data Sets in the Anti-Forensics System	43
3.3	Application Layer Block Functions	49
3.4	Attestation and Authentication Outcomes on Data Transacted . . .	52
3.5	Primitives for the Communication Protocol	54
3.6	Ideal Properties of D_I and D_F	60
3.7	Data Sealing with Previous State Consideration	61
3.8	Forensic Techniques and Anti-Forensic Outcome	79
5.1	Visibility of Components Relative to Disclosure	120
5.2	Summary of Simulation Parameters for Flooding Attacks	129
5.3	Summary of Simulation Parameters for Source Spoofing Attack . .	137
6.1	Performance Benchmark between TPM and CPU	156

List of Figures

2.1	Logical diagram of a TPM.	23
2.2	Constructing a chain of trust.	24
2.3	Sample snapshot of a PCR set.	25
2.4	Overview of the Shamir IBS Scheme.	38
3.1	Role overview of the security measures.	47
3.2	Conceptual overview of the TPM-based anti-forensics system. . . .	48
3.3	Block diagram of the anti-forensics system.	49
3.4	Communication protocol for anti-forensics system.	65
3.5	Execution flow for the anti-forensics client prototype.	78
4.1	The Android block diagram.	92
4.2	Executing Android (Android-x86) in RAM.	94
5.1	An advanced version of flooding.	106
5.2	A relay impersonation attack with acknowledgment counterfeiting. .	108
5.3	Median time for x_t and σ pair generation using different x_t gener- ation methods.	124
5.4	Terrain layout for simulation of flooding attacks.	128
5.5	Contact capacity between friendly nodes for simulation against flood- ing.	131
5.6	Contact capacity between adversary and friendly nodes for simula- tion against flooding.	132
5.7	Packet delivery rate against deliverable destination flooding. . . .	134

5.8	Terrain layout for source spoofing attacks (25 friendly nodes).	136
5.9	Packet delivery rate against spoofed source flooding attack (25 friendly nodes).	138
5.10	Terrain layout for source spoofing attacks (100 friendly nodes).	139
5.11	Packet delivery rate against spoofed source flooding attack (100 friendly nodes).	139
6.1	Initialization and key generation in the Shamir IBS scheme.	142
6.2	Generating a TPM key.	146
6.3	A TPM key hierarchy example.	148
6.4	Generate, load, and unload the KGA master key using a TPM.	151
6.5	Generating a signer key <i>bIdSigKey</i> for the identity object <i>bId</i> .	153

List of Abbreviations

API	Application Programming Interface
DER	Distinguished Encoding Rules
DTN	Disruption Tolerant Network
GRUB	Grand Unified Bootloader
IBS	Identity-Based Signature
IMA	Integrity Measurement Architecture
KGA	Key Generating Authority
MANET	Mobile Ad-Hoc Network
OOB	Out-of-Band
PCR	Platform Configuration Register
PKCS	Public Key Cryptographic Standard
SHA	Secure Hash Algorithm
TCG	Trusted Computing Group
TPM	Trusted Platform Module
TSS	TCG Software Stack
UUID	Universally Unique Identifier

Chapter 1

Introduction

1.1 Research Objective

One challenge in the use of cryptographic schemes is about how to keep their use inconspicuous, as they leave traces of their use. Such traces could lead to undesired consequences, such as coercion or torture by an adversary in an attempt to discover the cryptographic secrets employed. The author studies this issue, and as part of this dissertation engages in the design and analysis of methods and techniques for reducing the conspicuousness of traces left by cryptographic schemes and concurrently enabling its users to plausibly deny their use or rationale for their use.

The research methodology adopted is as follows. The author first identifies areas where traces left by cryptographic schemes could be used to mount pressure on users of the scheme. The author then proposes that suitable methods and techniques could be designed to reduce such conspicuousness of the traces, and further, enables plausible deniability such that users can have plausible excuses to hide the actual use of the proposed methods and techniques or the rationale of their use. Subsequently the author performs comprehensive literature survey for each of the areas outlined. This is followed by the design and analysis of the methods and schemes proper.

1.2 Motivation and Scope

Since the dawn of civilization, mankind has been employing cryptographic schemes to protect the confidentiality of information. The use of classical cryptographic schemes like the Atbash, Caesar Cipher, and Polybius Square demonstrates how civilizations understand the need for such protection schemes, though their techniques may not be sophisticated by today's standards (all the ciphers named are substitution ciphers breakable using frequency analysis).

Today, cryptography is about a lot more than just information confidentiality (encryption), and includes things like digital signatures and cryptographic hash functions among others. Nevertheless, their use almost invariably leave traces (e.g. ciphertext, digital signature). Such traces, while may be sufficiently unintelligible to an adversary, when present may lead to suspicion that certain, more valuable information may be present (although initial suspicion may not always be that cryptography has been employed). This may potentially lead to investigations being conducted to determine the composition of and information contained within the seemingly unintelligible data, which could then lead to the traces, for what they are, and the use of cryptographic schemes being discovered.

In certain situations, the discovery of traces resulting from the use of cryptographic schemes may not be desirable, as this may further lead to efforts by an adversary to obtain the cryptographic secrets to the purported cryptographic scheme(s) employed. For example, an adversary who discovers a ciphertext in a user's system may launch efforts into retrieving the decryption key for the ciphertext. Such efforts, depending on the situation, could include drastic measures like coercion or inhumane techniques like torture. However, creating cryptographic schemes that do not leave any traces at all (including outputs) is, as far as the author knows, yet possible.

An alternative line of thought would be to design methods and techniques that minimize or conceal the traces. At the same time, attempts could also be made to design these methods and techniques in such a way that provides plausible

deniability - the provision of avenues for a user to either plausibly deny the existence of the concealed traces, or if that is not possible, conceal the actual use of the schemes. For the latter, it means that when confronted about the use of some particular cryptographic schemes given the traces found, an entity, instead of merely denying that such cryptographic schemes have been employed, can for example provide plausible excuses supported by the scheme as to why the traces are present, which may even include the excuse that an alternative cryptographic scheme is used instead.

1.2.1 Hindering Forensics against Information

Computers and computing systems these days are systems that handle what are essentially information. Such information may require varying degrees of confidentiality - some required to be strictly confidential, while some others not so and can be stored as plaintext. To maintain information confidentiality, encryption schemes can be used to produce ciphertexts from the information (plaintext) for which without the correct decryption key, recovery of the plaintext from the ciphertext is rendered difficult.

A ciphertext is a natural consequence of using encryption schemes. Properly designed and used, ciphertexts produced by an encryption scheme should, among other things, not be usable by an adversary to acquire any significant information regarding the plaintext, even when analyzed together with other publicly available information of the encryption scheme¹. Such notion, known as semantic security [1, 2] or (equivalently) indistinguishability against chosen plaintext attack, could generally be thought of as follows - given two plaintexts chosen by an adversary and a ciphertext returned from the encryption of one of them with a random key, the probability of the adversary correctly guessing which plaintext of the two has been used to generate the ciphertext should not be significantly greater than

¹This would include information like how the encryption and decryption algorithms work, the public key corresponding to the ciphertext (for asymmetric schemes), or the cipher mode of operation (for symmetric schemes)

randomly picking one of the plaintexts. In other words, the ciphertext should not provide any significant information to an adversary over what it already knows about the plaintext even without the ciphertext.

However herein lies the problem - if the adversary knows for certain that encryption has been used, then the adversary knows that a ciphertext exists, thus further implying that there is a plaintext for it. Accordingly, if the adversary suspects that encryption has been used, then the adversary would suspect that a ciphertext exists.

By virtue of such rationale, an adversary, upon strongly (and correctly) suspecting the use of encryption, may attempt to attack the suspected user in order to gain information regarding the ciphertext, plaintext, and / or the cryptographic secret(s) employed. In this process, the user could be interrogated, coerced, or even tortured or subjected to violence to verify the presence of the ciphertext, and if verified its presence, to reveal any cryptographic secret(s) required to decrypt it.

In short, this leaves the user with limited options in the face of such an adversary - verify the presence of the ciphertext and disclose the corresponding secret(s), or face further hardship in the adversary's quest to gain access to the plaintext. Depending on the hardship afforded, this could eventually lead to the secret(s) being disclosed and the ciphertext decrypted. This however, is assuming that the consequence of revealing the plaintext is less dire than the hardship afforded.

What happens then if the plaintext contain information that will lead to even more dire hardship for the user?

The question then is this - is there a way out of these limited options? Could there be a third option that frees the user from hardship without leaking the secret(s) and plaintext?

1.2.2 Hinderings Forensics in Mobile Systems

The implementation of designs to protect information and communication channels would, once again, leave traces. Such traces, for example customized security software and scripts, may lead an adversary to believe that a computing system is used as a security system (on top of what it is used for), and this could thus place pressure again on the user like how if encrypted data is found. In fact, the consequence could be even greater depending on what these systems are designed for.

Concealing a system is much more difficult than concealing information. For starters, a concealed system has to be operational. This means the system should either be able to operate in spite being concealed, or designed to be surreptitiously unconcealed for operation and re-concealed once operation is complete.

Then there is the question of what to conceal. If it is just certain binaries or codes that are to be concealed, then the discovery of the container containing such concealed items may not be desirable and therefore, schemes should be designed to conceal these containers as well. If, on the other hand, it is an entire operating system that is to be concealed, then the design could be quite complex.

Again of course there is the issue of consequence for both discovery of the concealed items or the discovery of the act of concealment itself. For example, an espionage agent caught with cryptographically secured containers may find itself in serious trouble, but the alternative of revealing what is in the container may be equally as, if not more damaging.

This again begets the question - is there a way out of these limited options? Could there be a third option that first provides concealment, with a failover mechanism that permits plausible deniability if the act of concealment is about to be revealed?

1.2.3 Threat Mitigation in Disruption Tolerant Networks

In communication networks, threats come in many forms. One of them which is a major issue is of how to prevent unauthorized participation in the network. A network, especially one that communicates wirelessly, allows for any adversary with the proper equipment to access the wireless channel and potentially do things like tamper with what is sent or inject its own data into the channel.

A technique used to mitigate such threats is to render the communication channel more secure. For an insecure channel, this could mean the use of cryptography by nodes to encrypt network traffic. Therefore, any transaction would appear unintelligible to those without the cryptographic keys required to decrypt the traffic. This way, passive adversaries monitoring the channel would only be able to pick up these unintelligible data while active adversaries would find it difficult to work out how to tamper with the unintelligible data or how to correctly inject its own data into the channel in such a way that they would appear valid to the other nodes.

Security via such a technique alone therefore assumes that enabling encryption at the channel is sufficient to secure a network. This is true if an adversary cannot gain access to the cryptographic secrets required to decrypt the network traffic. However should an adversary manages to somehow gain access to the secrets, or if such security techniques cannot be employed, then the situation goes back to square one.

Therefore, an additional notion of protection would be needed - authorization - such that only entities authorized to participate in a network is allowed to do so. Naturally this requires a record of entities authorized to participate in the network, or a method to determine such, which is checked against whenever an entity requests to participate in the network.

Part of determining whether an entity is authorized to access a network requires that the authenticity of the requesting participant be determined. Such entity authentication process allows an entity to prove that it is in fact who it claims

to be using credentials that it has. Depending on the network, there may be a rule on when authentication is required. For example, a network may enforce authentication only once per session - for as long as the network session remains connected, an entity would not have to reauthenticate itself. Another network may require authentication at every communication instance. Such balance is determined by the network protocol and scheme used, but would generally be at least once when a network session is established.

In a disruption-tolerant network (DTN) however, the situation is slightly different. DTNs are opportunistic networks designed for communication in challenged environments, similar to mobile ad-hoc networks but with significant improvements. Improving on mobile ad-hoc networks, DTNs allow upper-level session connectivity to persist even under intermittent network connectivity, thereby allowing for more robust networks that are useful in environments like deep space, satellite, battlefield, or disaster zone communication. In a DTN, there is no guarantee that a session can complete without any interference leading to reconnection.

Given its nature of permitting continuous sessions over intermittent connectivity, challenges exist in verifying that a node, upon resumption of its DTN-backed session, is free from any interference by external entities. What this essentially means is that a node may be disconnected because it is being attacked or compromised by an adversary. Upon reconnecting to the DTN and resuming its session, without authentication, other nodes would not be able to verify if such incident has (or has not) happened.

Moreover, given the performance nature of DTNs, the method of authenticating per session-reconnect instance or establishing complex authentication protocols may not be applicable.

The first question is therefore this - when should authentication take place in DTNs?

DTNs can generally be classified as two broad types - hierarchical and flat topologies. A hierarchical DTN, where a few network nodes at the top of the hierarchy control a larger number of nodes at the lower layers in a tree structure, is

more suited as a strategic network since it allows real-world organizational structures to be preserved. Flat-topology DTNs, where there is little or no distinction between the nodes (i.e. each node may both be a sink and a relay), on the other hand are more suited for tactical scenarios like localized battlefields or disaster areas.

A flat-topology network used in a tactical scenario may encounter more fluid membership situations compared to a hierarchical one. For example, a battlefield-area network may have members added to it as reinforcements arrive, and this would require that other network nodes be able to authenticate the newcomers (since network nodes are both sink and relay).

Thus comes the second question - how should authentication be designed to take into account 1) the nature of such networks and its performance, and 2) the fluidity in membership?

While much research has been done on improving the reliability and performance of DTNs, the security aspect is often overlooked since network security would incur some amount of overhead. Without proper authentication procedures in place, an adversary would be able to participate in a targeted DTN, allowing them to attack that DTN as a network participant.

Of course then with authentication in place, an adversary may attempt to gain access through other means - one of them being to attack and compromise a friendly network node to gain the authentication secrets required to participate in the network. Security in this case would therefore have to take into account such maneuvers.

Trivially, this problem can be solved by constructing an impenetrable node to prevent any secrets from being extracted. For example in certain militaries, critical components in communication equipment are designed to self-destruct when intrusion is detected, thereby depriving an adversary the chance to obtain sensitive secrets contained within the equipment required to access the military communication network.

However, what happens then if an adversary not only captures the node (e.g.

a radio), but also the user directly controlling the node (e.g. the radio operator)? It would then be possible for the adversary to compel the user, for example using threats or coercion, to operate the node in such a manner as determined by the adversary, without the adversary having to obtain any secrets from the node. The nature of DTNs further exacerbates the issue - upon establishing or reestablishing a session to the DTN, it is difficult for other nodes in the network to verify that the connecting node and its user is not currently controlled by an unseen unauthorized entity.

The question therefore is this - for a node containing such secrets, is it possible to, with cooperation from its user, signal to other nodes that it is compromised or operating under duress, in a way that minimizes the suspicion of the controlling entity against the node and user? Better still, is there a way to send not just a signal, but more complex information as well?

1.3 Contributions

In this dissertation, designs are proposed to solve the issues outlined in the previous section. In particular, the major contributions presented in this dissertation are:

1. A Trusted Platform Module based system designed to hinder forensic extraction of information with provision for plausible deniability.
2. A deployment of the Android mobile operating system that hinders forensic extraction of binaries via concealment while providing the ability for plausible deniability as failover.
3. A lightweight protocol based on identity-based cryptography that facilitates node authentication in DTNs with provision for covert signaling of node state, together with further schemes for using the authentication information to mitigate threats in DTNs.
4. A method for using a Trusted Platform Module in an undocumented manner as a Key Generating Authority in the Shamir Identity-Based Signature

Scheme.

1.3.1 Hindering Forensics against Information

Given that it is yet possible to have encryption without ciphertexts, the author's approach to the problem is to design a client-server system that provides plausible explanation for the existence of the ciphertext. This entails the introduction of fake information - information that is meant to deceive and deliberately revealed under duress to support plausible deniability. The author employs the use of Trusted Platform Module enabled systems in the design, which are ubiquitous in OEM systems nowadays.

The design of the system is done based on how it could potentially be misused by a criminal to hide incriminating information from forensic examiners. This scenario is chosen based on the idea that any incriminating information possessed by a serious criminal, like fraudulent accounting information or list of illegal transactions, would be severely damaging if revealed and used as evidence.

The contributions in this section are as follows:

1. Design of a Trusted Platform Module based client-server system to hinder attempts at forensics extraction of information with provision for plausible deniability.
2. Analysis of the system and its feasibility in hindering information extraction.
3. Discussion on how the system could actually be used for legitimate purposes.

1.3.2 Hindering Forensics in Mobile Systems

At times, one may be required to use binaries that, if discovered, could lead to undesired consequences. For example, a clandestine operative found in possession of binaries used for communication with the base of operation may find it difficult to plausibly deny that the use of such binaries is not for espionage, even when the information transacted does not appear to be of such nature. In addition, the

base of operation or employer of the agent may not want to disclose its role in the entire scheme in order not to trigger greater repercussions.

The author's take at this involves a design that attempts to protect both the operative and employing organization. It involves designing a mobile operating system deployment in such a way that a user can create the illusion that the operating system is clean and free from any illegal and undesired binaries, but in actual fact that it is not the case and is perfectly capable of running them. The system incorporates a failover where the user can permanently remove such binaries in the event where total compromise is imminent. The design relies upon the ability for operating systems to run completely in memory.

The contributions in this section are as follows:

1. Design of a method that allows a user to plausibly show that an Android operating system is free from unwanted binaries, though in actuality they are hidden and are capable of being run.
2. Analysis of the system and its feasibility in hindering forensics extraction of the binaries.

1.3.3 Threat Mitigation in Disruption Tolerant Networks

The contributions in this section focuses on mitigating external threats in DTNs. Specifically, a design is proposed for authenticating network nodes and contact link construction in disruption-tolerant network with provision for a compromised node to alert other nodes covertly of its state.

The protocol for authentication and covert signaling is designed to be part of the neighbor discovery process. This enables nodes to authenticate one another when establishing contact links before any actual packet transfer is performed thus denying adversaries from fraudulently advertising beacon messages designed to ensnare other nodes into establishing connections to it. As covert signaling also takes place at this level, it further means that compromised nodes, with

the assistance from their operators, can signal their state covertly to other nodes during the neighbor discovery process.

The protocol is designed utilizing identity-based cryptography, and consideration is given to low design complexity due to the nature of DTNs.

The contributions in this section are as follows:

1. Design of a protocol that latches onto the neighbor discovery protocol in DTNs to provide for node authentication and state signaling.
2. Evaluation of routing performance when authentication information from the designed protocol is used in threat mitigation against flooding and spoofing attacks.

1.3.4 Using a Trusted Platform Module as a Key Generating Authority

A Trusted Platform Module is an industry-standard module that is bound to a system, providing secure, crypto-processor functions like asymmetric key generation and storage, asymmetric encryption / decryption, cryptographic hashing operations, and much more. Though useful as it may seem, this module, as mentioned ubiquitous in many OEM systems, is often underutilized or not utilized at all.

The author presents a way to use this module as a secure Key Generating Authority in a Shamir Identity-Based Signature Scheme implementation. It aims to demonstrate that the TPM can be used for more than what it is usually documented for, and sometimes such uses require just a small, low-complexity code footprint.

The contributions in this section are as follows:

1. Design of a method to utilize a Trusted Platform Module in an undocumented manner as a Key Generating Authority in the Shamir Identity-Based Signature Scheme.

2. Discussion on how the method presented can be used beyond the Shamir Identity-Based Signature Scheme.

1.4 Theme of Research

In designing solutions to the problems outlined, a common approach undertaken by the author is to ask - are there means to provide security in a way that complement a user's requirements, instead of having security as an imposed requirement? In the scenarios presented in the previous sections, all of them are systems where humans are ultimately the operators. While humans are sometimes called the 'weakest link' in a system, humans can do things that a digital system ultimately cannot - e.g. lie, deceive, withhold information, etc.

Existing, common methods of implementing and using security usually portray security as being 'in your face'. Users are made obvious that security is used and enforced, and the user has no choice but to go along with the imposed security requirements. Such a technique could work if the user accepts security as necessity, or if the presence of security will not be harmful to the user². However for an entity which views security as being in the way of achieving its goal, such display of imposed security could 1) provide knowledge to the adversary as to what security measures are in place, and 2) allow an adversary to not just attempt techniques to break the security, but also techniques to bypass them.

A metaphorical example would be comparing between using artilleries and land-mines (though not entirely accurate as digital security measures do not end up destroying an enemy) to defend a choke-point. Securing an area using artilleries would essentially let an enemy know that defensive measures have been deployed; the presence of artilleries being a deterrent to would-be intruders. Land-mines on the other hand are hidden; they secure an area without being visible or known to the enemy with the enemy only knowing after stepping on one (or being told about it).

²e.g. In locales where cryptography is banned, etc.

In addition, the use of security may leave traces, which may be undesirable. In situations where the presence of security may lead to severe consequences, the discovery of cryptographic traces may lead to such an outcome as well. Traces can be used against an entity to force a disclosure about the cryptographic techniques employed.

This is where the recurring theme of inconspicuous security and plausible deniability comes in - the exploitation of situations to cast doubt and ultimately convince in the user's favour where what is disclosed is not a definitive indicator, while maintaining security of the entity to be secured.

In the design of the anti-forensics system to safeguard information, plausible deniability is afforded so that the existence of incriminating information can be denied in a believable manner, with traditional security measures working in tandem to secure the confidentiality of the information and conceal its existence.

The second problem demonstrates how inconspicuous security could play a role in hindering forensics, this time against a mobile operating system. By not revealing what security measures are in place, security could be afforded for the user without the adversary knowing it, thus allowing a user to plausibly deny the existence of sensitive components.

The third problem, that of mitigating threats in DTNs, however, does reveal the security measure taken when securing the beacon message (or routing data) in order to provide authentication. However, concealed within that security measure is yet another security measure, designed to detect if the authenticating entity is truly valid and not compromised. In addition, inconspicuous use of the authentication information is demonstrated; to protect the network against flooding and spoofing attacks, without revealing to the adversary during the attack what action has actually been taken. Therefore from an adversary's point-of-view, the attack would have gone through successfully, yet it would be thwarted by the friendly node.

1.5 Dissertation Organization

The rest of the dissertation is organized as follows:

- Chapter 2 presents a literature survey of the problems identified and existing techniques to handle them, looking at the strength and weaknesses of the techniques.
- Chapter 3 details a system design to hinder forensics against information, and discusses potential positive uses for such a system. The system utilizes a Trusted Platform Module (TPM) and an integrity measurement subsystem to determine the trustworthiness of the system before decrypting incriminating information. The system engages in active anti-forensics that at the same time also allows a user to deny the presence of the incriminating information.
- Chapter 4 discusses a method of deploying Android to run in RAM to hinder forensics in mobile devices. It demonstrates how such a method would be disadvantageous to first responders in particular and forensics in general, as it is almost unlikely for first responders to know if an acquired device is actually engaging in measures to remove sensitive components.
- Chapter 5 outlines a protocol to authenticate beacon message information (which could also be used for authenticating routing data) using the Shamir identity-based signature (IBS) scheme, of which the signatures contain a hidden, translucent code that allows the verifier to know if the signer is compromised. Further, the chapter shows how the authentication information can be used to secure the network from flooding and spoofing attacks in a way that does not reveal to the adversary the defensive action taken.
- Chapter 6 demonstrates a way to use a TPM as a Key Generating Authority for the Shamir IBS scheme. It shows how such a module can be used as a security tool beyond documented techniques.

- Chapter 7 proposes future directions for research and concludes the dissertation.

Chapter 2

Literature Review

2.1 On Hindering Forensics

2.1.1 Electronic Evidence and Digital Forensics

Electronic records such as computer and network logs, e-mails, word processing files, and image files have been increasingly relied upon as essential and important evidence in criminal cases. As a result, guidelines for handling and processing have been proposed to ensure that electronic records recovered from sources adhere to the standards of evidence that are admissible in courts of law [3, 4, 5].

2.1.1.1 Digital Forensics

Electronic evidence can be defined as any information stored on or transmitted to or from an electronic device that is admissible to a court of law [6]. Due to their nature of existence, they can be difficult to destroy and easily duplicated, but at the same time they are also easily modifiable.

In order to obtain electronic records to be used as evidence, law enforcement agents conduct digital forensics examination on lawfully seized electronic sources or intercepted communication. The aim of this process is to investigate the sources and communication intercepts and retrieve information and data from them [7] in such a way that can later be admissible in court. This means that the acquisition

and retrieval must be controlled and documented in order to show that the sources and records contained within have not been tampered with in ways which would result in them being disqualified for use as evidence.

Some models and frameworks for the digital forensics process have been proposed [8, 9, 10, 11, 12], with the general aim of providing a base for scientific application of established theories and development of new ones. A review of some of these models and frameworks (and more) are provided in [12, 13]. Carrier and Spafford in [14] took a step further by identifying common concepts within the various frameworks and utilizing them as basis to compare different frameworks. In [15], Garfinkel summarizes the state of current digital forensics - suggesting that the “Golden Age” of digital forensics is about to come to an end and a digital forensics crisis emerging, and further discusses the way forward - the need for standardization of forensics data representation and forensics processing.

An interesting notion pointed out by Garfinkel is that existing digital forensics tools are designed 1) more for finding specific pieces of evidence, and 2) more for solving crimes committed against people where the evidence is digital, instead of crimes committed using a digital system against other digital system(s)¹.

Such a notion of evidence-based design can also be seen in academia, where literatures have been published on techniques (the design) to retrieve specific data or records (the evidence) (e.g. [16, 17, 18] (volatile memory recovery) [19, 20, 21, 22] (file carving) [23, 24] (network forensics)). However unlike techniques locked into digital forensics tools, the parallel development of models and frameworks for digital forensics mean that such techniques may not be limited to being applied to only on one particular model or framework.

As techniques are applied to obtain digital records or data, there is the need to determine if such records or data are admissible to a court of law. In the United States federal court system for example, this is governed by the Federal Rules of Evidence [25]. Frieden and Murray in [27] discussed the admissibility of electronic

¹“Put crudely, today’s tools were created for solving child pornography cases, not computer hacking cases. They were created for finding evidence where the possession of evidence is the crime itself.” [15]

evidence under the Federal Rules of Evidence.

2.1.2 Hinderling Digital Forensics

There has generally not been a consensus about how to categorize methods and processes used to hinder forensics. Rogers in [28], Peron and Legary in [29], and Harris in [30] have all presented their own categorization suggestions for anti-forensics respectively. Harris for example suggested the categories evidence destruction, evidence source elimination, evidence hiding, and evidence counterfeiting, to account for “distinct actions that compromise the availability or usefulness of evidence to the forensic process”.

2.1.2.1 Concealing Information

The fundamental objective of concealing information is that the information proper (i.e. the very information usable as evidence, for example a record of transactions) is rendered hidden from forensics. Information concealment, when employed properly, should hinder the extraction of information usable as evidence.

Information concealment techniques have long been used to hinder forensics examinations. Among such techniques include steganography [31, 32, 33, 34, 35], and use of covert and subliminal channels [36, 37, 38, 39, 40, 41]. Encryption schemes could also be considered as a form of information concealment, since encryption renders the information unintelligible and thus concealed without the decryption secrets.

Any information concealment technique requires a container, which could be existing (e.g. image steganography) or generated by the concealment technique (e.g. encryption). If the container is located locally within a system (e.g. alternate data stream of a local NTFS filesystem [42]), then there is always a likelihood of it being discovered when the system is subjected to forensics examination. Nevertheless with increase in network speeds and adoption of cloud computing, information concealment could also be done using off-system storage (e.g. non-local network

or cloud storage systems). Such concealment methods would therefore require application of additional techniques such as network traffic analysis and packet interception [43, 44], and network forensics [23, 24].

2.1.2.2 Data and Data Source Tampering

An alternative to information concealment, techniques to tamper with data and data sources could also be used to hinder digital forensics. Unlike information concealment which aims to conceal incriminating information from being discovered, data and data source tampering concerns the modification of data or its container in such a way that prevents the retrieval of useful information for use as evidence.

A trivial example of data tampering would be that of modifying a document's content. Changing an accounts entry to read 'PRINTER PURCHASE' from 'COCAINE PURCHASE' for example would have prevented the retrieval of the information 'COCAINE PURCHASE', something that is potentially incriminating.

Other methods of tampering could include the deliberate tampering of computer systems with the aim of altering or destroying data stored within before or during a forensic examination is conducted. Such action could be performed using software based methods such as secure erasers and data sanitizers, or more ingenious hardware techniques such as degaussing storage media or using charged capacitors to destroy critical microelectronic components when chassis intrusion is detected.

Given the stringent requirements for evidence handling, data or data source tampering could result in retrieved data not fulfilling the criteria of being admissible to courts.

2.1.3 Mobile Operating System

A mobile operating system is an operating system that runs on mobile devices, like smartphones and tablets. Common platforms in the market today include Google's Android [45], Apple's iOS [46], RIM's BlackBerry OS [47], and Microsoft's

Windows Phone [48].

Although designed for devices with different capabilities compared to a regular desktop computer, mobile operating systems may be susceptible to digital forensics, given their architecture. The Android operating system for example, is based on a Linux architecture [49, 50], and could be susceptible to Linux-style forensics (e.g. access via the Android Debug Bridge or JTAG). Guides for Android forensics can be found at [51, 52].

In [53] the authors proposed using information flow monitoring (with corresponding information flow policy) to protect against information leakage in the Android operating system. Such technique is useful to protect against applications that can be used to extract information from the operating system.

However, as mobile devices increasingly use common file systems like FAT, extraction of data is made much easier compared to when devices use proprietary file systems. Mobile devices typically use flash-memory as storage, some having them soldered on board and which may require desoldering of the chip before data acquisition. An introduction to flash memory forensics can be found at [54].

A guide for mobile phone forensics can be found at [55]. Although focused on mobile phones, the ideas presented could be adapted for other, similar architecture devices like tablets.

2.2 Trusted Platform Module and Chain of Trust

The TCG Architecture [56] is a specification developed by the Trusting Computing Group (TCG). This specification outlines the concept of trusted computing, which essentially ensures that a system will consistently behave in specific, prearranged ways, verifiable by external entities using a combination of hardware and software support.

At the core of the specification is the Trusted Platform Module (TPM) [57, 58, 59, 60], which provides security services to other hardware and software components in the platform. Within this module is a globally unique, 2048-bit RSA

key called the Endorsement Key [61] (EK) of which the private portion is out of bounds to any entity other than the module itself.

TPMs are usually implemented as microchips, or, in more recent times, integrated into other, larger silicon to enhance tamper resistivity (e.g. Intel's vPro technology). Most TPMs end up being bound to a platform with the most common (and widespread) being computer systems. This binding is usually physical in that a TPM is bound to a system either via soldering of the TPM microchip itself onto the system board, or inclusion into a larger silicon that is soldered onto the system board.

Figure 2.1 highlights the internal architecture of a TPM, which clearly shows its cryptographic components, operations and management components, volatile and non-volatile memory, and the I/O interface. The Endorsement Key (EK) resides in non-volatile memory, together with the Storage Root Key (SRK) [62] and Owner Authentication (OA). The RSA key slots, used to load and hold TPM-protected RSA keys, are located in volatile memory.

The ISO/IEC 11889 standard [57, 58, 59, 60] governing the TPM does not mandate a purely hardware implementation. It does however mandate that for a TPM to meet ISO/IEC 11889 standard, it must support at minimum the RSA, SHA-1, and HMAC cryptographic algorithms among other things. A TPM conforming to the ISO/IEC 11889 standard must also have at minimum 16 Platform Configuration Registers (PCRs). These PCRs, located within the TPM's volatile memory, are each 160 bits long and used for integrity measurement.

A TPM forms the Root of Trust for Measurement [63] of a trusted system, from which a chain of trust can be built. In a typical computer system, this chain extends from the Core Root of Trust for Measurement (CRTM) [63] up to the operating system and finally its executable application codes.

Constructing a chain of trust involves measuring the integrity of each component before they are loaded by their respective loader, as shown in Figure 2.2 (numbers showing the sequence taken during construction of the chain). Specifically, in each extension of the trust boundary, the following three steps are performed:

Figure 2.1: Logical diagram of a TPM.

1. Measurement of the target code through SHA-1 hashing.
2. Registration (extension) of the measurement into a PCR in the TPM.
3. Transfer of execution control to the target code.

At any one time, a system's state can be established by its set of Platform Configuration Register (PCR) values, and using those values, a decision can be made on whether or not a system is free of undesirable yet activated software which in turn forms a basis on whether or not the system can be trusted.

Extending a measurement into a PCR is a TPM native operation, and thus there is negligible risk of it being tampered by external entities. Equation 2.1 shows a PCR extend operation:

Figure 2.2: Constructing a chain of trust.

$$\begin{aligned}
 \text{PCR}_x[0] &= \text{INIT}_{\text{TPM}} \\
 \text{PCR}_x[i] &= \text{SHA-1}(\text{PCR}_x[i-1] \parallel m_i)
 \end{aligned}
 \tag{2.1}$$

In Equation 2.1, INIT_{TPM} is the initial value of PCR x , and m_i is the individual executable measurement (SHA-1 hash) for the i -th executable that is measured and loaded. From the equation, it can be seen that the PCR extend operation is recursive and the resulting PCR value is affected by all extended codes since the PCR is initialized. Such hash chaining ensures that any changes in the hash value or execution order of the earlier (i.e. upstream) executables will be reflected downstream in the PCR readings.

Although ISO/IEC 11889 mandates at least 16 PCRs, a typical TPM version 1.2 has 24 PCRs. Figure 2.3 shows a sample snapshot of a PCR set. Note the values 0x00..00 and 0xFF..FF. These are values of freshly initialized PCRs (whether

the PCR contain a sequence of 0x00 or 0xFF depends on the PCR number).

PCR-00:	A8	85	DD	E3	36	48	38	36	54	76	47	C2	C1	61	3F	22	49	02	01	44
PCR-01:	5B	93	BB	A0	A6	64	A7	10	52	59	4A	70	95	B2	07	75	77	03	45	0B
PCR-02:	5B	93	BB	A0	A6	64	A7	10	52	59	4A	70	95	B2	07	75	77	03	45	0B
PCR-03:	5B	93	BB	A0	A6	64	A7	10	52	59	4A	70	95	B2	07	75	77	03	45	0B
PCR-04:	F3	CC	8C	58	94	01	CF	7D	DA	7C	93	83	20	28	E5	30	4F	E8	71	BB
PCR-05:	2D	E0	2A	45	AE	96	0C	CE	AB	0F	B0	D5	0E	C0	6D	85	8B	BB	B0	A5
PCR-06:	5B	93	BB	A0	A6	64	A7	10	52	59	4A	70	95	B2	07	75	77	03	45	0B
PCR-07:	5B	93	BB	A0	A6	64	A7	10	52	59	4A	70	95	B2	07	75	77	03	45	0B
PCR-08:	7A	95	FA	BF	58	DB	F9	92	1A	66	BF	FC	4C	61	EB	8C	05	EB	A3	4C
PCR-09:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
PCR-10:	58	48	59	14	C4	7B	69	DE	0A	9B	DE	02	0C	E9	3F	95	D4	00	2F	AB
PCR-11:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
PCR-12:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
PCR-13:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
PCR-14:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
PCR-15:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
PCR-16:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
PCR-17:	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
PCR-18:	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
PCR-19:	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
PCR-20:	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
PCR-21:	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
PCR-22:	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
PCR-23:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

Figure 2.3: Sample snapshot of a PCR set.

2.2.1 Remote Attestation

Remote attestation is the ability for a remote entity to verify the integrity of a target machine. The entire process involves two machines - the challenger and the verifier. A sample remote attestation protocol is provided in the TCG Architecture Overview [56]².

The verifier's task is to gather its platform configuration and credentials, and transmit them to the challenger. It is implied that the challenger has means to independently validate the platform configuration of the verifier upon receiving

²Refer to Section 4.2.5.1, pp. 9-10 of [56].

them. Based on the validation outcome, the challenger can subsequently decide on the integrity of the verifier.

2.2.2 Data Binding and Data Sealing

A TPM has two methods of encrypting data - data binding and data sealing. They are outlined in the TCG Architecture Overview [56]³.

Data binding is the encryption of data to a TPM-protected TPM binding key. Bound data can be decrypted only by the binding TPM, unless the binding key is migratable and is migrated to other TPMs, thus allowing the bound data to be decrypted on those other TPMs.

Data sealing encrypts data to both the TPM as well as to the current state of the platform. Since the state of the platform is represented by the values of the Platform Configuration Registers (PCRs), data sealing requires the PCR numbers to be specified during encryption. The encrypted blob can subsequently only be decrypted if the platform state during decryption matches that during encryption, which implies that the PCR values of the specified PCRs must match. In addition, as sealing keys cannot be migrated, encrypted data can be decrypted only on the encrypting machine.

Data sealing is a very powerful concept in trusted computing. Since sealed data is bound to a specific TPM and platform state, the destruction of the TPM or an irreversible change in the platform state will render all data sealed to the TPM and platform state combination unrecoverable.

2.2.3 Integrity Measurement Architecture

The Integrity Measurement Architecture (IMA) [64] is an open source software architecture and implementation first developed by IBM for Linux on TPM-enabled platforms. It provides verifiable evidence regarding the current run-time of a measured operating system. This can then be used by other systems to make an

³Refer to Section 4.2.6.1, p. 15 and Section 4.2.6.3, pp. 15-16 of [56].

informed decision on the trustworthiness of this measured system.

As a continuation of the chain of trust at the operating system / application space, IMA measures each executed code and extends the measurement to a PCR (default being PCR 10) before execution proper. Such measurement and extension conform to the steps listed earlier in Section 2.2 and Equation 2.1. In addition to that, the individual measurements are also logged into a measurement list which is accessible via a *securityfs* pseudo-filesystem. This list is integrity-protected by the TPM at all times, and safeguards exist to prevent tampering of the measurement list.

The Integrity Measurement Architecture guarantees that all codes executed on the system, and other non-executable files specifically requested to be measured by an executable, are measured and appended to the measurement list. It also guarantees that the list of accumulated measurements is tamper-evident. However, due to its non-reactive nature, IMA does not and cannot prevent a system from becoming compromised. Therefore, the system should always be assumed to be compromised, with the individual measurements themselves (via the measurement list) and TPM (specifically the PCRs) integrity value used to reason otherwise.

Prior to version 2.6.30 of the Linux kernel, IMA is applied as a kernel patch. Version 2.6.30 onwards introduces built-in support for IMA, although this is not enabled by default.

2.3 On Threat Mitigation in Disruption Tolerant Networks

2.3.1 Disruption Tolerant Networks

Disruption / Delay tolerant networks (DTNs) [65, 66, 67], are networks designed for communication in challenged environments. Deployed as an overlay, they allow network functionalities to persist even under intermittent underlying connectivity, thereby allowing for more robust networks that are useful in challenged environ-

ments like deep space, satellite, battlefield, and disaster zone communication.

A DTN overlay consists of, among other things, a Bundle Layer [65]. This layer forms (and manages) the store-and-forward capability of a DTN that permits communication across disruptive / delay-prone underlay networks. Like a transport layer, it manages the handling and transfer of bundle across DTN nodes, and optionally, acknowledgment between endpoints. These network actions are governed by a bundle protocol [68]; its protocol data unit known as a bundle.

A bundle consists of two or more “blocks” of data. These blocks could either contain information used for destination delivery, or more importantly, data from applications utilizing the DTN. Bundles themselves can be fragmented, and each fragmented bundle is itself a bundle. Likewise, bundles can be merged to form bigger bundles. Therefore, mechanisms exist at the bundle protocol to fragment and merge bundles together as well as manage them as they traverse across the DTN.

While the Bundle Layer manages bundles, it relies on DTN routing protocols for handling data transfer between DTN nodes.

2.3.2 Routing in Disruption Tolerant Networks

Routing in DTNs are governed by the routing protocols used. There are broadly two categories of routing protocols - forwarding-based schemes and replication-based schemes. In forwarding-based schemes, only a single copy of each message or fragment is present within the network. This results in less transmission and storage overhead network-wide, and no cleanup requirements after delivery since only one copy of the message exist. However, since the protocol relies only on one copy of a message, how well the protocol performs would generally depend on how well the protocol’s algorithm forwards the message. [69, 70, 71, 72] describe a few forwarding-based schemes, though some schemes may apply replication strategies for routing information dissemination [71].

In replication-based schemes [73, 74, 75, 76, 77, 78, 79], a message is replicated

into multiple copies and flooded into the network. Such a scheme introduces redundancy, and therefore increases transmission and storage overhead. The rationale for such schemes is that with more replicas, a message would have more path options to reach its destination, thus lowering the delivery latency and increasing the chances of successful delivery. Nevertheless with replicas in the network, it means that it could be harder to clean up remaining replicas after successful delivery of a message. The performance of such protocols generally depends on balancing the network capacity with the replication strategies employed to obtain optimum delivery performance.

2.3.3 Threat Model

Threats to DTNs come in many forms. In [80], Farrell and Cahill identified the following threats to DTNs:

1. Bundle modification, injection.
2. Unauthorized access, use of DTN resources.
3. Denial of service attacks (e.g. amplification attacks).
4. Threats against confidentiality and integrity.

Modification and injection attacks are what Farrell and Cahill term as the “usual range” of potential exploits. Such exploits can take place at the underlying network layers on nodes both physically part of or not part of the DTN (recall that a DTN is an overlay network), assuming those layers and nodes are exploitable. These attacks, according to them, would then have to be mitigated by the DTN.

The issue of unauthorized access and use of DTN resources is a major one considering the resource-scarce nature of DTNs - primarily due to the immediate reduction in available resources caused by such actions, and also that the act of gaining unauthorized access may consume significant DTN resources. In addition, an adversary who gains unauthorized access would be able to launch attacks from

within the network, further decreasing the available resources for legitimate DTN communication.

Another major concern is that of denial of service attacks. Such attacks include the amplification attack [81], where loopholes within a network protocol could be exploited to repeatedly generate traffic up to a point that chokes the network. An example given in [80] is how an infinite bundle storm could happen by exploiting a forwarding report to generate another forwarding report (which loops infinitely).

Attacks against confidentiality and integrity, though not specific to DTNs, is an issue considering such attacks could result in for example information leakage (confidentiality breach) and wasted resources (integrity tampering).

Though the threats identified in [80] appear more focused towards bundle protocol security, it can be said that the threats are not unique to that particular layer in DTN security. Attacks like data injection for example, could be directed towards the routing layer to falsify route decision-making. However given the nature of DTNs as an overlay network, attacks directed at and into the DTN (e.g. packet injection, routing data modification) would almost certainly involve ingress at the lower layers.

Security in disruption-tolerant networks (DTNs) therefore can be thought of as falling into two categories, namely bundle protocol security and routing protocol security. Bundle protocol security in DTNs deals with providing security at the bundle protocol for the transacted bundles between communicating sender-receiver DTN node pairs; where such pairs may not be next-hop neighbors in the DTN topology (e.g. source to destination, security-aware router to security-aware router, etc.).

Routing protocol security on the other hand deals with the security of DTN routing protocols, and involves issues pertaining to secure and reliable DTN routing. This would include protecting the DTN from routing-level threats like routing data injection and node impersonation that may happen due to security lapses at the DTN routing protocol or lower.

2.3.4 Bundle Protocol Security

In addition to outlining the threats to DTNs, Farrell and Cahill [80] also outlined some open issues for bundle protocol security. First among them is the level of flexibility of the security protocol, though the complexity of the solution should be taken into account as well. Second is issue of key management, as traditional key management schemes may not suit DTNs due to for example the number of round-trips required (e.g. TLS) or the requirement for permanent, trusted keys to be used which limits scalability.

The Bundle Security Protocol Specification [82] outlines a security protocol for the Bundle Protocol, to be implemented in “security-aware” nodes. It defines four types of security blocks which may be included in a bundle, shown as follows (note that the term hop in the following describes a transfer between security-aware nodes, which may actually consist of multiple hops that traverse through non-security-aware nodes):

Bundle Authentication Block (BAB) Used for verifying the authenticity and integrity of the bundle between a single hop.

Payload Integrity Block (PIB) Used for verifying the authenticity and integrity of the payload from the PIB originator (i.e. the node that added the block).

Payload Confidentiality Block (PCB) Used for indicating that the payload has been encrypted at the PCB originator.

Extension Security Block (ESB) Used for providing security for non-payload blocks.

Seth and Keshav [83] presented a security architecture for end-to-end communication based on a threat model that 1) rogue DTN routers may masquerade as valid DTN nodes, 2) DTN routers may be hijacked and compromised, though will

be detected eventually (though the authors did not specify, nor propose any suitable detection schemes), 3) end-systems can be hijacked or turn malicious, and 4) wireless communication can potentially be eavesdropped and its security broken.

The solution presented in [83] involves the use of hierarchical identity based encryption (HIBE) and hierarchical identity based signature (HIBS) schemes. The architecture primarily focuses on the protocol designs, with the authors not specifying any cryptographic schemes to use in particular and generally assuming that the HIBE and HIBS schemes will work as intended. Instead of key revocation, the solution utilizes periodically expiring keys - realized simply by generating keys on an identity concatenated with the validity period. This means that an entity would have to obtain new keys periodically from the key generating authority.

Given the provision of security for end-to-end communication over an insecure channel, the architecture entails some complexity in its design, for example round-trip communication in certain situations. Moreover, there are some issues unaddressed, namely the detection of hijacked and compromised routers, verification of an entity's claim to an identity (the authors assume that the authority would be able to determine), and the benefits of using the architecture in authentication compared to using certificates with short-duration expiry.

Asokan et. al. in [84] answered the outstanding issue in [83] through comparing with short-lived certificates, and further compared the use of identity-based cryptography (IBC) with traditional cryptography for end-to-end message transfer. Firstly, they concluded that short-lived certificates are as effective as identity-based cryptography for authentication in DTNs. Then, the authors constructed two different protocols for private messaging in disconnected environments - one using IBC and the other using traditional cryptography. Comparing these two protocols, the authors conclude that the IBC protocol is more efficient, particularly in terms of the server load and network connectivity requirements for receivers.

In addition, [84] proposed a solution to the issue of verifying an entity's claim to an identity. This is done by using existing security infrastructure and a bootstrap mechanism.

As an extension to [84], [85] further proposes a scheme to authenticate fragments of bundles using an improvement of the ‘toilet-paper approach’ (attaching a signature to each fragment). The approach proposed uses a binary hash tree to minimize the number of signatures required at the expense of increased hashing. Since hashing is supposed to be more efficient than signing, the approach is claimed to perform better, though only an evaluation on the number of computations required are presented.

In [86], Kate et. al. presented a security solution that is shown to be more efficient than the end-to-end security architecture presented by Seth and Keshav [83]. Incorporating the key agreement scheme by Sakai et. al. [87], the authors also showed that the solution can be extended to provide pseudonym-based end-to-end anonymity without change in setup, although this requires the presence of a trusted gateway to map receiver pseudonyms for communication.

As DTNs are used to transfer multiple bundles simultaneously, the need for optimized security further comes into the picture. Individual bundle security, while appropriate for small-scale traffic, may not be scalable due to the overhead incurred per security check or enforcement. Zhu et. al. [88] proposed a Batch Bundle Authentication Scheme, which improves performance by authenticating bundles in batches instead of individually at security-aware forwarders. Using an identity-based batch signature scheme, its goal is to weed out bad bundles as early as possible within the network. Nevertheless, the protocol’s performance is affected when message injection happens due to the weeding out of injected messages from the bundle batches.

Other methods for providing security to DTN bundles are described in [89, 90, 91].

2.3.5 Routing Protocol Security

A fundamental characteristic of disruption-tolerant networks is the lack of an established end-to-end path during communication and reliance on opportunistic

connectivity for packet delivery, where even without an established path, routing is performed such that a packet eventually gets delivered to its destination by transferring it from one DTN node to another. Such nature of DTNs makes their security requirements in general, and in particular routing protocol security, different from other forms of network. In fact, [92] lists routing protocol security as an open issue in DTN security.

In the previous subsection, an overview of security at the bundle protocol is provided. The security schemes mostly utilize identity-based cryptography in a bid to minimize end-to-end communication. The lack of established end-to-end paths during communication is shown not to be a great hindrance for security at the bundle protocol. This is because for the most part, the routing of packets appears invisible to the bundle protocol (the focus of minimizing end-to-end communication in bundle protocol security is for saving DTN resources). It does however poses a challenge for security at the routing protocol. Without established end-to-end paths (and its corresponding path discovery process, e.g. like in MANETs [93, 94]), enabling routing protocol security through such avenue would not be applicable. Routing protocol security would instead have to look at securing the route hops, culminating in a network of secured hops as an alternative to achieve end-to-end, temporally out-of-sync path security.

On the one side, since there is no end-to-end path setup in DTNs, attacks that rely on disrupting such process (e.g. [95, 96, 97]) would no longer be possible. However, such lack of end-to-end path setup could also open DTNs to new forms of attack. In a simple example, an adversary in a traditional MANET impersonating an existing node and placing itself within the same path as the impersonated node would yield suspicion due to the dual-appearance of a node identity within a single end-to-end path. In a DTN however, without complete path information, the adversary could potentially insert itself without any problems at all; a second encounter with the actual node may probably be assumed to be the node moving from one location to another. Such flexibility afforded to an adversary could lead to further damage.

In [98], Choo et. al. studied the robustness of DTNs against routing attacks. The evaluation was in response to [99], which shows that attacks on DTNs (particularly the MaxProp routing protocol) are rendered to be akin to simple network failures where it could be sufficiently mitigated by replication-based routing protocols. Nevertheless the evaluation is limited to only the MaxProp routing protocol, though it provides an insight on how two types of attacks - impersonation and non-deliverable packets, could seriously affect that protocol.

2.3.6 Disruption-Tolerant Network Neighbor Discovery

Nodes in a DTN inform one another of their presence via the neighbor discovery process [100] (updated [101]). This is usually handled by a discovery agent, which:

- Advertise a DTN node's available convergence layer via the broadcasting, multicasting, or unicasting of beacon messages at certain intervals.
- Process incoming beacon messages to allow the establishment of contact links with other DTN nodes.

Each node participating in a DTN is identified via an endpoint identifier (EID), which must be unique to that node within the DTN. This EID would naturally be sent along with the beacon message so as to allow beacon receivers to know which node the advertised convergence layers belong to.

When a beacon message is picked up by receiving DTN nodes, they can use the information about the convergence layer(s) contained within the message (e.g. network address, layer type, etc.) to establish link(s) with the advertiser. Due to the constraint of bandwidth however, no acknowledgment is sent back to the advertiser upon receipt of a beacon message or establishment of a link. Since there is lack of acknowledgment, a node would not know if other nodes have picked up its beacon advertisement.

Beacon messages are therefore advertised frequently via as many convergence layers as possible to compensate for the non-persistent connectivity so as to allow

as much information as possible about the network topology to be picked up by other nodes. The lack of acknowledgment is therefore offset by the idea that other nodes would likely pick up a node's beacon message at one time or another. Moreover, as other nodes may also be transmitting beacon messages, receiving one may signal to a node that its own beacon may have also been received at the other end.

Each DTN node would have a table of links based on information parsed from received beacon messages. Given the dynamic nature of DTNs, such table would be updated frequently with the arrival of new beacon messages. This table of contacts could then be used by the routing protocol when delivering packets to other DTN nodes.

As beacon messages are used to determine presence and connectivity, they are valid targets for attacks by adversaries seeking to spoof network participation. Once in the network, adversaries can proceed to launch attacks on the routing protocol and / or beacon protocol.

While security for the neighbor discovery process and beacon messages are sparsely discussed in [100, 101], inspiration for protecting beacon messages can be gained from techniques used to protect other existing network protocols.

The IEEE 802.11w standard [102] has been proposed to protect 802.11 wireless networks from spoofing attacks. Such protection is afforded via cryptographic protection (using message integrity code) of the Deauthentication and Disassociation frames, and Security Association (SA) teardown protection mechanism for Association and Authentication requests, in addition to protecting certain Action Management Frames. It relies on the use of IEEE 802.11i [103] (implemented as WPA2), since the Pairwise Transient Key (PTK) is derived from the protocol's 4-Way Handshake process between the access point (AP) and client station (STA).

What this means however is that 802.11w does not afford protection before the PTK is created. Therefore, attacks such as the DoS attack in [104] could still work on the network. In addition, although disassociation attacks can be mitigated with 802.11w due to the protection of Disassociation frames, an attack

with similar effects to the disassociation attack has been demonstrated in [105] by means of a forged open system authentication frame.

An interesting concept presented in the IPv6 Secure Neighbor Discovery [106] is that of using Cryptographically Generated Addresses (CGA) [107], in conjunction with timestamps, nonces, and RSA signatures to prevent replay and provide authentication support. CGAs are IPv6 addresses that are associated with a public key. This is done by generating a cryptographic hash of the public key, and using that hash as the right-most 64 bits (the interface identifier) of the IPv6 address.

CGAs provide a form of identity protection, in that an adversary cannot spoof signed messages purportedly originating from the address owner. Adversaries can, however, create their own IPv6 CGAs that contain a combination of any subnet prefix of their choice and their own public key.

2.4 Identity-Based Signature Scheme

Identity-based signature (IBS) schemes are asymmetric-key signature schemes that uses the identity of the signer as the public key in the verification process. Unlike traditional digital signatures schemes, there is no need for a verifier to obtain the signer's public key prior to verification since that key is deducible from the signer's identity.

Given that there is no need to obtain and store individual public keys, IBS provides certain advantages over traditional digital signatures, notably elimination of the need for public key infrastructure during the signing and signature verification operations, the ability to discard key generation secrets and eliminate further key generation, the ability to hide signer-verifier relations by virtue of only having to store a set of public components for verification, and the prevention of disclosure of information regarding the group or its members (e.g. number of group members, etc.).

IBS was first presented by Shamir in [108], based on the hard problem of integer

factorization. The signature scheme has been proven secure against forgery under chosen-message attacks [109]. An overview of this scheme, overlaid on Figure 2 in [108], is presented in Figure 2.4.

Unlike traditional asymmetric-key signature schemes, the signer in an IBS scheme cannot generate its own key-pair. Instead, IBS schemes require the presence of a key generation authority (KGA) whose sole functions are to:

- Generate the signer's private key (g for the Shamir IBS scheme) corresponding to the signer's identity i .
- Distribute / publish the common public parameters to all verifiers (n , e , and $f(\alpha, \beta)$ for the Shamir IBS scheme).

Upon completion of its functions (i.e. key generation for all signers and publishing of the common public components), the KGA can be retired. Given the enormous responsibility of a KGA, it is important that the KGA be a secure and trusted authority.

Figure 2.4: Overview of the Shamir IBS Scheme.

Chapter 3

Hindering Forensics against Information

Electronic records such as computer and network logs, e-mails, word processing files, and image files have been increasingly relied upon as essential and important evidence in criminal cases. As a result, guidelines for handling and processing have been proposed to ensure that electronic records recovered from sources adhere to the standards of evidence that are admissible in courts of law (e.g. in [3, 4, 5]). In order to obtain these records, law enforcement agents employ forensic examination of seized electronic sources. While thus far, forensics have proven useful in extracting incriminating evidences, it is possible to develop a system that hinders this effort.

The TCG Architecture [56] is a specification developed by the Trusting Computing Group (TCG). This specification outlines the concept of trusted computing, which essentially ensures that a system will consistently behave in specific, prearranged ways, verifiable by external entities using a combination of hardware and software support.

At the core of the TCG specification is the Trusted Platform Module (TPM) [57, 58, 59, 60], an industry standard (ISO/IEC 11889) secure crypto-processor that can generate and securely store information such as keys and cryptographic

secrets. It is typically deployed as a microchip affixed to the motherboard of a computer, though the ISO/IEC 11889 standard governing the TPM does not mandate a purely hardware implementation.

While the TCG specification and TPM could have many positive uses such as in the realms of digital rights management, integrity verification, and remote system auditing, it is also possible that they be misused by criminals to encrypt data in such a way that effectively denies its access to law enforcement agents even with forensic aid. In addition, it is also possible for such systems to conceal the existence of encrypted data. Since the keys are never exposed to the criminal, it is practically impossible for enforcement agents to coerce or force a criminal to reveal it. Such an undesirable though possible usage has been conveyed to some law enforcement agencies by Leong [110] in 2006.

This chapter presents specifications and analysis of an anti-forensics system constructed by utilizing TPM-enabled security on a client-server system. It provides the system specifications and detailed analysis of its anti-forensic capabilities. Although presented as a criminal-centric system, this work is important to the field of digital forensics as it helps to answer the following questions:

- What are the requirements for such anti-forensics system, and how could such a system be realized?
- What are the implications for law enforcement agents if such a system is used in real-world crime scenarios?
- Are there any positive uses for such a system?

3.1 System Objectives and Requirements

3.1.1 Anti-Forensic Measures

The system employs a combination of three anti-forensic techniques to achieve its anti-forensic capability, namely:

- Information confidentiality
- Information and action hiding
- Plausible deniability

For design verification and analysis in subsequent sections, the anti-forensic techniques shall first be represented mathematically. A combination of GNY logic [111] and predicate logic is used for this purpose. Notations used and basic statements in GNY logic are listed in Table 3.1. Logical postulates derived from the basic statements can be found in [111].

The entities and data sets involved in the anti-forensics system are described in Table 3.2.

Table 3.1: Notations Used and Basic Statements in GNY Logic

Notation / Basic Statement	Description
$+K$	The public key of the asymmetric key pair K .
$-K$	The private key of the asymmetric key pair K .
$P \triangleleft X$	P is told formula X .
$P \triangleleft *X$	P is told formula X which it did not prior convey within the session (X did not originate from P) ¹ .
$P \ni X$	P possesses, or is capable of possessing, formula X .

Continued on next page...

¹ X could have been conveyed by P in a previous session.

Table 3.1 – continued from previous page

Notation / Basic Statement	Description
$P \mid \sim X$	P once conveyed formula X .
$P \mid \equiv C$	P believes, or is entitled to believe, that statement C holds.
$P \mid \equiv \#(X)$	P believes, or is entitled to believe, that formula X is fresh.
$P \mid \equiv \phi(X)$	P believes, or is entitled to believe, that formula X is recognizable.
$P \mid \equiv P \xleftrightarrow{S} Q$	P believes, or is entitled to believe, that S is a suitable secret for P and Q .
$P \mid \equiv \xrightarrow{+K} Q$	P believes, or is entitled to believe, that $+K$ is a suitable public key for Q .

3.1.1.1 Information Confidentiality

The system maintains information confidentiality by controlling the disclosure of incriminating information, revealing it only when the system is in a trusted state.

Let the universe of discourse representing the system states be composed of two non-overlapping sets - the trusted state set T , and the non-trusted state set T' . Information confidentiality holds within the system when:

$$(Y \in T) \longleftrightarrow (C \ni D_I) \quad (3.1)$$

Table 3.2: Entities and Data Sets in the Anti-Forensics System

Entity / Data Set	Description ^a
L	The law enforcement agent.
Z	The criminal.
C	The user-controlled client portion of the anti-forensics system.
S	The server portion of the anti-forensics system.
Y	The anti-forensics system, encompassing both client and server.
D_I	The data set containing incriminating information.
D_F	The data set containing decoy information.

^aAny other principals, data sets, or entities shall be defined as and when required.

i.e. the client C is capable of possessing the incriminating information D_I if and only if the anti-forensics system Y is in a trusted state (T).

Note that Equation 3.1 does not specify anything about what happens to D_I after being disclosed at C , or to whom D_I is disclosed (i.e. who is actually controlling C ; the criminal Z or the law enforcement agent L). Therefore, consideration must be given to hinder D_I from being visible to unintended users or beyond the duration of which the system is in a trusted state.

3.1.1.2 Information and Action Hiding

Information and action hiding is defined such that when subjected to a forensic examination, the system shall provide no indication that any data containing incriminating information has been present or manipulated on it.

The following is assumed when considering the capability of law enforcement agents:

$$L \models \phi(D_I) \tag{3.2}$$

Let the data manipulation function be F , the aim of information and action hiding can therefore be represented as follows:

$$\begin{aligned} \neg(L \ni D_I) \\ \neg(L \triangleleft F(D_I)) \end{aligned} \tag{3.3}$$

The first statement in Equation 3.3 is straightforward - that law enforcement agents should not be able to possess the incriminating data D_I . The notion ‘possess’ instead of ‘seeing’ is used, since only when law enforcement agents possess D_I that they can use it as evidence.

The second statement states that they should not see the act of data manipu-

lation².

3.1.1.3 Plausible Deniability

The notion of plausible deniability in the context of the system is such that the system shall provide the capability for its intended user (in this case the criminal Z) to plausibly deny the existence of incriminating information.

Although the previous two requirements should suffice to ensure that D_I or its manipulation are not disclosed to law enforcement agents, it does not allow the criminal to project the non-existence of D_I . Without this ability, the existence of D_I can still be suspected from the perspective of the law enforcement agents. This can be represented by:

$$L \models [\Pr(\exists D_I) = \alpha] \quad (3.4)$$

Nevertheless, since the criminal owns D_I and knows its contents, it is possible for the criminal to create a data set D_F that looks like, and is as manipulatable as D_I , but contains no incriminating information. This data set D_F could then be presented as a decoy in an attempt to deceive law enforcement agents into believing that D_F is the actual D_I .

However, it is also possible that the law enforcement agents suspect the existence of D_F . Such instance could be written as:

$$L \models [\Pr(\exists D_F[D_F \neq D_I]) = \beta] \quad (3.5)$$

Note that α and β cannot be precisely quantified as they are influenced by real-world factors (e.g. available information, competency of law enforcement agents, etc.). Consequently, there is also no constant value that could be used as threshold such that law enforcement agents cease to believe the existence of D_I or D_F .

²If F is a function, $F(X)$ is the act of manipulating X with F .

Assuming that the anti-forensics system only has the data sets D_I and D_F , the aim of plausible deniability within the context of the system is therefore to reduce the value of β in Equation 3.5 when D_F is disclosed to the law enforcement agents, thereby leading them to lessen their belief that D_F is not D_I compared to when before D_F is disclosed. That is, the law enforcement agents' belief should be such that:

$$\Pr(D_F \neq D_I \mid L \triangleleft D_F) \leq \Pr(D_F \neq D_I) \quad (3.6)$$

Taking Equation 3.6 further, a stronger aim of plausible deniability within the context of the system would be to make the left hand side of Equation 3.6 close to 0. In such a case, when the law enforcement agents are told D_F , Equation 3.5 can be simplified to be:

$$L \models \neg(D_F \neq D_I) \quad (3.7)$$

In addition to Equation 3.7, the criminal should also believe that the law enforcement agents believe Equation 3.7. This can be written as:

$$Z \models L \models \neg(D_F \neq D_I) \quad (3.8)$$

3.1.2 Legal Consideration

Information confidentiality is a fundamental requirement for all secure information systems. There are countries that afford protection against self-incrimination [112], which could include protection against disclosure of encryption keys to law enforcement agents that would be used to decrypt suspected data [113]. However, some countries (e.g. United Kingdom³) make it illegal to refuse decryption of data

³The Regulation of Investigatory Powers Act 2000, Part III.

or disclosure of decryption keys for use in a criminal investigation [114, 115].

To counter such problems of key revelation by force or coercion, techniques for information and action hiding as well as mechanisms for plausible deniability are required for the system to avoid disclosing incriminating information (albeit illegally) in such legal environments. With information and action hiding, there should be no indication that data containing incriminating information was present or manipulated, and therefore, no prima facie for refusal to decrypt data.

In the event that evidence of data manipulation (the act of, not the information) is somehow discovered by law enforcement agents, employing plausible deniability techniques allow for denial that the actual incriminating information exists, thus resulting in no prima facie for possession of incriminating information.

A flowchart depicting the roles of the security domains as outlined is shown in Figure 3.1. When properly implemented, the system should be able to avoid reaching the compromised state.

Figure 3.1: Role overview of the security measures.

3.2 System Design

3.2.1 High-Level Architecture

Figure 3.2: Conceptual overview of the TPM-based anti-forensics system.

A conceptual overview of the anti-forensics system is presented in Figure 3.2. The complete system consists of a client, server, and out-of-band (OOB) device. The latter notifies its livelihood to the client through a separated channel (e.g. infrared, Bluetooth, NFC). The client is where all data manipulation takes place, while the remote server is used to store encrypted data after manipulation. The server is assumed to be in a remote location out of the law enforcement agents' jurisdiction. To be in a trusted state, the OOB device must be detected by the client, and the client must be successfully attested by the server.

The system block diagram is depicted in Figure 3.3. Table 3.3 describes the functions of the application layer blocks that form the anti-forensics system.

Figure 3.3: Block diagram of the anti-forensics system.

Table 3.3: Application Layer Block Functions

Component	Description
Authentication / Attestation Module	Retrieves from and sends to server authentication and attestation information (client). Performs authentication of client and attestation of client state, and returns result of authentication and attestation to client (server).
Monitor Agent	Monitors the presence of the out-of-band device, and takes appropriate action (e.g. client termination) when device presence is not detected.

Continued on next page...

Table 3.3 – continued from previous page

Component		Description
Notification Agent		Notifies the user about the trust status of the client (whether the client in a trusted state). In a real-world scenario, this notification should be discreet and known only to the intended users.
Data Manipulation Module		Facilitates data manipulation for users, performs data sealing and unsealing, and coordinates transfer of sealed data with server.
Data Storage Handler		Handles storage and retrieval of sealed data when requested by the client. Relies on information about the client state from the Authentication / Attestation Module to determine if data transfer involves incriminating or decoy data.

The system utilizes two data sets - one containing incriminating information D_I and one containing decoy information D_F . Detailed description of the data set properties are provided in the subsequent subsections.

3.2.2 Design Considerations

The anti-forensics system is developed from the perspective of a criminal, and how it could be used by criminals to store and manipulate data containing incriminating information while hindering forensic examinations by law enforcement agents.

The core of the system is a TPM-enabled client running Linux with Integrity Measurement Architecture (IMA). The client builds a chain of trust and measures every individual code executed on the system since booting up. As a consequence, its state can be established at any time by reading the IMA measurement list and

Platform Configuration Register (PCR) values.

The remote server must authenticate and attest the client prior to committing every data transaction. The conditions for authentication and attestation are as follows:

Authentication The client is said to be authenticated if and only if it is verifiable by the server via the pre-distributed client TPM signature public key. Successful authentication is the prerequisite for attestation.

Attestation The client is said to be successfully attested if and only if it is authenticated and its chain of trust corresponds to a valid, trusted chain of trust verified by using the pre-distributed client clean measurement list, with all external components accounted for. Successful attestation is a prerequisite for the system to be in a trusted state.

It is assumed that a copy of the client's TPM signature public key and trusted state measurement list would be transferred to the remote server via a trusted channel during system initialization. Likewise, the server's signature public key would also be transferred to the client during that time.

The server terminates immediately the transaction if authentication fails. However to provide for plausible deniability, the transaction would continue if attestation fails (e.g. OOB device not detected). In this instance, the system runs in a non-trusted state and instead of D_I , the decoy data D_F would be used instead. A summary of the data transacted for every possible authentication-attestation combination is shown in Table 3.4.

The OOB device serves two functions - one as a tripwire for successful attestation, and another as a threat signaler to the client. The first function has been discussed. As for threat signaling, an abrupt termination of the OOB device communication while the client is in operation would signal to the client that the trusted environment has been compromised. Based on that information, the client takes evasive actions like surreptitiously erasing all incriminating data from the display and memory, thereby hiding the act of incriminating data manipulation.

Table 3.4: Attestation and Authentication Outcomes on Data Transacted

Results			
Authentication	Attestation	Outcome	Data Transacted
Success	Success	Trusted state	Incriminating
Success	Fail	Non-trusted state	Decoy
Fail	- ^a	System terminate	None

^aSince the system terminates when authentication fails, attestation fails by default and no data is transacted.

3.2.3 Attestation Mechanism and Design

The attestation procedure is inspired from [116]. However, instead of applying the proposed step of verifying individual entries against a known database of trusted and malicious executable fingerprints, the anti-forensics system uses a more stringent method of attestation supported by a single, fingerprint set of trusted executable (the pre-distributed client's trusted state measurement list).

Let L_F be the set of trusted executable codes and their respective hashes pre-distributed to the server, and L_C be the set of codes executed on the client and their respective hashes sent to the server for attestation. The sets are therefore essentially populated by ordered pairs each containing an executable name x and its respective hash value h , given as follows:

$$\begin{aligned}
 L_F &= \{\langle x_{F_1}, h_{F_1} \rangle, \langle x_{F_2}, h_{F_2} \rangle, \dots, \langle x_{F_n}, h_{F_n} \rangle\} \\
 L_C &= \{\langle x_{C_1}, h_{C_1} \rangle, \langle x_{C_2}, h_{C_2} \rangle, \dots, \langle x_{C_n}, h_{C_n} \rangle\}
 \end{aligned} \tag{3.9}$$

Attestation succeeds if and only if the following holds:

$$\forall \langle x, h \rangle [\langle x, h \rangle \in L_C \leftrightarrow \langle x, h \rangle \in L_F] \quad (3.10)$$

Following Equation 3.10, the singular step can be further decomposed into three primitive steps, supported by three sets A , M , and P , as follows:

$$\begin{aligned} \langle x_\alpha, h_\alpha \rangle \in A &\longleftrightarrow [\langle x_\alpha, h_\alpha \rangle \in L_C \wedge \forall \langle x, h \rangle \in L_F [x_\alpha \neq x]] \\ \langle x, h \rangle \in M &\longleftrightarrow [\langle x, h \rangle \notin L_C \wedge \langle x, h \rangle \in L_F] \\ \langle x, h_\beta \rangle \in P &\longleftrightarrow [\langle x, h_\beta \rangle \in L_C \wedge \langle x, h \rangle \in L_F \wedge (h_\beta \neq h)] \end{aligned} \quad (3.11)$$

It can be seen from Equation 3.11 that set A contains entries of codes executed on the client but is not contained in the fingerprint set, set M contains entries present in the fingerprint set but is not present in the client set (implying non-executed code), and set P contains entries where there is a hash mismatch between the client set and fingerprint set (implying tampered executable).

If all of the above-mentioned sets are empty at the end of the verification process, it means that all and only all of the (untampered) executables listed on the fingerprint set L_F are executed by the client. This implies that there has been no tampering of executable codes at the client of which measurement is in L_F , and that no additional executable has been executed (e.g. which may compromise security by monitoring the client) or executables supposed to run but prevented to do so on the client. In other words, the system is supposed to be in exactly the state it was when L_F was generated.

The final step thus succeeds if and only if the sets A , M , and P are empty at the end of verification. Equation 3.11 forms the basis for the attestation algorithm used in the prototype.

3.2.4 Client-Server Transaction / Communications Protocol

A client-server transaction is divided into 2 sections, namely:

- Attestation and authentication
- Data transfer

Table 3.5 lists the primitives that form the communication protocol between the client C and server S .

Table 3.5: Primitives for the Communication Protocol

Component	Description
N_S	Server S generated nonce. This component must be uniquely generated for each transaction.
N_C	Client C generated nonce. This component must be uniquely generated for each transaction.
$Sig_{TPM,SIG,C}$	Digital signature using the client C pre-generated TPM signature key $K_{TPM,SIG,C}$. Signing is natively performed by the TPM.
$Sig_{SIG,S}$	Digital signature using the server S pre-generated signature key $K_{SIG,S}$.
$PCR_{10,C}$	TPM Platform Configuration Register (PCR) 10 in client C . Default PCR where IMA extends all its measurements.

Continued on next page...

Table 3.5 – continued from previous page

Component	Description
$L'_{IMA,C}$	IMA measurement list for client C , showing individual measurements of all codes executed at application space.
$L_{IMA,C}$	The pre-distributed valid, trusted measurement list for client C , showing trusted individual measurements of all executable codes that must be executed. This is stored in the server for use during attestation.
R_S	Attestation result. Takes one of 3 values - <i>SUCCESS</i> , <i>FAIL_{ATTEST}</i> , or <i>FAIL_{AUTH}</i> .
$R_{S,RECV}$	The R_S corresponding to the most recent <i>RECV</i> operation within the session. If the <i>RECV</i> operation has yet to be performed within the session, $R_{S,RECV}$ takes on <i>FAIL_{ATTEST}</i> . A session is defined as the time when client C starts to operate until the time when the client terminates.
COM	Client operation request command. Either <i>SEND</i> or <i>RECV</i> .
$K_{TPM,ENC,C}$	Client C TPM sealing key blob. Used for encryption to the platform configuration of C . Consists of the private and public keys; both keys encrypted to the TPM storage root key at C before being released (implies that key blob is only loadable onto its generating TPM, which is at C). For each TPM sealing cycle, a new key pair is generated.

Continued on next page...

Table 3.5 – continued from previous page

Component	Description
$E_{TPM,ENC,C}$	Data sealed by client C using the public component of $K_{TPM,ENC,C}$. Performed internally at C within the TPM after loading the key blob $K_{TPM,ENC,C}$ into the TPM.
$K_{SYM,C}$	Symmetric key for client C used to encrypt user data. After successful encryption, $K_{SYM,C}$ is encrypted using $K_{TPM,ENC,C}$.
$E_{SYM,C}$	Encryption of client C data with key $K_{SYM,C}$ using a symmetric encryption algorithm (e.g. AES-CBC).
$Data$	Client data, D_I or D_F depending on the attestation outcome as per Table 3.4.

The complete protocol is as follows. The authentication and attestation section is performed first:

1. $S \rightarrow C : N_S$
2. $C \rightarrow S : Sig_{TPM,SIG,C}\{N_S \parallel N_C \parallel PCR_{10,C}\} \parallel N_C \parallel PCR_{10,C} \parallel L'_{IMA,C}$
3. $S \rightarrow C : Sig_{SIG,S}\{N_S \parallel N_C \parallel R_S\} \parallel R_S$

Upon receiving $PCR_{10,C}$ and $L'_{IMA,C}$ in Step 2, server S processes the information (as per the previous subsections) and determines R_S . If $R_S = FAIL_{AUTH}$, both client C and server S terminate. Else, C sends the operation command to S :

4. $C \rightarrow S : Sig_{TPM,SIG,C}\{N_S \parallel COM\} \parallel COM$

Upon receiving COM , the data transfer section begins. If $R_S = SUCCESS$, D_I will be transacted. Else if $R_S = FAIL_{ATTEST}$, D_F will be transacted. Therefore, with $Data$ representing either D_I or D_F , the protocol continues such that if $COM = RECV$ (data read) then

$$5. S \rightarrow C : Sig_{SIG,S}\{N_S \parallel K_{TPM,ENC,C}\} \parallel K_{TPM,ENC,C} \parallel E_{TPM,ENC,C}\{K_{SYM,C}\} \parallel E_{SYM,C}\{Data\}$$

Else, if $COM = SEND$ (data write) then

$$5. C \rightarrow S : Sig_{TPM,SIG,S}\{N_S \parallel K_{TPM,ENC,C}\} \parallel K_{TPM,ENC,C} \parallel E_{TPM,ENC,C}\{K_{SYM,C}\} \parallel E_{SYM,C}\{Data\}$$

When server S receives data after the $SEND$ command, it continues to commit the data to storage (using R_S) to determine the data type). Note that S only stores the $K_{TPM,ENC,C} \parallel E_{TPM,ENC,C}\{K_{SYM,C}\} \parallel E_{SYM,C}\{Data\}$ portion.

For a data write operation, sealing is done between Steps 3 and 4. That is, client C seals its data the moment it receives R_S from server S . Properties of the data sets with reference to the R_S value are discussed in the subsequent subsections.

3.2.5 The Extend Ordered Measurement List Process

The Integrity Measurement Architecture (IMA) measures every executable immediately before they are executed. It appends each measurement result to a list it maintains, and extends them to the PCR it uses via Equation 2.1. The measurement list and PCR values are therefore based on the order in which the executable codes are executed.

Since the client achieves a trusted state if and only if it executes every executable⁴ in the predefined set of trusted executable codes (and nothing more), the IMA measurement list of the client in such state would therefore reflect a permutation of that trusted set as reflected by Equation 3.9.

⁴Implies that the executable codes are not tampered.

As these permutations may not be in constant order across system reboots due to the use of multithreading and multicore processing in modern systems, if left as-is, the end value of the PCR used by IMA in a trusted state will consistently be different, and retrieving sealed data will be very difficult, if not impossible if data is sealed to the PCR used by IMA.

The rationale behind the Extend Ordered Measurement List process is thus to make a copy of the IMA measurement list in a running system, ordered to a consistent, deterministic order, and extend this ordered list into another PCR so that data sealed to this well-known permutation may be used for data sealing and retrieval across reboots. If the system is in a trusted state, the end result will always be constant.

The Extend Ordered Measurement List is executed before data sealing or unsealing by the client, and performs the following using an unused, pre-selected PCR:

1. Reset the pre-selected PCR to its initial state.
2. Read the IMA measurement list into memory.
3. Sort the measurement list in order based on hash values.
4. Extend the sorted list entries one by one using the equations in Equation 2.1 into the pre-selected PCR.

When the process is complete, data manipulation would use the newly extended PCR for sealing and unsealing when in a trusted state. This ensures that data sealed at one trusted state instance can always be unsealed in another trusted state instance.

It is very important to note that the Extend Ordered Measurement List process must ensure the following:

1. Protection of the PCR in use. This means that no other entity must be allowed to extend to the PCR used by this process. If an attempt to extend

the PCR is detected, measures must be taken mitigate the attempt (e.g. invalidate the PCR, trigger a kernel panic).

2. The module that performs the process must be system-implemented and not application-implemented. Therefore, the module should best be implemented as a system module, usable by other entities but cannot be controlled or tampered with.
3. The module starts during system startup. The startup script that loads the module must be at the same level in the chain of trust as that of the kernel image, to prevent bypass of the module.

The first requirement is important since if any entity could extend to the PCR used by the process, it could mean that the same entity could reconstruct a fake ordered measurement list and extend it, thus leading to falsification of attestation results to the client.

The second and third requirements, when coupled with PCR protection, would prevent the module from being tampered with or bypassed, as an attacker would need to modify the startup script which will then be reflected in the chain of trust.

3.2.6 System Data Sets

In an ideal scenario, both the incriminating D_I and decoy D_F are differentiated by the PCRs used during sealing. Since sealed data can only be unsealed when the selected PCRs match, manipulating the PCR selection during sealing therefore allows D_F to be made a close decoy of D_I , while protecting D_I from being forcefully unsealed in a non-trusted environment.

Table 3.6 summarizes the PCR sets used in sealing D_I and D_F , and the states which they can be unsealed under such ideal scenario.

However, such arrangement may result in an undesirable scenario where information in D_I gets sealed as D_F , and subsequently accessible through D_F . Such violation could be illustrated as follows.

Table 3.6: Ideal Properties of D_I and D_F

Data Set	Sealing PCR Set ^a	States Allowed for Manipulation
D_I	CRTM, BIOS, Boot Loader, OS Kernel, EOML (for IMA), OOB-D	Trusted state
D_F	None	Trusted and non-trusted states

^aCRTM = Core Root of Trust Measurement, EOML = Extend Ordered Measurement List, OOB-D = Out-of-Band Device

Let T and T' be the trusted and non-trusted states respectively, $R(D)$ be the data read transaction and $W(D)$ be the data write transaction. A typical data editing scenario would therefore be $R(D)$ followed by $W(D)$. To violate access control and obtain D_I , the following steps can be done:

1. $R(D_I) \in T$
2. $Execute(\mu) \xrightarrow{causes} (T \rightarrow T')$
3. $W(D_I) \in T'$
4. $R(D_I) \in T'$

It is not hard to see how law enforcement agents could manipulate such situation to obtain D_I , as all that is needed is to raid and seize control of the client machine immediately after detecting an $R(D_I)$ operation⁵, tamper with the machine's integrity, and perform a data write, resulting in D_I being saved as decoy D_F without any PCR protection and therefore accessible under all tamper scenarios.

⁵This could possibly be done through legal network monitoring.

To address this, there is a need to disassociate D_I and D_F from their ideal sealing PCR selection. Instead, data sealing should take into account the immediate previous $R(D)$ state (denoted $R_{S,RECV}$) within the session, and if an attempted violation is detected, seals the data with a random PCR set which is first extended with random values. Table 3.7 summarizes the way data is sealed with due consideration given to the immediate previous read state, $R_{S,RECV}$.

Table 3.7: Data Sealing with Previous State Consideration

$R_{S,RECV} :=$ $[R(D) \in$ $\{T, T'\}]$		$W(D)$				
State ^a	Data	State	PCR Set		Data	Subsequent Unseal State
T	D_I	T	CRTM, BIOS, Boot Loader, OS Kernel, EOML (for IMA), OOB-D		D_I	Trusted state
T'	D_F	T'	None		D_F	Trusted and non-trusted states
T	D_I	T'	Random		D_F	None ^b

^aNote that it is not possible for $R(D)$ to be in T' to be followed by $W(D)$ to be in T due to the non-reversibility of the non-trusted to trusted state.

^bGiven a random PCR set with random values, it is computationally infeasible to find the same matching set with the exact same random values required to unseal the data.

In addition to the normal operating scenario where $W(D)$ is preceded by $R(D)$,

there exists a situation during the system first use which $W(D)$ will not be preceded by $R(D)$ due to the non-existence of D_I or D_F within the system. In such cases where $R_{S,RECV}$ is undetermined, $W(D)$ will seal the data based on the current system state it is in (whether T or T') to create either D_I or D_F . In order to maintain the ability to be in a trusted state, such actions would need to be performed by the client executable and not by a separate executable.

3.2.7 Data Encryption

Data encryption is done on client C via the following steps (refer to Table 3.5 for description of primitives):

1. Generate a random symmetric key $K_{SYM,C}$.
2. Generate TPM sealing key blob $K_{TPM,ENC,C}$.
3. $E_{SYM,C}\{Data\}$.
4. $E_{TPM,ENC,C}\{K_{SYM,C}\}$ using the correct PCR set.
5. Construct $K_{TPM,ENC,C} \parallel E_{TPM,ENC,C}\{K_{SYM,C}\} \parallel E_{SYM,C}\{Data\}$.
6. Append $Sig_{TPM,C}\{N_S \parallel K_{TPM,ENC,C}\}$.

$$Sig_{TPM,C}\{N_S \parallel K_{TPM,ENC,C}\} \parallel K_{TPM,ENC,C} \parallel E_{TPM,ENC,C}\{K_{SYM,C}\} \parallel E_{SYM,C}\{Data\}$$

would then be sent to the server for storage. In Step 4, the client C would automatically choose the PCR set used to seal $K_{SYM,C}$ based on the value of R_S and $R_{S,RECV}$ (refer to Table 3.7).

Note that the $K_{TPM,ENC,C}$ blob can be published freely because it can only be loaded onto its generating TPM. Intercepted encrypted data blobs cannot be used on any arbitrary TPM-enabled machines by virtue of this.

New $K_{SYM,C}$ and $K_{TPM,ENC,C}$ are generated for every encryption-decryption cycle. This is necessary to ensure security, though at the expense of key generation times. Moreover, $K_{SYM,C}$ is wiped from memory right after it is sealed.

3.2.8 Data Decryption

When server S is requested to send $K_{TPM,ENC,C} \parallel E_{TPM,ENC,C}\{K_{SYM,C}\} \parallel E_{SYM,C}\{Data\}$, it first generates its signature $Sig_{SIG,S}\{N_S \parallel K_{TPM,ENC,C}\}$ and appends it to the message. Using the signature, the client could verify that the message is not a replay, and continue to perform the decryption proper.

Data decryption involves running the following steps:

1. Parse $K_{TPM,ENC,C} \parallel E_{TPM,ENC,C}\{K_{SYM,C}\} \parallel E_{SYM,C}\{Data\}$.
2. Unseal $E_{TPM,ENC,C}\{K_{SYM,C}\}$ to get $K_{SYM,C}$.
3. Decrypt $E_{SYM,C}\{Data\}$ to get $Data$.

The structure $K_{TPM,ENC,C} \parallel E_{TPM,ENC,C}\{K_{SYM,C}\} \parallel E_{SYM,C}\{Data\}$ (and its corresponding signature) is the same regardless of whether the data is incriminating or fake. Therefore, during traffic analysis, both data will look similar and there is no way to tell from the structure whether the data that will be decrypted contains incriminating information or is a decoy.

$K_{SYM,C}$ and $K_{TPM,ENC,C}$ are wiped from memory immediately after their uses are no longer required. This one time property eliminates key storage on the client and prevents key retrieval by memory dumping.

Since $K_{SYM,C}$ is only present on the client momentarily, the possibility of them being intercepted are almost negligible, as the intercepting entity would have to be executed on the client machine between right after attestation and right before $E_{TPM,ENC,C}\{K_{SYM,C}\}$ is complete (for the encryption process), or between right after unsealing $E_{TPM,ENC,C}\{K_{SYM,C}\}$ and right before data decryption is complete (for the decryption process).

3.3 Security Analysis of Design

Before analyzing the design proper, it is imperative to note the following:

1. Sealed data can only be unsealed by the TPM that generated it, and only if the selected platform configuration matches that during sealing.
2. TPM keys from one TPM cannot be arbitrarily loaded onto any other TPM unless specifically authorized. Even then, not all key types can be authorized for loading into other TPMs.
3. A system can be in a trusted state if and only if it adheres to Equation 3.11.
4. Any additional executable codes executed on a system while in a trusted state will convert the system into a non-trusted state.

The entire communication protocol for the system is depicted in Figure 3.4.

3.3.1 Authentication and Attestation

Authentication and attestation involve the first three steps within the protocol. In GNY logic, they can be written as follows:

1. $C \triangleleft *N_S$
2. $S \triangleleft * \{ * \{ H(N_S, N_C, PCR_{10,C}) \}_{-K_{TPM,SIG,C}}, *N_C, *PCR_{10,C}, *L'_{IMA,C} \}$
3. $C \triangleleft * \{ * \{ H(N_S, N_C, R_S) \}_{-K_{SIG,S}}, *R_S \}$

Before proceeding, a few assumptions are required. They are as follows:

$$\begin{aligned}
S \ni N_S & \quad C \ni N_C & C \models \xrightarrow{+K_{SIG,S}} S \\
S \models \#(N_S) & \quad C \models \#(N_C) & C \models S \ni -K_{SIG,S} \\
S \models \phi(N_S) & \quad C \models \phi(N_C) & C \models S \Rightarrow S \models * \\
S \models \xrightarrow{+K_{TPM,SIG,C}} C & &
\end{aligned} \tag{3.12}$$

To authenticate client C , server S verifies $\{ H(N_S, PCR_{10,C}) \}_{-K_{TPM,SIG,C}}$ using the key $+K_{TPM,SIG,C}$ transferred beforehand. Authentication is possible because of two factors:

Figure 3.4: Communication protocol for anti-forensics system.

- Only client C can generate $\{H(N_S, PCR_{10,C})\}_{-K_{TPM,SIG,C}}$ since $-K_{TPM,SIG,C}$ cannot be forged or loaded elsewhere.
- N_S is a random nonce freshly generated for every transaction; therefore replay attacks are not possible.

At the end of a successful authentication, the following holds:

$$\begin{aligned}
S &| \equiv C \mid \sim (\{H(N_S, N_C, PCR_{10,C})\}_{-K_{TPM,SIG,C}}, N_C, PCR_{10,C}, L'_{IMA,C}) \\
S &| \equiv C \ni -K_{TPM,SIG,C} \\
S &| \equiv C \ni PCR_{10,C}
\end{aligned} \tag{3.13}$$

The attestation procedure then starts and involve the following two phases. It is assumed that S has a copy of a valid, trusted measurement list $L_{IMA,C}$ to compare with the received $L'_{IMA,C}$. The ordering of entries in $L_{IMA,C}$ is not important for attestation.

1. Verifying the aggregate of $L'_{IMA,C}$ against $PCR_{10,C}$.
2. Based on the stored copy of the trusted measurement list $L_{IMA,C}$, verify that $L'_{IMA,C}$ is trusted by using the method proposed in Section 3.2.3.

When the first phase of attestation completes successfully, S can extend its belief domain to:

$$S \mid \equiv C \ni (PCR_{10,C}, L'_{IMA,C}) \tag{3.14}$$

Server S can then proceed with the second phase believing that the list it is verifying indeed originated from client C .

Based on the authentication and attestation results, S then generates R_S , signs it together with N_S and N_C , and sends the entire message to client C (Step 3 of the protocol). When C receives the message, it could deduce the following:

$$\begin{aligned}
C &| \equiv S \mid \sim (\{H(N_S, N_C, R_S)\}_{-K_{SIG,S}}, R_S) \\
C &| \equiv S \mid \equiv \#(N_S) \\
C &| \equiv S \ni R_S
\end{aligned} \tag{3.15}$$

This allows C to believe that the R_S value it uses as basis for further action is the one that S possesses. Since C also believes that S is honest and competent, it could also deduce the following:

$$C \mid \equiv S \mid \equiv R_S \tag{3.16}$$

From Equation 3.15, C can proceed to believe that:

$$C \mid \equiv \#(N_S) \tag{3.17}$$

With such beliefs, the system could then proceed to transact data.

3.3.2 Data Read Transaction

When client C is performing a data read, it sends $COM = RECV$ to server S , in which S then replies with the encrypted data as per Step 5 (of the communication protocol presented in Section 3.2.4). They can be represented as follows:

4. $S \triangleleft * \{ * \{ H(N_S, RECV) \}_{-K_{TPM,SIG,C}}, * RECV \}$
5. $C \triangleleft * \{ * \{ H(N_S, K_{TPM,ENC,C}) \}_{-K_{SIG,S}}, * K_{TPM,ENC,C}, * \{ K_{SYM,C} \}_{+K_{TPM,ENC,C}}, * \{ Data \}_{K_{SYM,C}} \}$

Upon receiving the client request and once the signature is verified, S is entitled to believe that the request originated from client C , and that it is not a replay. That is:

$$\begin{aligned}
S &| \equiv C \mid \sim (\{H(N_S, RECV)\}_{-K_{TPM,SIG,C}}, RECV) \\
S &| \equiv C \ni RECV
\end{aligned} \tag{3.18}$$

Server S then sends the encrypted data⁶ to C . When C verifies the signature, it could proceed to deduce the following:

$$\begin{aligned}
C &| \equiv S \mid \sim (\{H(N_S, K_{TPM,ENC,C})\}_{-K_{SIG,S}}, K_{TPM,ENC,C}) \\
C &| \equiv S \equiv \#(\{H(N_S, K_{TPM,ENC,C})\}_{-K_{SIG,S}})
\end{aligned} \tag{3.19}$$

By virtue of being able to load the TPM key, C can further deduce that the following occurred in a previous transaction:

$$\begin{aligned}
C &\mid \sim K_{TPM,ENC,C} \\
C &\ni K_{TPM,ENC,C}
\end{aligned} \tag{3.20}$$

Client C then proceeds to use the private key portion from $K_{TPM,ENC,C}$ (referred to as $-K_{TPM,ENC,C}$) to decrypt $\{K_{SYM,C}\}_{+K_{TPM,ENC,C}}$. $K_{TPM,ENC,C}$ is then unloaded and wiped after decryption is performed. Successful decryption yields:

$$C \ni K_{SYM,C} \tag{3.21}$$

Using $K_{SYM,C}$, C can then decrypt and obtain the data from $\{Data\}_{K_{SYM,C}}$. Upon decryption, $K_{SYM,C}$ is wiped.

3.3.3 Data Write Transaction

From the communication protocol perspective (refer to Section 3.2.4), a write transaction is similar to a read transaction, except with $COM = SEND$ and the server receiving encrypted data as per Step 5 of the protocol. They can be represented as follows:

⁶Whether the encrypted D_I or D_F is sent depends on R_S prior.

4. $S \triangleleft * \{ * \{ H(N_S, SEND) \}_{-K_{TPM, SIG, C}}, * SEND \}$
5. $S \triangleleft * \{ * \{ H(N_S, K_{TPM, ENC, C}) \}_{-K_{TPM, SIG, C}}, * K_{TPM, ENC, C},$
 $* \{ K_{SYM, C} \}_{+K_{TPM, ENC, C}}, * \{ Data \}_{K_{SYM, C}} \}$

As with the read transaction, once server S verifies the signature for the request, it is entitled to believe the following:

$$\begin{aligned} S &| \equiv C | \sim (\{ H(N_S, SEND) \}_{-K_{TPM, SIG, C}}, SEND) \\ S &| \equiv C \ni SEND \end{aligned} \tag{3.22}$$

Server S then receives the encrypted data and appended signature. Verifying the signature, S adds to its belief that:

$$\begin{aligned} S &| \equiv C | \sim (\{ H(N_S, K_{TPM, ENC, C}) \}_{-K_{TPM, SIG, C}}, K_{TPM, ENC, C}) \\ S &| \equiv C \ni K_{TPM, ENC, C} \end{aligned} \tag{3.23}$$

On successful verification, server S then proceeds to store $\{ K_{TPM, ENC, C}, \{ K_{SYM, C} \}_{+K_{TPM, ENC, C}}, \{ Data \}_{K_{SYM, C}} \}$, using R_S to determine whether to store as D_I or D_F . Note that S does not need to verify the validity of the components within $\{ K_{TPM, ENC, C}, \{ K_{SYM, C} \}_{+K_{TPM, ENC, C}}, \{ Data \}_{K_{SYM, C}} \}$, as any tampering done to it or any of the components will lead to a situation where the data could not be decrypted by client C later on.

3.3.4 Information and Action Hiding

3.3.4.1 Information Hiding

In order to retrieve D_I from the system, the criminal would need to acquire all of the following:

- the client machine C
- the out-of-band device E , activated properly

- the trusted state T on the client C
- the sealed data containing incriminating information X
- the client TPM sealing key K for sealed data X

If all the above prerequisites are represented as a set V_{D_I} , such a set can be defined as:

$$V_{D_I} = \{C, E, T, X, K\} \quad (3.24)$$

It can be deduced from the system design and Equation 3.24 that the prerequisites are transitive. For example, in order to achieve a trusted state T , possession of the client C and proper activation of the out-of-band device E are required (though not sufficient). Such transitivity involved can be written as follows:

$$\begin{aligned} \{X, K\} &\longrightarrow T \\ T &\longrightarrow \{C, E\} \end{aligned} \quad (3.25)$$

To achieve T , the criminal must achieve all of the following (refer to Equation 3.11 for the definition of the sets):

- be using the designated client C
- possess and properly activate the out-of-band device E
- execute all executable codes in the trusted measurement list so that $M = \{\}$
- not execute any executable not in the trusted measurement list so that $A = \{\}$
- not tamper with any executable listed in the trusted measurement list so that $P = \{\}$

Let $M = \{\}$ be written as M_0 , $A = \{\}$ be written as A_0 , and $P = \{\}$ be written as P_0 , the trusted state T can be written as a set as follows:

$$T \longleftrightarrow \{C, E, \{M_0, A_0, P_0\}\} \quad (3.26)$$

Replacing T in Equation 3.24 with Equation 3.26 and removing redundancies:

$$V_{D_I} = \{\{C, E, \{M_0, A_0, P_0\}\}, X, K\} \quad (3.27)$$

For any entity W to obtain D_I , it must first comply with the following:

$$W \models \phi(V_{D_I}) \wedge W \ni V_{D_I} \quad (3.28)$$

Therefore, using Equation 3.28:

$$W \ni D_I \longleftrightarrow W \models \phi(V_{D_I}) \wedge W \ni V_{D_I} \quad (3.29)$$

The criminal would easily be able to recognize and obtain V_{D_I} , since he or she possesses C and E , and knows how to achieve M_0 , A_0 , and P_0 in order to achieve T . When T is achieved, X and K can be retrieved from the server, therefore completing the V_{D_I} set. With V_{D_I} , the criminal can proceed to extract D_I from X .

On the other hand, the ability for law enforcement agents to obtain D_I would depend on their knowledge about the V_{D_I} set, and their ability to acquire the elements of the set.

Since it is defined that law enforcement agents only have guaranteed access to the client C , it can be assumed that C is easily obtained. However, with a sufficiently discreet E^7 and $\{M_0, A_0, P_0\}$ set, the task of obtaining the complete

⁷e.g. using infrared beams, RSSI values, etc.

V_{D_I} set may be rendered either 1) impossible, or 2) temporally long enough such that it becomes possible for the remote server to be shut down, thus disabling the system.

In the first scenario where law enforcement agents could not obtain E or $\{M_0, A_0, P_0\}$, it is clear from Equation 3.28 that they would not be able to obtain D_I from the system.

In the second scenario, assume that law enforcement agents obtain E and $\{M_0, A_0, P_0\}$ at time t . Information hiding is preserved if between t_0 (where law enforcement agents first obtain C) and t , the server is taken offline. By doing so, law enforcement agents would only be able to obtain the set $\{C, E, \{M_0, A_0, P_0\}\}$, as X and K would not be available for transaction.

Such measure is practically sound considering it could be trivially implemented (e.g. by having an accomplice at the server's location, or using a remote shutdown script).

Note that both scenarios previously described do not take into account lawful network interception. Nevertheless, it can be seen that if X and K (the only two components in V_{D_I} transferred over the network) are intercepted, they would be rendered useless as any attempts to forcibly utilize X and K at the client would require additional executable codes, which when executed will void A_0 (due to IMA and EOML restrictions).

Therefore, by careful manipulation of the system to prevent law enforcement agents from procuring V_{D_I} , the following holds as a result:

$$\neg(L \ni V_{D_I}) \longrightarrow \neg(L \ni D_I) \quad (3.30)$$

From Equation 3.30, it can be seen that right hand side of the statement corresponds to the objective of information hiding as defined in Equation 3.3.

3.3.4.2 Action Hiding

Action hiding pertains to the ability for the system (specifically the client) to project absence of any acts of incriminating data manipulation. Naturally, such projection only occurs when the system is not in use.

The system by virtue of its architecture allows for action hiding when the system is not in use. The client only stores the client-side application, the TPM signing key, and the server signature public key. The two keys stored at the client by themselves do not reflect their use as part of the system.

At the same time, the client-side application could be named and located such that it is not obvious of its true purpose (unless subjected to low-level binary analysis⁸). Since the application is the only component within the client that directly interacts with the data, hiding it therefore hides the act of data manipulation (the second statement in Equation 3.3).

In addition, practical methods like disabling swap partitions and performing manipulation only in memory (and wiping the data from memory once manipulation is complete) further strengthen the action hiding capabilities of the system by increasing the effort required to determine the function of the application.

3.3.5 Plausible Deniability

Fundamentally, there are two methods to deny the existence of an information set. One, an entity could project the non-existence of such information set by providing supporting reasoning and information to project such non-existence. Alternatively, an entity could instead project the existence of such information set, but provide a decoy information set and reasoning that the decoy set is in fact the information set of interest.

Information hiding provides the first method of denial. However, it is possible that such denial may not be accepted, especially if law enforcement agents have strong reason to believe the existence of D_I . Therefore, the system employs the

⁸Such low-level analyses may not be conducted due to practicality, e.g. in a system with many seemingly innocuous binary files.

decoy information set D_F to allow for the criminal to plausibly deny the existence of D_I by reasoning that D_F is D_I .

In order to obtain D_F from the system, the following are required:

- the client machine C
- the sealed data containing decoy information X_F
- the client TPM sealing key K_F for X_F

If written as a set V_{D_F} as per Equation 3.27, such a set can be written as:

$$V_{D_F} = \{C, X_F, K_F\} \quad (3.31)$$

Again from the system design, a user could obtain X_F and K_F upon possession of C . X_F is decrypted using K_F to obtain D_F .

D_F is designed such that it mimics D_I in every single way except 1) the exclusion of incriminating information, and 2) it is accessible when an entity obtains V_{D_F} (instead of V_{D_I} for D_I). Subsequently, given D_F , an entity should be able to read, write, and edit it the same way it could edit D_I (on the same client machine C).

It is important to note that since the law enforcement agents have yet to see the contents of D_I (or D_F) at the point of forensic examination, they cannot prove that:

1. D_I contains incriminating information.
2. D_F exists.
3. D_F is a decoy.

Nevertheless, although the law enforcement agents cannot prove the existence of D_F and its function as a decoy, it is assumed that they can suspect that D_F exists for that purpose as per Equation 3.5. In addition, the law enforcement

agents could also only suspect that D_I contains information of interest i (i.e. incriminating information) before any forensic examination is conducted. This basis for pursuing D_I could therefore be written as (where γ denotes the assumed probability that D_I contains i):

$$L \models [\Pr(i \in D_I) = \gamma] \quad (3.32)$$

If presented with a data set D_I that contains no i , the law enforcement agents would have no means of proving otherwise (in the context of the system), since to prove otherwise would require proof that:

1. The D_I served is a decoy.
2. That there exists a true D_I .
3. That the purported true D_I contains i .

Therefore, when presented with D_F , the law enforcement agents should lessen their belief that D_F is not D_I compared to before D_F is disclosed. This can be written as (where β_0 is the belief probability that D_F is not D_I prior to the disclosure of D_F):

$$L \triangleleft D_F \longrightarrow L \models [\Pr(D_F \neq D_I \mid L \triangleleft D_F) = \beta_0 - \delta_\beta] \quad (3.33)$$

Note that Equation 3.33 does not correspond to any GNY postulates, and is in fact a consequence of the inability of the law enforcement agents to prove that D_F is not D_I .

It can be seen that the right hand side of Equation 3.33 is an alternate representation of Equation 3.6, and thus the aim of plausible deniability.

Further leveraging on the safeguards in place to prevent disclosure of E and $\{M_0, A_0, P_0\}$, and therefore D_I , the law enforcement agents should consistently

obtain V_{D_F} instead, of which D_F will be served. Therefore, $\Pr(D_F \neq D_I \mid L \triangleleft D_F)$ should continue to decrease with each D_F revelation instance, which ultimately leads to:

$$L \triangleleft D_F \longrightarrow L \models \neg(D_F \neq D_I) \quad (3.34)$$

The right hand side of Equation 3.34 corresponds to Equation 3.7, which is the stronger aim of plausible deniability.

The subsequent statement in Equation 3.8 can be ascertained based on subsequent actions by the law enforcement agents upon reaching Equation 3.33 (or Equation 3.34). For example, such ascertainment could come when the case is dropped against the criminal due to lack of evidence (assuming that the system is the only source of evidence). In such situations, although the law enforcement agents may still suspect that D_F is not D_I , they are compelled to believe otherwise (e.g. inability to prove that D_F is a decoy).

3.3.6 Role of the Trusted Platform Module

While it may not seem obvious the use and need for the TPM within the system, it is imperative to note that the system would not function as intended should the TPM-based functions be replaced with similar functions not using a TPM.

Specifically, there are three domains that the role of the TPM can be noted:

Data encryption The system utilizes data sealing which allows decryption only on the encrypting machine, with matching platform configuration. Without TPM-based encryption, data can be decrypted anywhere as long as the provided keys / credentials match.

Attestation Attestation relies on the platform configuration which is guarded by the TPM. Without TPM-based guarding, forgery of the platform configuration is possible (i.e. tampering of the measurement list).

Authentication The TPM provides an identity through the use of a TPM-generated signature key by the authenticating entity. This TPM-based key cannot be forged, and cannot be loaded into another TPM unless explicitly allowed. Without such TPM-based key protection, the identity proving technique is either open to forgery, or requires significant addition of anti-forgery measures.

3.4 Prototype Implementation and Anti-Forensics Capabilities Test

A basic working prototype of the system with the OOB software simulated has been implemented using the following components:

Server Dell Optiplex GX620 running Fedora 8.

Client Dell Optiplex 745 running Fedora 8, configured with the TrouSerS TSS Stack, Grub-IMA boot loader [117], and a kernel with IMA patch.

The execution flow for the client prototype is presented in Figure 3.5. The Dell Optiplex 745 has an embedded Atmel TPM version 1.2.

It is important to note that no swap partitions have been allocated to prevent swap partition analysis. Without a swap partition, keys and incriminating data will always be manipulated in memory, and they will be wiped from memory when no longer required, thus significantly increasing the difficulty of retrieving these data (e.g. by using techniques described in [18]).

Evaluation is performed by putting the system through some common forensic techniques. They are listed with the subsequent outcomes in Table 3.8. The evaluation goes to show that the system hinders forensics examination sufficiently, with the only way of obtaining D_I is to run the system in a trusted state.

Figure 3.5: Execution flow for the anti-forensics client prototype.

Table 3.8: Forensic Techniques and Anti-Forensic Outcome

Forensic Technique	Outcome	Analysis
Duplicate client hard disk, run on another computer without TPM. ⁹	Client fails to start due to missing TPM.	The client has been designed not to start if it fails to detect a TPM.
Duplicate client hard disk, run on another computer with TPM. ⁹	Client fails to start due to inability to load client signature key.	The client signature key has been created to be non-migratable, and it demonstrates adherence to the TCG specification of not being loadable on a system with a different TPM Storage Root Key.
Duplicate client hard disk, run on another computer with TPM. TPM signature key created to be used as client signature key. ⁹	Client starts, but transaction results in <i>FAIL_{AUTH}</i> . Client then terminates.	Adherence to rules for non-possession of $K_{TPM,SIG,C}$, as the server detects an invalid / fraudulent $K_{TPM,SIG,X}$; the first two beliefs in Equation 3.13 do not hold.

Continued on next page...

⁹The scenarios were first run without the OOB device and again with the OOB device.

Table 3.8 – continued from previous page

Forensic Technique	Outcome	Analysis
Client machine run as-is, OOB component simulator not executed.	Client starts, transaction results in <i>FAIL_{ATTEST}</i> due to missing OOB component simulator.	The non-execution of the OOB component simulator serves both to simulate the non-presence of an OOB device <i>E</i> in Equation 3.26 and doubles as failure trigger for non-empty <i>M</i> in Equation 3.11. Attestation failure shows adherence to both.
Client machine run as-is, with OOB component simulator.	Client starts, transaction results in <i>FAIL_{ATTEST}</i> due to non-execution of ‘required’ application ¹⁰ .	Attestation failure due to non-empty <i>M</i> in Equation 3.11.
Client machine run as-is, with OOB component simulator and additional applications.	Client starts, transaction results in <i>FAIL_{ATTEST}</i> due to presence of additional applications in measurement list.	Attestation failure due to non-empty <i>A</i> in Equation 3.11.

Continued on next page...

¹⁰The ‘required’ application was set to be ‘gedit’.

Table 3.8 – continued from previous page

Forensic Technique	Outcome	Analysis
Client machine running in trusted state, additional application subsequently executed.	Client reverts from trusted to non-trusted state. Attempts to save D_I from trusted state to D_F fails.	State reversion due to failure in possessing A_0 in Equation 3.26. Failure to save D_I to D_F shows adherence to policy listed in Table 3.7.
Client machine running in non-trusted state, D_F replaced with an earlier intercepted D_I over the network during read.	Client unable to decrypt D_I .	Failure to decrypt D_I shows adherence to Equation 3.27, where the equation does not hold due to non-possession of T .
Client machine running in trusted state, machine left untampered.	D_I is obtainable.	This is the expected outcome when Equation 3.27 holds.

3.5 Positive Uses

If looked at from a different perspective, such systems, although criminal-centric in design, may be useful in the areas where there is a need to hinder forensic-like extraction of information.

One such area would be that of tactical military communication, where it is can be used to prevent access to friendly tactical information on data equipment captured by adversaries. As tactical information would no longer be stored within the equipment but at the command center, an adversary would first need to know how to run the data equipment in a trusted state before being able to obtain the actual tactical information (this assuming that the command center has yet to block access to information that could be unsealed by the captured device).

Another area which such systems could be used is in the intelligence sector, where headquarters could employ pre-sealed decoy information (i.e. false orders that are benign) to allow operatives to claim plausible deniability in the field.

A third area of use would be that of creating designated secure document access areas, where only within such areas that sensitive and classified documents could be accessed. In such areas, the area itself contains (or is) the OOB device, and any attempts to access information outside the area will yield negative results.

3.6 Conclusion

This chapter describes a mechanism that lays the foundation for establishing a secure, anti-forensics system. The system design considers various vectors in which forensic examination can be conducted. Security analysis of the system shows how each component contributes to the overall objective of the system being anti-forensics capable. An important note is that the system is designed to hinder forensics, not prevent it. Therefore, as a hinderer, it works as it should. However, as shown, the system does not completely prevent forensics since human factor comes into play.

Nonetheless, if such systems were to be used in a real world scenario, classical

forensic approaches will no longer work effectively. There is also no single approach to extracting evidence from such systems, as criminals who use such systems could tailor them to their own, customized protocols (albeit based on the specifications in this paper). Forensic extraction of evidence in such systems in real world scenarios will therefore require complete understanding of the targeted system before an approach is made to circumvent the active anti-forensic measures in place.

This page is intentionally left blank.

Chapter 4

Hindering Forensics in Mobile Systems

At times, one may be required to use binaries that, if discovered, could lead to undesired consequences. For example, a clandestine operative found in possession of binaries used for deciphering information received from the base of operation may find it difficult to plausibly deny that the use of such binaries is not for espionage, even when the information transacted does not appear to be of such nature. In addition, the base of operation and employer of the agent may not want to disclose its role in the entire scheme in order not to trigger greater repercussions.

The issue of discovered binaries has well been an issue with the increasing ubiquity of digital devices, particularly smartphones and tablets. Devices can be lost, stolen, seized, or even accidentally handed over to a hostile entity, who may attempt to discover the contents and binaries present in the device. Even in cases of theft where devices may be stolen for the hardware, there is no surety that they do not happen because of the data (i.e. contents and binaries). What this implies is that devices are, at times, stolen for the what they contain.

For devices to be stolen for their contents, it can be assumed that such acts would require some level of sophistication compared to a typical thief (i.e. the perpetrators would somehow know or suspect that the device contains information

of interest). From that, it could also be assumed that the perpetrators may be capable of performing forensics analysis on the obtained device.

The term anti-forensics often comes with the notion of attempts to resist forensics examination against law enforcement agents. While that may be true, it is also possible that anti-forensic techniques be used in the protection of contents, specifically contents like binaries that may require handling only by certain, trusted users (and perhaps at certain designated areas). Practical systems of such nature are hard to come by, and in most cases of anti-forensics, it is passive (e.g. manually encrypting data and wiping them after manipulation).

4.1 Challenges for Enabling Anti-Forensics in Mobile Devices

Mobile digital devices are increasingly powered by what is known as a mobile operating system. These can be closed source systems (e.g. Apple iOS), or open sourced ones like Google Android. They are tailored specifically for the hardware in mobile devices (e.g. touchscreen, cellular radio, etc.), and usability in a mobile environment.

The main challenge of anti-forensics within such a domain would be to prevent unauthorized disclosure of contents stored within the device under scenarios of device exfiltration. Such scenarios may include the exfiltration of device while powered on and in use, while powered on but not in use, and while powered off.

Another challenge would be to prevent not only the disclosure of the contents, but also the ability to project non-existence of such contents to an adversary. This may include measures such as information and action hiding, as well as plausible deniability.

Techniques such as steganography and data hiding have been used to intentionally deceive investigators into believing that the data is not present. While useful in hiding data, they require overhead in the form of a carrier or container, which

in turn may be discovered if stored locally. Moreover, the amount of overhead needed would depend on the specific technique used and may not be practical to be used in hiding binaries in a mobile operating system.

In [118], Dunn et. al. propose a method in which malwares can be cloaked using a Trusted Platform Module (TPM). Though demonstrating yet another misuse of a TPM, it could potentially serve as a base for enabling anti-forensics of binaries.

One fundamental issue with [118] is that for the scheme to work, the victim's system has to have a local malware that is capable of kernel-level operations. Disregarding the capability of the local malware, the very presence of such an entity could be discovered during forensics of the device.

Nevertheless, [118] outlines a fundamentally practical method to hide sensitive binaries (or their portions) - by obtaining them only when they are to be used.

4.2 Considerations for Anti-Forensics in A Mobile Operating System

Before proceeding to design an anti-forensics technique for the Android operating system, it is beneficial to first identify some specific characteristics of a mobile device, and how they can be used to the advantage of an entity seeking anti-forensics.

Battery Powered The majority of mobile devices run on batteries, with occasional charging from a wall socket. This means that barring being charged, a device can be powered off by simply removing its battery.

Limited Interface The user interface for mobile devices is usually minimalistic, which favours simple operations as opposed to a full fledged computer system. There are usually no ready applications that allow access to the layers of the operating system below the application layer.

Flash-Based Storage Mobile devices usually use flash-based storage, and the

operating system is usually stored on a flash memory chip soldered to the system board of the device. This makes removing the chip harder compared to a computer, as one may have to desolder the chip from its board, which usually requires the powering down of the device.

Connectivity-Rich Mobile devices usually have multiple methods of communication, e.g. WiFi, Bluetooth, cellular, etc. some of which are more likely to be on than off (e.g. the cellular radio in a mobile phone).

The characteristics listed would have implications on how forensics is conducted on a mobile device, and correspondingly the anti-forensic techniques.

Battery Powered During forensics examination, a forensics examiner may have to decide whether to power the device down. This may be risky, as there is no guarantee that there are no shutdown triggers (for both proper and abrupt shutdown) present in the system to wipe sensitive information. Moreover, as devices usually enter a standby mode after being idle for some time, there is also no guarantee that there is no trigger present during the mode-switch or unlocking procedure (e.g. a trigger to wipe sensitive data if the correct unlock pattern is not presented).

Limited Interface Without the ability to access lower layers, first responders may not know what lies beneath the application layer (e.g. whether there are any hidden triggers). Moreover, there would invariably be a time delay between the device being acquired by first responders to the time it is subjected to forensics examination, which may allow a time-based trigger to be executed.

Flash-Based Storage The need to desolder the flash chip may potentially damage the chip if not done carefully. Moreover, as access to the system board is required, it could mean having to power down the device, which may execute hidden triggers.

Connectivity-Rich A live connection can be used to send a kill-code to wipe sensitive information in the device or disable the device outright. Even if first responders decide to utilize signal jammers to prevent kill-codes from being sent, there is still a possibility that a trigger could be programmed to execute when signal state switches from connected to no connectivity.

4.2.1 Anti-Forensic Measures

Based on the characteristics listed, a recurring possibility for anti-forensics in a mobile device would be to implement hidden triggers, given the many possible locations such triggers could be implemented. The use of triggers however, has to take into consideration the nature of which anti-forensics is used for. For example, if an operative were to use a device which literally self-destructs the device once the device is acquired by an adversary, it would be problematic as the operative would not be able to explain plausibly why such destruction occurred without giving himself or herself away.

To safeguard binaries from being discovered, destruction of a device would be overkill, as it only requires secure deletion of the following

- The binary or binaries of interest.
- Any data generated for exclusive use by the binary or binaries.
- Any other binaries used to exclusively decode or decrypt the binaries of interest.
- The chains of binaries used by the said decoder or decryptor, if they are used exclusively by the decoder or decryptor.

It can be seen that the list of entities required to be removed form a complete set of entities to remove to safeguard the binaries, and can be generally explained this way. The first item is trivial - they are the items of interest and have to be deleted. For the other items, since their existence cannot be plausibly explained without

the binaries, securely deleting them is therefore necessary to prevent having to disclose the existence of the binaries.

The next issue at hand would be how to proceed with deletion when required. Trivially, one could program a trigger that deletes each and every binary and entity of interest once hostility is detected. However, if there are many entities to be deleted, or the hostility cannot be predicted accurately, such technique could be cumbersome or even ineffective. Therefore, it is better to move these entities somewhere volatile and ‘pre-delete’ the entities from persistent storage before deletion is actually called for, restoring them to persistent storage only when the risk of disclosure and acquisition is within safe levels.

4.2.2 RAM-Based Method to Hinder Forensics

One possible method to hinder forensics in Android-based devices would be to run sensitive binaries off the volatile RAM. This would allow the operating system to reduce the traces present in persistent storage by acquiring off-device binaries and run them in RAM, which will be purged once the device is powered off and restarted.

Such a method is practical except for the fact that again, a local decryptor is required to decrypt the binaries if they are encrypted, and if the binaries are not, then they can be easily exposed during transfer. The local decryptor may thus have to be present on the device much like the local malware in [118].

Even if the entire decryptor and sensitive binary set is made off-device, it could still arouse suspicion as to the presence of decryption functions and a binary blob contained within, signaling the presence of an encrypted entity.

Taking this one step further, the entire operating system could be deployed in such a way to execute from RAM instead, with a plausibly deniable ‘clean’ image left on the persistent storage to mislead forensic agents. A stronger goal would be to have the operating system itself performing the ‘cleanup’ of persistent storage - thus not requiring user intervention for the process. This could take the form of an

operating system that boots and loads itself to RAM, then automatically engages in ‘cleanup’ of the persistent storage to remove traces of incriminating information. Then, only if a correct trigger is executed, the incriminating information can be written back to persistent storage.

4.3 The Android Operating System and Forensics

The Google Android operating system was launched in 2008 and at that time meant to run on smartphones as a mobile phone operating system. The Android OS runs on a modified Linux kernel, with a native library layer and process virtual machine (called Dalvik) upon which the application codes run on and utilize. The kernel and system layer components are hidden from the user, who should only use the user interface to operate the operating system.

The Android block diagram is shown in in Figure 4.1. The layers within the Android OS can be divided as follows (from top to bottom):

- Application Layer (the user interacts with this layer)
- Application Framework
- Native Libraries and Services (the Dalvik VM is at this layer)
- Linux kernel

Although meant that way in terms of usability, such intentions will not hold under forensics. Given an Android-powered device, a sufficiently capable forensics expert would be able to access the underlying system layers. Therefore, it has to be assumed that a digital device running Android, if acquired and put through forensics examination, may yield contents hidden at the layers below the application layer.

Nevertheless, accessing entities within a mobile device is not as easy as compared to say, a computer. For example, a device’s built-in persistent memory may

Figure 4.1: The Android block diagram.

have to be desoldered from the device to be analyzed, which is much harder than unplugging the cables on a hard drive.

4.4 Executing Android in RAM

Android is fundamentally a small operating system. Based on compiled source codes from the Android-x86 project [119], an entire operating system would take around 500MB, uncompressed. Given that mobile devices pack an ever increasing amount of RAM, the entire operating system (with the exception of the kernel) could be loaded into and run from there.

By default during image generation, Android-x86 compresses the main Android system into a compressed *squashfs* filesystem image, which is decompressed on the fly during runtime when run as a live disk. This is not desirable if one wishes to write to the filesystem as the *squashfs* filesystem image is read-only. Therefore, the generated image may have to be manually decompressed into a write-enabled partition on the persistent storage¹.

The Android-x86 project utilizes *busybox* as the init program of choice after kernel execution. At this point of execution, the initialization script could be modified to copy the Android system (containing the libraries, virtual machine, application layer and applications) to RAM (specifically, a *tmpfs* filesystem). Upon completing the *busybox* initialization script, execution is then transferred over to the Android system which would be in memory. The Android system then executes its own init program and script.

Android's init program allows for mounting of filesystems, therefore its init script could be written in such a way to mount the filesystem in persistent storage containing the Android system and remove the sensitive components outlined in the previous subsection. The script would also have to remove, from the copy of itself on persistent storage (recall that the system is executing what was essentially prior on the persistent storage), any commands relating to mounting and deleting the sensitive components as well as appending the init script.

When the Android init completes, the filesystem on persistent storage is now without any binaries of interest or other, related sensitive components. In addition, the Android init script on the persistent storage would not reflect the presence of commands to mount and delete those components, as they have been removed via the init script that is in RAM. These components continue to exist in the copy of the Android filesystem in RAM.

To restore these components and entries back to persistent storage, the reverse of the deletion script (i.e. copy deleted components from RAM to persistent

¹Alternatively, the image generation process could be modified to not compress the filesystem by default.

storage and restore commands to the persistent storage's Android init script) is done via a triggered script (e.g. a lock screen gesture trigger). Naturally, this has to be done before shutdown else the components will be lost.

The sequence of events are shown in Figure 4.2.

Figure 4.2: Executing Android (Android-x86) in RAM.

It is not hard to see how some semblance of anti-forensics can be achieved via such a measure. When discovery of the components are imminent, the device operator merely needs to switch the device off (without triggering the restore script) and the components will not be restored, thus analysis of the persistent storage will reveal only that the Android system runs off memory, which can be plausibly explained (e.g. better performance running off memory).

It is important to note though that any modification to the Android operating system, especially modifications designed to conceal actions, may be viewed with suspicion if discovered. In the demonstrator used to test the feasibility of executing Android in memory, no Android base code has been changed.

4.4.1 Managing the Communication Channel

Although the Android operating system is capable of running from memory, it still can leave traces of a sensitive binary's operation. For example, if the binary communicates with an external entity over a network, the communication may

be intercepted by an adversary. The operator may then be forced to explain and decode the intercepted communication, which could lead to the knowledge that some kind of secret binary has been used, even if the sensitive binary is no longer present. In hostile situations where legislation preventing self-incrimination is non-existent or does not hold, refusal to explain or decode the communication may also result in severe consequences.

One method to minimize such undesired outcome would be to communicate via a steganography-enabled or subliminal channels, where it is much easier to deny the existence of the actual communicated information, though the overhead may be high. However, since the containers are securely discarded once received and decoded by the receiver, the storage overhead is at least no longer an issue compared to when steganography is used to hide binaries.

A second method would be to utilize a communication channel that sends both a real and fake message simultaneously, where whether the real or fake message is revealed depends on the key entered. In a trivial example, the encrypted sequence $E_{K_1}(m_1)||E_{K_2}(m_2)$ where m_1 and m_2 being real and fake messages respectively could be plausibly explained as an encryption of only m_2 , with the $E_1(m_1)$ portion being a random padding (it will appear as random bits when decrypted using K_2). Naturally, a more sophisticated scheme would be required for practical purposes.

4.5 Evaluating a RAM-Based Android Instance

An image of the Android operating system, version 4.0 for the x86 platform (sources obtained from the Android-x86 Project [119]) is compiled with modifications made to the scripts to provide anti-forensics and support for the test system's hardware. Specifically, the anti-forensics portion of the scripts perform the following two things:

On Startup Copy the Android filesystem to RAM and transfer execution to the instance in RAM. From the instance in RAM, automatically delete contents and script entries from the persistent storage related to the copying of the

Android filesystem and execution transfer to RAM (in addition, restore the modification timestamp for modified entities to the original timestamp before modification).

On Shutdown Restore the deleted contents and script entries back to the persistent storage, from RAM. For the purpose of evaluation, the script performing this is triggered automatically when a shutdown is detected (blocking shutdown until the script completes). Like the portion of the script to do content deletion on startup, it restores the modification timestamp for modified entities to the original timestamp before modification.

The aim is to evaluate the system in terms of usability, and detectability of the anti-forensic measures. Before compilation, the source code for *busybox* is updated to the latest version, as the one provided by Android-x86 does not have certain features required for anti-forensics (e.g. the command *touch*)

The operating system is installed and run on a Lenovo X61s laptop. The amount of modification required to the scripts to handle the device's hardware is minimal, and most modifications are done for anti-forensic purposes. The installation process automatically decompresses and copies the Android filesystem to the disk, thus no manual decompression of the *squashfs* filesystem is required.

In terms of general performance, there is a negligible performance difference when running the operating system off RAM compared to running it off the disk, although theoretically RAM access should be faster than disk access. This can be attributed to the device's capabilities - a dual-core 2GHz processor with 2GB of RAM. Even with around 400MB used for the Android system, there is still about 1.6GB remaining, way more than what could be utilized by the Android operating system. In addition, it may also be attributed to the disk access pattern, which seemed more sparse compared to traditional desktop-based operating systems.

Nevertheless, running the operating system off RAM incurs overhead in terms of the time required to copy the filesystem from disk to RAM and delete the sensi-

tive components from disk. Compared to running the operating system from disk, the time it takes to boot the evaluation system is about 15 to 20 seconds longer². Shutting down properly (in order to trigger the anti-forensics portion to restore contents to persistent storage) takes some negligible few seconds longer on the evaluation system compared to a disk-based run without anti-forensic measures.

Next, the system is checked to ensure that the Android init script executes the anti-forensic measures it is programmed to do. Deleting files off the persistent storage and copying them back are accomplished without issues, but getting the init script to remove from or restore to the copy of itself on the persistent storage, the lines corresponding to the deletions are a little harder; using *sed*, a mistake in the line numbers would result in the wrong lines being deleted or added.

Hiding applications from the user interface is then tested. This entails renaming a .apk file (Android application package) in the applications filesystem directory (of the one in RAM) to another extension to simulate deletion, and observing the changes at the user interface, specifically the applications screen. It is found that right after the file extension is renamed, the application ceases to show on the applications screen. It reappears right after the extension is restored. In both cases, the time it takes from when the rename actually takes place, to the changes being visible, is less than 1 second. It can therefore be concluded that the user interface component monitors the applications filesystem directory and immediately applies changes once they are detected.

Given that deletion is tested to be successful, the next agenda would be to test if the Android operating system delays command execution. This is tested by letting the system run, up to when the scripts have deleted the sensitive contents from the persistent storage. The system is immediately powered down and the disk's filesystem analyzed. It is found that the deleted data and script entries remain deleted, thus the system is safe against the immediate power-down method of attempting to retain data.

²This is not a definitive indicator of performance, as the duration required depends on many factors including, among other things, the disk access speed, and may even be influenced by the different versions of the Android operating system

Overall, once a system has booted up properly (with deletions taking place), it is impossible to tell from the user interface that anti-forensic measures have been taken (and what those measures are). This is a disadvantage for first responders, who may have to identify if a device is engaging in anti-forensics. Without the ability to tell, and assuming the likelihood of triggers at locations within the system which is ready to execute when undesired behaviour is detected (e.g. jamming of radio signal, etc.), it would further disadvantage forensic examiners who could only gain access to the device later.

4.5.1 Discussion

In the system developed in Section 3, a fake data set (D_F) is one that contains no incriminating information, designed to deceptively project the absence of another data set (D_I) containing the incriminating information. In that system, depending on the trusted / non-trusted state, only one data set is presented, so it is an either-or situation. In this evaluation, incriminating information can be thought of as certain contents (e.g. applications) within the operating system that should be hidden from / removed before forensic analysis. The fake data set D_F can thus be thought of as the operating system deployment without such contents, while the incriminating data set (D_I) is one with those contents intact.

One major difference between the two systems (that in Section 3 and this one) is that in this system, both D_I and D_F exist concurrently when the system is running. However in this system, the set of entities required to obtain D_I , V_{D_I} (recall Equation 3.24) consists of among other things the volatile RAM itself, which can be easily tampered with by disconnecting the power source. In addition, as stated previously, extra triggers can be put in place to monitor the system and erase D_F when required. Such triggers may include for example detecting changes in the connected power source (i.e. whether a charger is connected to the device).

Consider a scenario where the evaluation system is deployed together with two additional triggers - one to detect if a charger is connected (and if not, purge

D_I by abrupt shutdown), and another that monitors the network interfaces for a command to purge D_I (again, via abrupt shutdown). Recall that abruptly shutting down the evaluation system would result in D_F residing in disk, and subsequent reboots will load D_F . V_{D_I} would now be $V_{D_I} = \{X, R, T_C, T_N\}$, where X is the device, R is the RAM (powered) on X hosting D_I , T_C is the trigger on charger disconnect, and T_N is the trigger on network command. In such a scenario, R and T_C are volatile, and can be manipulated to purge D_I . If the operator wants a more volatile system, then he or she can design the system deployment in such a way that V_{D_I} covers more triggers, thus more avenues to purge D_I . Therefore, the system security depends on the crafting of a well-designed V_{D_I} set, instead of like the system designed in Section 3.

4.6 Conclusion

This chapter shows the feasibility of affording anti-forensics via running a mobile operating system in RAM. Given that in a real device, access to the layers beneath the user interface may not be possible for first responders, such tactic, combined with possible trigger scripts strategically placed within the operating system may be used to remove evidence of any sensitive contents before the device can actually be analyzed by a forensics examiner.

This page is intentionally left blank.

Chapter 5

Threat Mitigation in Disruption Tolerant Networks

While much research has been done on improving the reliability and performance of disruption tolerant networks, the security aspect is often overlooked since network security would invariably incur some amount of overhead. Without proper security, adversaries would be able to participate in a targeted DTN, subsequently allowing them to launch further attacks into the network.

Security in DTNs can be generally classified into two categories - bundle protocol security and routing protocol security; corresponding to the protocols used in DTNs. Bundle protocol security deals with providing security for transacted bundles at the bundle protocol between sender-receiver DTN node pairs (e.g. source to destination, security-aware router to security-aware router, etc.), while routing protocol security on the other hand deals with the security of DTN routing protocols, involving issues pertaining to secure and reliable DTN routing.

The main challenge to DTN routing protocol security (and DTN protocol security in general) is the intermittent and unpredictable connectivity inherent in such type of networks. With intermittent connectivity, routing protocols employ strategies that do not require the establishment of end-to-end connected paths in order to route packets. While this indirectly negates the effects of end-to-end path-based

attacks (e.g. black hole, path cost manipulation, etc.), it still leaves a DTN open to other, non-path based routing protocol attacks, like flooding, impersonation, and Sybil attacks.

Another major challenge to DTN routing protocol security is that of the routing protocol design itself. In order to optimize performance, some routing protocols like MaxProp, RAPID, and PRoPHET rely on mutually exchanged routing information for better route and buffer management decisions. These exchanged information, while beneficial, can also be a vector for attack that could be manipulated by adversaries to disrupt routing in the network.

In addition, with its nature of permitting continuous upper-layer sessions over intermittent connectivity, challenges exist in verifying that DTN nodes, upon resumption of its DTN session, is free from any interference by external entities. This essentially brings into question the possibility that a node may be disconnected because it is being captured or compromised by an adversary.

5.1 The Case for Authentication and Authorization

In DTNs, the non-existence of routing protocol security allow nodes to freely participate in the network - something that could potentially be beneficial to increase available network capacity and connectivity. However this is true only if the nodes are honest; i.e. that they do not sabotage the network or work in ways to disrupt further the intermittent, unpredictable, and constrained network that is characteristic of a DTN. Without security, there is little guarantee that delinquent nodes can be prevented from joining the network, or once in, isolated or removed from the network.

The response to this issue can generally be seen to fall into two categories. On one hand there has been research to show that disruptions potentially caused by such delinquent nodes could be mitigated by designing better routing protocols,

such that intentional disruptions will not harm the network more than what normal disruptions cause. On the other hand, there has also been research to show that carefully crafted attacks can severely degrade the performance of the routing protocols.

5.1.1 Tactical Level Disruption Tolerant Networks

A tactical level DTN is defined as one where all the nodes in the DTN are at the same level, with no hierarchy in place (although the entire DTN may be controlled by some entity at a higher hierarchy). Nodes would most likely have similar equipment with similar communication capabilities, but that is not strictly required. Each node would be both an end terminal and a relay.

It is not hard to imagine a real-life tactical level DTN. Take for example a localized disaster zone where the communication infrastructure has been damaged. In order to communicate in such environment, rescuers would need to set up their own communication network. Given that communication blind-spots would likely exist in such areas, a DTN would be a potential choice for the overlay network in order to preserve session continuity, with the underlying convergence layers being whatever communication capabilities (e.g. UHF radio, WiFi, etc.) that their own communication devices are capable of. Such a scenario, where a DTN is formed by a group of rescuers at the same hierarchical level possessing devices with similar communication capabilities, is an example of a tactical level DTN.

Tactical DTNs can be used in many real-world scenarios, ranging from search and rescue in friendly grounds, all the way to battlefield communication in enemy territory. In search and rescue scenarios within friendly territories, it is more likely than not that the DTN is in use in a non-hostile environment (i.e. there would be few, if not no adversaries). This is a logical assumption since there are probably no adversaries around, and that there is probably little to gain by attacking a network used for peacetime / crisis operations.

This is in contrast to a battlefield scenario. In such a scenario, there can be

a lot to gain by attacking an enemy's network, be it to eavesdrop on communication, sabotage the network, or to jam completely the enemy's communication capabilities. From the point-of-view of the side that is (potentially) attacked, it is important therefore to set up DTNs that could not only fulfill their communication requirements (performance), but also prevent their adversaries from achieving their goals (security).

5.1.2 Attacks Against Disruption Tolerant Networks

In [98], it is shown that two types of attacks - flooding of non-deliverable packets and impersonation, are particularly effective against the MaxProp routing protocol. Though mentioned as standalone attacks, the attacks when described are in fact combination attacks. The flooding of non-deliverable packets, for example, is combined with falsification of routing information to ensure the non-deliverable packets continue to stay in the network. The impersonation attack (where adversaries impersonate message destination nodes), is combined with false acknowledgment flooding which results in nodes dropping packets thinking they have been delivered.

It should be noted that the attacks proposed in [98] require the adversaries to have prior knowledge about node identities in the network, for example to impersonate as. Although it can be argued that adversaries will eventually gain knowledge of the node identities, the question is whether they are actually required for launching successful attacks in a DTN.

In addition, as [98] focuses solely only on MaxProp, it would be interesting to find out how effective are attacks against other DTN routing protocols.

5.1.2.1 Flooding

Flooding is perhaps one of the most primitive forms of attack. It is essentially an attack where adversary nodes attempt to flood as many packets as they can into the network, resulting in resource contention and ultimately if successful, the

degradation of network performance. The attack therefore relies on having the adversary produce and inject packets at rates fast enough to overwhelm available network resources, which in a DTN, are the network channel capacity and the node buffer capacity.

In order to keep packets within the network, adversaries may decide to flood packets destined for existing nodes in the network, instead of a random address. This is done in the hope that the friendly nodes would not discard or lower the priority of the packets, since they are, after all, destined to a deliverable node.

A more advanced version of flooding involves adversaries flooding packets into the network destined for other adversaries. The goal here would be to ensure that the packets are deliverable, thus letting the routing protocol prioritize them for delivery. The catch here would be that the adversary receivers would not actually receive the packets, giving reasons such as insufficient buffer, etc. thus keeping the packet in the network (and kept propagating) until the message TTL expires. Figure 5.1 depicts how such an attack works.

A thing to note about the advanced flooding attack compared to the non-deliverable packet attack in [98] is that in [98], the routing tables of the friendly nodes have to be tainted with the address of the non-existent destination to keep the non-deliverable packets in the network, while in this attack, such tainting is not required - a friendly node's routing information is influenced by contact with the adversary nodes.

A third method of flooding involves spoofing the source of the packets to hinder nodes from detecting that the packets came from adversarial sources. Such an attack may involve simply changing the packet source to reflect a friendly node, or changing the path altogether to reflect a believable path. The aim would be to fool friendly nodes into thinking that the packet is legitimate, and that there is no reason to flag the packet as fraudulent.

(a) Adversary node X sends a packet to friendly node B, destined for adversary node Y.

(b) B forwards the packet to A, who has a higher chance of delivering the packet to Y.

(c) A attempts to deliver the packet to Y. Y declines with an excuse (e.g. insufficient buffer space).

(d) Y moves to a location closer to C. A now forwards the packet to C, as it has a higher chance of delivery.

(e) C attempts to deliver the packet to Y. Y again declines with an excuse. In the mean time Y generates a packet destined for X.

Figure 5.1: An advanced version of flooding.

5.1.2.2 Impersonation

The advanced flooding attack in the previous subsection can be thought of as a kind of basic impersonation attack; impersonation in a sense that the adversaries assume the character of a friendly node and subsequently floods messages to other adversary nodes who also assume the character of friendly nodes.

Another method of attack via impersonation is that of assuming the character of a friendly relay. In such an attack, the adversaries would impersonate friendly nodes and accept packets for relay. By passively gathering contact opportunities and building its routing data collection or outright falsifying them, an adversary could appear as a favourable delivery candidate for some messages. This would result in friendly nodes forwarding messages to the adversary for delivery. However, the adversary, upon receiving these messages, would mark them as delivered (communicating this to other nodes if the routing protocol allows so) and stop forwarding them, essentially creating a black hole.

Such an attack would be particularly effective against routing protocols that utilize acknowledgments to reduce the number of packets in the network. A depiction of such an attack is shown in Figure 5.2.

A defense mechanism against counterfeit acknowledgments is presented and tested in [99]. The mechanism involves accepting acknowledgments (and subsequently deleting acknowledged messages) if and only if the message has been, or is present on the node.

5.1.2.3 Hijacking

Node hijacking is a serious threat, in that friendly nodes are attacked, captured, and used by an adversary to attack the network. Traditional network security measures may not work in detecting such a threat, since the hijacked node appears to other nodes very much the friendly node it is, except for who is operating it.

Countering such threat therefore requires additional approaches in conjunction with network security protocols.

(a) Node A generates and sends a packet to node B, destined for node G.

(b) Node B forwards the packet to adversary X, who claims to have a higher chance of delivery to G.

(c) X finishes the transfer and notifies B. Everything seems normal as of this moment.

(d) X generates an acknowledgment for the packet and sends it to C and F. X then deletes the packet.

(e) C and F propagate the acknowledgment. B deletes the message once acknowledgment is received. A moves closer to D.

(f) A attempts to forward the packet to D, but D rejects stating that the packet has already been acknowledged, therefore delivered.

Figure 5.2: A relay impersonation attack with acknowledgment counterfeiting.

5.2 System Objectives and Requirements

5.2.1 High-Level Adversary Model

Traditionally, adversaries in network security are modeled as entities that only attack the network in certain, predefined ways. Such a model is valid considering that the general aim of network security protocols is to prevent the successful achievement of an adversary's goal on the network (whether it is to penetrate the network, jam communication, etc.). The adversary loses if it cannot achieve its goal, and it is most often assumed that the adversary then gives up attacking the network.

In a much more dynamic, real-life scenario, it can be assumed that network nodes, whether friendly or adversary, could be directly controlled by human operators. Being human controlled, it can further be said that such an adversary would therefore have the capability to analyze gathered information, adapt to a given situation, and would probably not give up that easily upon failing. What this means essentially is that such intelligent adversary may not continuously attempt failed methods in order to achieve its aim, but would probably adapt and change tactics in order to win. In addition, it can also be plausibly assumed that such adversary may not solely attempt to attack a network from the communication channel alone, but may attempt to attack it by attacking and capturing physical network assets like the network nodes and their operators.

Finally, it can be assumed that in the event of a node being captured or compromised, there will exist a delay between when a node is initially captured or compromised and when an adversary finally obtains sufficient usable knowledge about the actual workings of the node and network. Such concept of possession of knowledge about a captured or compromised node by an adversary can be loosely represented as follows:

$$\begin{aligned} K_0 &= \emptyset \\ K_t &= K_{t-1} \cup \chi_t \end{aligned} \tag{5.1}$$

where K_t symbolizes all the knowledge possessed by the adversary about the captured or compromised node at time t , and χ_t is the new knowledge obtained between times $t - 1$ and t . An adversary is said to have obtained sufficient knowledge about the actual workings of the network and node (and therefore able to use the node as a vector for network attacks) when K_t exceeds a threshold K_S ¹. Naturally, K_S can never be precisely quantified due to its nature. However, knowing that there exist a K_S , and that such K_S could possibly be attainable, measures could be taken to prevent the adversary from obtaining K_S , while steps are taken to inform other nodes in the DTN of what has happened. While all that are taking place, measures can also be taken to hide such acts from being discovered by the adversary.

5.2.2 Threat Mitigation Aims and Measures

Given the adversary model, the aims of the protocol can be refined as follows:

- An adversary should not be able to impersonate completely as a friendly DTN node.
- If a friendly node is captured or compromised by an adversary, it should be able to inform other friendly nodes about its situation while hiding such acts of informing from the adversary itself.
- Friendly nodes in the DTN should be able to use the node authentication information to mitigate threats that an adversary may launch.

In order to achieve these aims, the following three measures are used, namely:

- Non-interactive authentication
- Covert communication of node state
- Plausible deniability of covert communication

¹ K_S is essentially the sum of knowledge about the correct workings of the DTN node.

5.2.2.1 Non-interactive Authentication

Non-interactive authentication involves using identity-based signatures to secure DTN beacon messages, allowing authentication of these messages to happen without the need for two-way communication. This would therefore maintain the same number of message transfers as a standard beacon message system (albeit larger message size), but allows a receiver to verify that a beacon message is authentic and valid, and not injected.

Such a move is the first step in keeping the DTN free from adversaries that attempt to participate in the network by attempting to impersonate friendly DTN node identities (since they would not be able to generate authentic signatures required to coerce other nodes into believing their identity claim).

Additionally, non-interactivity allows nodes to authenticate their counterparts without having to disclose any information regarding the authentication outcome or subsequent actions undertaken. It means that an adversary would not know whether or not its authentication attempt was successful.

5.2.2.2 Covert Communication of Node State

Covert communication of node state involves nodes utilizing translucent sets in order to consistently communicate their current state (i.e. whether it is compromised / not compromised) to other nodes. In order to maintain covertness, such communication would be piggybacked along beacon message announcements. Such piggybacking also reduces the message overhead to transmit the information.

5.2.2.3 Plausible Deniability of Covert Communication

Plausible deniability of covert communication would entail having the node, with cooperation from its valid operator, provide a plausibly deniable reason to reduce suspicion that covert node state communication is taking place. Such plausible deniability is especially important in protecting and allowing covert communication to continue when a node is compromised, where an adversary could be analyzing

the compromised node and / or interrogating the operator to figure out how the node operates (and therefore may come across the covert communication though not knowing what it is).

5.3 Proposed Protocol

The proposed protocol can be divided into two parts, both operating together to provide threat mitigation:

- A Shamir IBS based beacon message protection mechanism.
- A translucent set based plausibly deniable covert node state communication mechanism.

It is assumed that the DTN would be a closed, tactical-level DTN, and there is a command center H that grants membership and serves as the key generation authority (KGA).

It is also assumed that each DTN node in the network is controlled and operated by a local human operator, and they can be physically compromised or captured, but as per real-life, there will be a delay before an adversary finally obtains sufficient usable knowledge as per Equation 5.1.

It is finally assumed that the valid human operator of a captured or compromised node may, within a certain time frame from the node being captured or compromised, interfere with the node². This time frame would be the time between when the node is initially captured or compromised, to the time required by the adversary to obtain either K_S or sufficient knowledge that the threat of interference by the valid operator exists.

The protocol operates within a few stages, namely:

- Network pre-deployment.
- Signature key generation.

²e.g. Via a dead-man's switch or command.

- Beacon message construction and announcement.
- Beacon message verification.

5.3.1 Network Pre-Deployment Phase

As per Shamir IBS, the command centre H first selects two primes p, q . It then computes $n = pq$ and $\phi(n) = (p-1)(q-1)$, and generates a d, e pair such that:

$$\begin{aligned} 1 < e < \phi(n) : \gcd(e, \phi(n)) &= 1 \\ d : de &\equiv 1 \pmod{\phi(n)} \end{aligned} \tag{5.2}$$

H then further selects two primes p_t, q_t such that $p_t q_t < n$ to be used for translucent set member generation. It then computes $n_t = p_t q_t$ and $\phi(n_t) = (p_t - 1)(q_t - 1)$, and generates a δ, ϵ pair such that:

$$\begin{aligned} 1 < \epsilon < \phi(n_t) : \gcd(\epsilon, \phi(n_t)) &= 1 \wedge \epsilon \neq e \\ \delta : \delta\epsilon &\equiv 1 \pmod{\phi(n_t)} \end{aligned} \tag{5.3}$$

In addition, H selects a one-way function ρ such that $\rho : \mathbb{Z}_{n_t} \rightarrow \mathbb{Z}_n$.

Finally, H selects a one-way function f such that $f : \{0, 1\}^* \times \{0, 1\}^* \rightarrow \{0, 1\}^\lambda$.

5.3.2 Signature Key Generation

Each DTN node μ submits their endpoint identifier i_μ to H . H then generates the signature key g_μ for μ , where:

$$g_\mu = i_\mu^d \pmod{n} \tag{5.4}$$

H returns g_μ , and at the same time also hands over $n, e, f(a, b), \delta, \epsilon, n_t$ and $\rho(c)$, to μ . g_μ , as the signature key, δ, ϵ, n_t , and $\rho(c)$ must be treated as secret by the node. Generally, it is assumed that the transfer of $g_\mu, n, e, f(a, b), \delta, \epsilon$,

n_t and $\rho(c)$ are done over a secure channel (specifically, only g_μ , δ , ϵ , n_t and $\rho(c)$ require transmission over a secure channel).

After generating signature keys for all DTN nodes, H can proceed to destroy the secrets d , p , q and $\phi(n)$ to prevent further key generation. H can also proceed to destroy p_t , q_t and $\phi(n_t)$ to further safeguard the translucent set key.

5.3.3 Beacon Message Construction

In the proposed protocol, a beacon message announcement by a DTN node serves three purposes, namely:

1. To provide the information required by other DTN nodes to establish links.
2. To identify itself as a valid DTN node.
3. To notify other DTN nodes about its current state.

For the purpose of the equations that follow, the beacon message information (i.e. the *actual* information used in constructing links) of a node μ shall be represented as β_μ .

In addition to β_μ , the protocol would require the following to construct a complete beacon message:

- the current timestamp T ,
- the beacon expiry duration τ , and
- the node endpoint identifier i_μ .

Note that β_μ and T are real-time values (i.e. obtained at time of constructing the beacon message). τ may or may not be real-time, and may depend on for example whether the implementation adopts a dynamic or preset beacon interval.

Node μ starts by selecting a random v such that $v \in \mathbb{Z}_{n_t}$ and $v > 1$, proceeding to compute:

$$\begin{aligned}
r &= \rho(v) \\
t &= r^e \mod n
\end{aligned}
\tag{5.5}$$

If $\rho(v) = 1$ or $\rho(v) = 0$, a new v is generated and Equation 5.5 repeated.³

By using knowledge of the maximum possible length of the signature to be generated ($2|n|$), as well as the capabilities of the convergence layer which the beacon message will be advertised (e.g. bandwidth, latency, etc.), a padding length h is determined such that $h > |n_t|$.

A member of the translucent set with length h is then computed. The construction of the translucent set is adopted from a generic construction outlined in [120], which for the purpose of this protocol, is detailed towards an RSA-based construction and is outlined from the perspective of the constructor.

Let γ be an instance of the RSA encryption function, such that:

$$\begin{aligned}
\gamma(x) &= x^\epsilon \mod n_t \\
\gamma^{-1}(y) &= y^\delta \mod n_t
\end{aligned}
\tag{5.6}$$

Let $k = h - |n_t|$, node μ then computes:

$$\forall i = 1 \dots k, (\gamma^i(v) \mod 2) = b_i \tag{5.7}$$

$$v_F = \gamma^{k+1}(v) \tag{5.8}$$

Bit b is a hard-core bit for RSA [121], since given only $\gamma(x)$, ϵ , and n_t , the probability of determining b from x is the same as random guessing.

The values of b_i for all $i = 1 \dots k$ is then concatenated with v_F to form:

³If $r = \rho(v) = 1$, then $r^{f(t, M_\mu)} = 1$, thus $s = g_\mu$, essentially leaking the signer private key. If $r = \rho(v) = 0$, then $\sigma = (t, s) = (0, 0)$.

$$x_t = (v_F, b_1, b_2, \dots, b_k) \quad (5.9)$$

x_t is a member of an RSA-based translucent set with the parameters $\{\delta, \epsilon, n_t\}$.⁴

Node μ then concatenates i_μ , β_μ , T , τ , and x_t to form $M_\mu = i_\mu || \beta_\mu || T || \tau || x_t$.

It then signs M_μ as follows:

$$\begin{aligned} s &= g_\mu \cdot r^{f(t, M_\mu)} \mod n \\ \sigma &= (t, s) \end{aligned} \quad (5.10)$$

Node μ then advertises (σ, M_μ) as per the DTN implementation. (σ, M_μ) would then be picked up by neighbouring DTN node, verified, and processed.

5.3.4 Beacon Message Verification

The beacon message verification process ensures that:

1. the entire beacon message has not been tampered with,
2. the beacon message comes from a friendly DTN node,
3. the beacon message is not a replay, and is the latest for that particular advertising node⁵, and
4. that the advertising DTN node is not compromised or sending the beacon message under duress.

A friendly receiver of a beacon message advertisement will be in possession of the advertised message $(\sigma, M_\mu) = ((t, s), i_\mu || \beta_\mu || T || \tau || x_t)$, as well as the components n , e , $f(a, b)$, δ , ϵ , n_t and $\rho(c)$.

⁴ δ is used in the verification of translucent set membership, and must be kept secret.

⁵It is possible that due to the intermittent nature of DTNs, an earlier beacon message may be picked up after a later one, either due to deliberate or circumstantial causes.

The verification process starts by verifying the Shamir IBS signature for message M_μ :

$$s^e \equiv i_\mu \cdot t^{f(t, M_\mu)} \pmod{n} \quad (5.11)$$

If Equation 5.11 holds, the message has not been tampered with along the network, and that it came from the advertising DTN node μ .

The next step is to verify that the beacon message is not a replay, and is the latest for that advertising node μ . The protocol requires that the receiver stores the timestamp attached to the last beacon message stored from node μ , denoted as T_{L_μ} . Let the current receiver timestamp be T_{now} . The message M_μ is verified not a replay, and is the latest, if the following holds:

$$(T_{now} \leq T + \tau) \wedge (T_{L_\mu} \leq T) \quad (5.12)$$

The first part of Equation 5.12 checks that the message has not expired, while the second part checks that the message is newer than the previously processed one.

The final step in verification is to verify that the DTN node μ is not compromised and / or advertising under duress. This is done by verifying that the following holds for the received x_t and t (k is essentially $|x_t| - |n_t|$):

$$\forall c = 1 \dots k, (\gamma^{-c}(v_F) \bmod 2) = b_{k+1-c} \quad (5.13)$$

$$t = \rho(\gamma^{-(k+1)}(v_F))^e \pmod{n} \quad (5.14)$$

Equation (5.13) verifies that the received x_t is a member of the translucent set with the parameters $\{\delta, \epsilon, n_t\}$, while Equation 5.14 verifies that x_t is coupled to σ

(with successful verification at Equation 5.11). If Equations 5.13 and 5.14 hold, the receiver can strongly deduce that node μ is not compromised or sending the beacon message under duress.

If the received beacon message passes all the verification tests, the beacon message information β_μ is then processed, and the value of T_{L_μ} for that advertiser is updated to T_{now} .

5.4 Covert Signaling and Plausible Deniability

From the previous section, it can be seen that the protocol utilizes a common random seed v to generate both the signature σ and the member of the translucent set x_t . During the verification process, once it can be shown that the received t can be derived from the received x_t , a beacon receiver can deduce that the advertiser is not sending the beacon under duress.

The prerequisite for an adversary to successfully achieve full network participation would be for it to first successfully capture or compromise a friendly node. This is because without access to a friendly node, the only security-related item that an adversary could see via eavesdropping on the DTN would be σ , which accompanies every beacon message and is quite clear of its function. x_t would be seen as just random padding (due to its nature as member of a translucent set), and without knowing the real function of x_t , an adversary would not be able to know the link between x_t and t .

Recall the notion of K_S , which is defined as the knowledge threshold in which an adversary would, upon attaining, be able to manipulate the beacon message protocol. In general, this could be said to include three things, namely the knowledge of:

1. the generation of σ , with the corresponding generating key g_μ ,
2. the existence and generation of x_t , with the corresponding ϵ ,
3. the existence of $\rho(c)$ and δ , and the link between x_t and t .

If an adversary manages to discover all the three items above in a captured / compromised node, then that adversary would be able to fully participate in the network as though it is that node. It will not be able to with anything less than that.

In the event of a node capture, the protocol is designed such that with collusion from the friendly operator, the signature key g_μ would be sacrificed to the adversary. The idea would be that since an adversary would have seen σ being sent over the DTN, the adversary would probably have suspected the existence of a signature key present within the node.

At the same time g_μ is being sacrificed, the friendly operator would have to achieve one of two things, namely either:

- purging of the translucent set secret $\{\delta, \epsilon, n_t\}$, or
- purging of the one-way function $\rho(c)$.

If the friendly operator manages to purge $\{\delta, \epsilon, n_t\}$, then the adversary will not be able to create or verify received x_t values. If the adversary continues to participate in the network using g_μ , the adversary would have to substitute x_t with another value, which would trigger verification failure in Equation 5.13 at the receiving nodes. This would signal to the beacon receivers that the advertising node is no longer in control of a friendly operator.

If on the other hand the operator purges $\rho(c)$, the adversary would not be able to create x_t values that are coupled to t or verify the link between received x_t and t portions. Since this would trigger verification failure in Equation 5.14 if the adversary continues to participate in the network, beacon receivers could conclude that the advertising node is not in control of a friendly operator.

Such covert signals, when sent, are sufficient for the rest of the DTN nodes to deduce that a given node is captured / compromised. Considering that the entities to be purged are small in nature (either three values or a function), the kill switch to activate the purge should be implementable in a sufficiently discreet manner to avoid arousing any suspicion when triggered by the friendly operator.

A summary of how the disclosure of the components affect visibility and capability of the adversary is presented in Table 5.1.

Table 5.1: Visibility of Components Relative to Disclosure

Disclosed	Visible	Capability
None	$M_\mu = i_\mu \beta_\mu T \tau x_t$ $\sigma = (t, s)$ x_t random, unrelated to t .	None
g_μ	$M_\mu = i_\mu \beta_\mu T \tau x_t$ $\sigma = (t, s)$ x_t random, unrelated to t .	Can utilize g_μ to generate σ using random r for t and random x_t in M_μ .
$g_\mu, \rho(c)$	$M_\mu = i_\mu \beta_\mu T \tau x_t$ $\sigma = (t, s)$ $\exists \rho(c)$ x_t random, unrelated to t .	Can utilize g_μ to generate σ using random r for t and random x_t in M_μ . Knows that a one-way function $\rho(c)$ exists.
$g_\mu, \{\delta, \epsilon, n_t\}$	$M_\mu = i_\mu \beta_\mu T \tau x_t$ $\sigma = (t, s)$ $x_t = (v_F, b_1, b_2, \dots, b_k)$ $\forall c = 1 \dots k, (\gamma^{-c}(v_F) \bmod 2) = b_{k+1-c}$ x_t unrelated to t .	Can utilize g_μ to generate σ using random r for t . Can generate x_t in M_μ using random v .

Continued on next page...

Table 5.1 – continued from previous page

Disclosed	Visible	Capability
$g_\mu,$	$M_\mu = i_\mu \beta_\mu T \tau x_t$	Can utilize g_μ to
$\{\delta, \epsilon, n_t\},$	$\sigma = (t, s)$	generate σ using $r =$
$\rho(c)$	$x_t = (v_F, b_1, b_2, \dots, b_k)$	$\rho(v)$. Can generate
	$\forall c = 1 \dots k, (\gamma^{-c}(v_F) \bmod 2) = b_{k+1-c}$	x_t in M_μ using v .
	$t = \rho(\gamma^{-(k+1)}(v_F))^e \bmod n.$	Complete disclosure
		achieved.

5.4.1 Plausible Deniability and Disinformation

When implemented properly, the protocol should continue to function when certain entities are purged by either substituting with random values (for items that are generated using the purged entities), or bypassing the function altogether (for functions that require purged entities to operate). This could be implemented quite simply by having a branch at such functions, where the proper branch is taken only when certain entities (e.g. $\{\delta, \epsilon, n_t\}$ or $\rho(c)$) are present.

Such implementation provides a degree of plausible deniability for the friendly operator due to two reasons:

- it is hard to prove that the operator actually knows what is going on within the protocol beyond merely being a user of it, more so when the protocol in question is a relatively low-level DTN protocol.
- while the branches show that the functions do indeed exist and perform actions such as generating translucent set values when certain entities are present, it cannot be proven, without the purged entities, that the protocol has actually executed such functions.

Such lack of certainty could be then played to the advantage of the friendly DTN nodes by engaging in disinformation towards the captured node. Assuming

that the adversary, upon capturing a node would decide to test if the captured node could successfully participate in the DTN, the other friendly nodes, having already been discreetly notified of the capture via the covert signals, could engage in disinformation towards the captured node by for example deliberately supplying false beacon information or bundles which appear real.

5.5 Security Comparison to Bundle Authentication Block

Given that this protocol is designed to cater for beacon message security and provide for degrees of security in the event of a node capture, there is no protocol out there that it could be directly compared to. The closest comparison could be made to the Bundle Authentication Block (BAB) [82], if assuming the BAB mechanism could be extended to beacon messages, and that all nodes are set up as security-aware forwarders. In such a setup, the proposed protocol provides a relatively higher level of protection and efficiency due to:

- the protocol allowing provisions for defense (with operator intervention) and plausible deniability in scenarios of node capture.
- the utilization of identity-based signatures which eliminates the need for shared symmetric keys (though the network deployment may take longer due to the need for signature key generation).
- the use of a shared RSA key to generate translucent set members which are then used to signal a node's state to other nodes within the network.

Comparing the protocol to the 802.11w standard [102] (which shares some similarities in terms of protecting a network from spoofing via message-integrity protection of management components), the difference is that 802.11w is meant for access point (AP) to client station (STA) security and that a Pairwise Transient Key (PTK) is required to be generated for each connecting STA, while the scheme

proposed here is meant for inter-node security. The scheme presented here also allows for authentication to be performed concurrently with the message integrity check due to the use of identity-based signatures. Therefore, authentication can be accomplished in one message transfer. In addition to that, the scheme allows for node state to be communicated in a discreet manner.

5.6 Improving the Protocol

5.6.1 A More Efficient Construction for the Translucent Set Member x_t

Recall that in Equation 5.7, the construction of the translucent set member x_t involves running an RSA modulo exponentiation for $k + 1 = h - |n_t| + 1$ times (h being the padding size for the beacon message, n_t being the modulus of the translucent set generating key, and $h > |n_t|$).

As it has been discovered in [122] that up to $O(\log \log N)$ bits are hard-core for RSA, it means that the construction of x_t could be sped up by taking $\log \log N$ bits per RSA modulo instead of a single bit. This would reduce the amount of modulo exponentiations required by a factor of Figure 5.3 (i.e. $\lceil k/(\lfloor \log \log N \rfloor) \rceil + 1$), which translates to a faster construction. Consequently, the verification process also benefits from having the same number of modulo exponentiations as per required during construction.

Figure 5.3 shows the median time to generate both a translucent set member x_t and subsequent identity-based signature σ using the different methods for constructing x_t . With a 2048-bit modulus, it can be seen that the median time using the $\log \log N$ method is far less than what it takes compared to using the single bit method. For example, the median time for generating an x_t and σ pair with a 24-bit translucent bit string is slightly less than 0.08 seconds, while it takes 0.2 seconds for the same bit-length using the single bit method.⁶

⁶The experiment was conducted on a machine equipped with an Intel Core 2 Duo E6700, running at 2.66GHz. The executable used to generate the signatures utilizes the GNU Multiple

Figure 5.3: Median time for x_t and σ pair generation using different x_t generation methods.

5.6.2 Implementing the Security Protocol at the Routing Protocol

For routing protocols that exchange routing data to facilitate routing (e.g. PProPHET, MaxProp), the security protocol can be moved up to the routing protocol, thus executing the security protocol only when there is a contact. Such a move could reduce the number of times the security protocol is executed, since the number of actual contacts may be less than the number of beacon messages sent. Implementing the security protocol in such a way would entail securing the routing data, instead of the beacon message.

By moving up to the routing protocol, the beacon expiry duration τ would no longer be required during construction of the signature, since the signature expires the moment the contact ends. M_μ is therefore $M_\mu = i_\mu || \{\text{routing data}\} || T || x_t$, where i_μ being the node's identity, T being the node's current timestamp, and x_t

Precision Arithmetic Library [123].

the node-generated translucent set member.

Moreover, instead of verifying that Equation 5.12 holds, since τ is no longer present, a node now only needs to verify that $T_{L_\mu} \leq T$. Recall that T_{L_μ} is the timestamp of the last signature from a node μ (thus corresponding to the time the last successful contact is initiated), stored for every node encountered. Verifying that $T_{L_\mu} \leq T$ however does not remove the possibility of an adversary hijacking the node μ , thus x_t must still be verified.

The question is then, what to secure in the routing protocol. While it is trivial to just secure every single transaction at the routing protocol, it is not the case that such extreme measures are required. In fact, assuming that a contact session is non-resumable, signing the initial routing data transacted would be sufficient. The signature could then be verified, and for that entire contact session, the node at the other end would be authenticated.

Taking an example, consider a typical PROPHET routing protocol [78]. When two PROPHET nodes establish contact with each other, they first exchange their delivery predictability information before engaging in message transfer. Using the proposed modification here, instead of a node transferring just its delivery predictability information (let's call this m_μ), it would instead 1) concatenate that information with i_μ , T , and x_t to form $M_\mu = i_\mu || m_\mu || T || x_t$, 2) cryptographically hash M_μ to get $H(M_\mu)$, 3) sign $H(M_\mu)$ to get $\sigma = (t, s)$, and finally 4) transmit $(\sigma, M_\mu) = ((t, s), i_\mu || m_\mu || T || x_t)$ over to the receiver.

For the receiver, it needs to 1) verify σ (Equation 5.11), 2) verify that $T_{L_\mu} \leq T$, and 3) verify x_t and t (Equations 5.13 and 5.14). Once that is successfully done, the receiver can proceed to process m_μ , and it can consider the sender authenticated for as long as the contact session does not disconnect.

The overhead incurred when implementing the security protocol at the routing level is thus as follow:

Network overhead Additional overhead is required to send $|\sigma| + |M_\mu| - |m_\mu|$ bytes. $|\sigma|$ is bounded by $2|n|$, where $|n|$ is the signature key length. x_t in

$|M_\mu|$ is bounded by $|n_t| + \zeta$, where $|n_t|$ is the translucent key length and ζ is the number of translucent bits.

Processing overhead For a sender, additional processing is required to generate (σ, M_μ) . The most intensive processing happens during the generation of x_t and the subsequent identity-based signature σ . The overhead required for such an operation has been discussed in the previous subsection. On the receiver end, the verification process can be thought of as a reverse of the signature generation process, therefore the overhead required at the receiver is similar to that of the sender.

Storage overhead Receivers are required to store T_{L_μ} (the last encountered signature timestamp) values for every encountered node. Assuming that the total number of valid nodes (nodes which will be able to successfully authenticate) in the network is α , the storage overhead required to store T_{L_μ} values is bounded by $\alpha|T_{L_\mu}|$.

5.7 Utilizing the Information from the Security Protocol

The security protocol allows a node to know if another node is a valid node and whether or not it is under friendly control. This information could be used to thwart attacks within a DTN.

5.7.1 Using the Information to Prevent Flooding Attacks

Utilizing the knowledge of whether a node is valid and is under friendly control, measures can be taken to mitigate the effects of an adversary node launching a flooding attack within the DTN.

Given that flooding has been shown to be ineffective against MaxProp, a set of simulations is run instead to demonstrate the effects of flooding against PRoPHET

and Spray and Wait, and how, by being able to single out adversary nodes, the effects of flooding can be mitigated. It demonstrates how the information can be used to disallow adversaries from participating in and subsequently attacking the network.

A set of simulations is performed with the P_{Ro}PHET and Spray and Wait protocols, using a modified version of the Opportunistic Network Environment (ONE) simulator [124] running on the same, random seeds. The set of simulations for both routing protocols are first run with 25 friendly nodes, and repeated again with 100 friendly nodes.

Adversary nodes are deployed as additional nodes. This reflects better the real-world scenario of a DTN coming under direct, open attack from adversaries, given that more often than not adversaries in such kind of attacks use their own nodes instead of reconfiguring existing nodes to do their bidding (which is harder to accomplish for the adversary)⁷.

In the simulations, each adversary adopts the flooding tactics described in the first two paragraphs of Section 5.1.2.1, and generates a substantial amount of messages to flood the DTN. Up to 12 adversaries are simulated for the 25-node scenario (with each adversary generating on average 600 messages per hour), and up to 50 adversaries are simulated for the 100-node scenario (each adversary generating on average 120 messages per hour). Flooded messages are crafted such that they are destined for existing, valid nodes, thus potentially keeping them in the network longer than they would be if they are sent to random made-up destinations. Mitigating the threat of flooding involves the following from friendly nodes:

1. Nodes exchange link setup information (both P_{Ro}PHET and Spray and Wait; from beacon message) and routing information (for P_{Ro}PHET).
2. If the node at the other end of the connection cannot be authenticated via

⁷This is in contrast to a scenario where adversaries infiltrate the DTN by capturing and operating friendly DTN nodes. In the simulated scenario, signature verification would have failed, and therefore verifying x_t is redundant. For the infiltration scenario, since σ may be successfully verified, verification of x_t is required to let the verifier know if the node is compromised.

the received information, it is flagged and stored in a blacklist containing such nodes.

3. The node receives and forwards packets as per normal.
4. When the node buffer is full, the node first selects packets that originate from nodes in the blacklist for deletion.

Each friendly node generates messages at an average rate of 10 messages per hour (for the 25-node scenario) and 12 messages per hour (for the 100-node scenario); each message between 512kB and 1.5MB in size. All nodes have a buffer size of 256MB each, and are equipped with 3 radios, with a range of 40 meters (54Mbps), 70 meters (300Mbps), and 300 meters (70Mbps) respectively. The nodes are free to move around a 5km by 5km open terrain, and adopt a random waypoint movement model.

Each simulation instance is run for 43200 simulated seconds (12 simulated hours). A summary of the simulation parameters can be found in Table 5.2. A snapshot of the terrain layout and radio range for both the 25-node and 100-node scenarios are shown in Figure 5.4.

(a) 25 Friendly Nodes (12 adversaries) (b) 100 Friendly Nodes (50 adversaries)

Figure 5.4: Terrain layout for simulation of flooding attacks.

Table 5.2: Summary of Simulation Parameters for Flooding Attacks

Parameter	Value
Number of friendly nodes	25 or 100.
Terrain size and type	5000m by 5000m open terrain.
Node movement model	Random waypoint; movement speed at between 1.5ms^{-1} to 2.5ms^{-1} . Stops at end of each waypoint for between 0 to 900 seconds (0 to 15 minutes).
Node radio	3 radios; range of 40 meters (54Mbps), 70 meters (300Mbps), and 300 meters (70Mbps) respectively.
Node buffer	256MB.
Node communication rate	Average 10 messages per hour per node (one new message generated from an arbitrary friendly node between 0 and 29 seconds) for 25-node scenario, average 12 messages per hour per node (one new message generated from an arbitrary friendly node between 0 and 6 seconds) for 100-node scenario; each message is between 512kB and 1.5MB in size with a TTL of 360 minutes (6 hours).
Simulation duration	43200 simulated seconds (12 simulated hours).

Before executing the simulations proper, the cumulative contact capacity density is first determined. Contact capacity is defined as the amount of data transferable between two nodes within a contact window (i.e. the time these nodes are within radio range of each other). This is done by obtaining the contact window for each encounter, and multiplying them with the radio speed for that encounter.

However since the ONE simulator does not report the radio in range for each contact window, simulations to determine contact capacity was performed by repeating a same-seed simulation three times, with each time only a single radio is enabled. The contact capacity readings from each simulation run are then aggregated to form a final, cumulative contact capacity density chart. This was repeated for various combinations of friendly-adversary numbers, and the results are shown in Figures 5.5 and 5.6.

In Figure 5.5, it can be seen that around 85 percent of the contact capacity between friendly nodes exceed twice the node buffer size of 512MB. This means that it is likely that during an encounter, a pair of friendly node would be able to transfer their entire buffers over to each other. A similar observation can be made for the contact capacity between adversary and friendly nodes in Figure 5.6, with around 80 percent of them exceeding a single node buffer capacity of 256MB. For the case between adversary and friendly nodes, it is sufficient to look at the buffer capacity of a single node (i.e. that of the friendly node's), the adversary will always decline to receive messages. Given the observed contact capacity distribution, the effects of channel congestion should be negligible, and congestion experienced, if any, should be that of buffer congestion.

The packet delivery rate (of good packets) from the set of simulations are shown in Figure 5.7. It can be seen that the mitigation measure results in sustained delivery rates even under the most severe attack (PRoPHET: $\sim 70\%$ at 25 nodes, $\sim 33\%$ at 100 nodes; Spray and Wait: $\sim 92\%$ at 25 nodes, $\sim 92\%$ at 100 nodes). Simulation runs without authentication show a sharp decrease in performance as the number of adversaries increases to maximum (PRoPHET: $\sim 11\%$ at 25 nodes + 12 adversaries, $\sim 8\%$ at 100 nodes + 50 adversaries; Spray and Wait: $\sim 13\%$ at

(a) 25 friendly nodes

(b) 100 friendly nodes

Figure 5.5: Contact capacity between friendly nodes for simulation against flooding.

(a) 25 friendly nodes

(b) 100 friendly nodes

Figure 5.6: Contact capacity between adversary and friendly nodes for simulation against flooding.

25 nodes + 12 adversaries, $\sim 10\%$ at 100 nodes + 50 adversaries).

Note that under the threat mitigation measure, since deletion happens only when a node's buffer is full, it means that an adversary's packets may invariably get forwarded to other friendly nodes. Moreover, as other friendly nodes may have not encountered the said adversary in the early phases of the simulation, it could result in the packets being forwarded for some time before being removed. However, it can be said that such forwarding does not result in detrimental performance impact (based on the results in Figure 5.7), and in fact can be used to deceptively project that the adversary's packets are being forwarded.

5.7.2 Using the Authentication Information to Prevent Source Spoofing Attacks

In a source spoofing attack, an adversary modifies the packet source to hide the packet's origin, which if known may lead to the packet being discarded. Doing so however may not stop the spoof from being discovered - eventually, a node which has been spoofed will know it when it receives a packet for forwarding purportedly from itself of which it has not generated.

To counter such a problem, routing protocols may be modified to do a spoof-check; i.e. if a node discovers a packet purportedly by itself, checks its log to see if it has generated such a packet. If it has not generated the packet, then the node may flag the path as hostile - the logic being that one of the nodes in the path is the originator of the spoofed packet. The node can then weed out any packets that have ever traversed the path, knowing that packets from such a path are likely spoofed. Such a technique however, would require that path information be part of the packet and updated at every hop.

Another way of countering the problem is to utilize the security protocol, and then perform packet tagging. This would be easier to implement as it requires less modification on the routing protocol (merely implementing the security protocol at the beacon message or routing data level, and a packet tagging / cleanup

(a) 25 friendly nodes

(b) 100 friendly nodes

Figure 5.7: Packet delivery rate against deliverable destination flooding.

mechanism).

To demonstrate the effectiveness of using the authentication information to

prevent spoofing attacks, delivery rate simulations are performed for a scenario where spoofing, in conjunction with flooding, could render severe performance degradation.

The simulations are first performed with 25 friendly nodes, divided into 5 circle-shaped clusters. The cluster center points are arranged in a single line from left to right, with the nodes not being able to move beyond the adjacent clusters' center point (the movement range is the diameter of the circle). The nodes in the left-most cluster function as receiver-only nodes, and all other nodes in the other clusters would create and forward packets to those five nodes. Such a scenario would entail the right-most cluster having to forward packets through the most hops to reach the receiver cluster, and therefore, based on the cluster's distance to the receiver, would experience the greatest disruption due to spoofed packets.

The adversaries are divided into four adversary clusters (also circle-shaped), each one of them placed between friendly clusters. Adversaries in a cluster may not move beyond the center points of the immediate friendly clusters on either side. Simulation is performed with one, two, and three adversaries in each adversary cluster (for a total of 4, 8, and 12 adversaries), each adversary generating 20 messages per hour destined for the receiver cluster.

Spoofing is simulated in such a way that adversaries can only spoof messages belonging to nodes that they actually know exist, either through a message, routing metadata, or direct contact. This adds realism to the scenario since given their limited movement range, an adversary at one end of the terrain may not know the existence of nodes at the other end until it chances upon information of their existence.

For the simulations, a more aggressive packet tagging technique is used, in which the buffer is checked after each contact instance to discard undesired packets. For the path filtering (pattern filtering) technique, a packet is discarded if it has traversed a flagged path, while for packet tagging with the authentication information (hereby known as authenticated filtering), a packet is discarded if it does not come from an authenticated contact session. Both schemes strive to

keep the buffer free from undesired packets by removing them as fast as they are capable of being discovered.

The PRoPHET routing protocol is used, and the simulations are run using two buffer management strategies - FIFO (which was used in the previous subsection), and most forwarded (MOFO). The FIFO buffer management strategy is self-explanatory; packets that are queued the longest gets dropped first when the buffer is full. MOFO on the other hand, drops packets that have been forwarded the most.

A summary of the simulation parameters can be found in Table 5.3. A snapshot of the terrain layout and radio range for the simulation is shown in Figure 5.8.

Figure 5.8: Terrain layout for source spoofing attacks (25 friendly nodes).

The results of the simulation is shown in Figure 5.9.

Both the pattern filtering and authenticated filtering technique yield similar performance, though as mentioned earlier, the pattern filtering technique would require a detection algorithm to be incorporated into the routing protocol and path information to be included in the packets, whereas less modification is required for authentication and its filtering technique. Moreover, given that pattern filtering requires looping through every message and check if it has traversed a blacklisted path, it requires a total of mn operations; m being the number of messages in the buffer and n being the number of blacklisted paths, performed at the end of every contact instance. This compared to authenticating once per contact session, and

Table 5.3: Summary of Simulation Parameters for Source Spoofing Attack

Parameter	Value
Number of friendly nodes	25, divided into 5 clusters, arranged in a single line. The nodes in the left-most cluster functions as receiver-only.
Terrain size and type	6000m by 2000m open terrain.
Node movement model	Cluster-based with a cluster radius of 1000m; movement speed at between 0.5ms^{-1} to 2.5ms^{-1} . Stops at end of each move for between 0 to 900 seconds (0 to 15 minutes).
Node radio	Single radio; range of 250 meters (12Mbps)
Node buffer	256MB.
Node communication rate	Average 10 messages per hour from each sender node (one new message generated from an arbitrary sender node to a receiver node between 0 and 18 seconds), each message 256kB in size with a TTL of 360 minutes (6 hours).
Simulation duration	43200 simulated seconds (12 simulated hours).

Figure 5.9: Packet delivery rate against spoofed source flooding attack (25 friendly nodes).

then essentially deleting all packets from the buffer from that contact session at the end of it if authentication fails.

Another interesting point to consider is that while packet filtering works for small-scale networks (e.g. the 25 friendly node scenario), it may not be as scalable as the authentication and its filtering technique. To demonstrate this, the simulations are rerun with 100 friendly nodes.

Simulation parameters are per the previous simulation runs, except for the following. Each friendly cluster now contains 20 nodes, and simulation is performed with 4, 8, and 12 adversaries per cluster for a total of 16, 32, and 48 adversaries (recall that there are four adversary clusters and five friendly clusters). Each adversary generates an average of 30 messages per hour, again destined for the receiving cluster.

A snapshot of the terrain layout and radio range for the simulation is shown in Figure 5.10.

The results of the simulation is shown in Figure 5.11.

Figure 5.10: Terrain layout for source spoofing attacks (100 friendly nodes).

Figure 5.11: Packet delivery rate against spoofed source flooding attack (100 friendly nodes).

In such dense networks, with increased adversary activity, the delivery rate of pattern filtering suffers performance decline as the number of adversary increases. The decline is continuous for the MOFO buffer strategy, while for the FIFO strategy, pattern filtering manages to keep up when the adversary count is 16, but then declines as the number of adversaries increase.

Authenticated filtering, on the other hand, maintains the pre-attack perfor-

mance under both buffer strategies and increasing number of adversaries.

5.8 Conclusion

This chapter provides a novel protocol for protecting beacon message information, of which such protection can be utilized to prevent adversaries from launching attacks into the DTN. In certain circumstances, such protection could also be moved up and integrated into the routing protocol. In addition, the coupling of such protection with translucent sets provides the ability for a DTN node to inform other DTN nodes about its security state discreetly. This therefore allows a node (with some input from its operator) to notify other nodes within the DTN when it is captured or compromised, therefore allowing other nodes to take action including engaging in disinformation against the captured / compromised node.

Moreover, it is shown that the authentication information can be used to mitigate threats within the DTN. Flooding and spoofing attacks, which severely impacts the delivery rates in routing protocols that are not protected, are shown to be mitigated when schemes utilizing the authentication information is used.

Chapter 6

Using a TPM as Key Generating Authority for the Shamir IBS Scheme

The protocol to protect beacon message information in Chapter 5 relies on a key generating authority (KGA) for the Shamir identity-based signature (IBS) scheme to generate signing keys for the DTN nodes. Measures have to be taken to ensure that the KGA is a secure and trustworthy authority, since a compromised KGA and key generating secrets could result in fraudulent key generations.

This chapter presents another application of the TPM - this time as a key generating authority (KGA) in the Shamir identity-based asymmetric signature (IBS) scheme. It seeks to further demonstrate how TPMs can be used for alternative, unintended, and sometimes more lightweight causes that extends beyond the normal, documented, specification-outlined uses. The implementation can be used as a KGA candidate (for the portion to generate IBS signer keys) for Chapter 5 to afford the tighter security provided by using a TPM for such a purpose.

6.1 The Shamir Identity-Based Signature Scheme

Recall that there are four main operations in the Shamir IBS scheme, namely initialization, key generation, message signing, and verification. The process of generating the master key pair happens during the initialization phase, which must take place before signer keys can be generated and identity-based signing performed. This is most commonly performed at the KGA, though it is not strictly a must.

The KGA can begin to generate signer private keys after the master private key is generated. Figure 6.1 presents the initialization phase and key generation operation in detail.

Figure 6.1: Initialization and key generation in the Shamir IBS scheme.

Given that the KGA has full authority over the generation of master and signer keys, and that a compromised KGA could lead to unauthorized signer key generation, it is therefore imperative that the KGA be a secure and trusted authority.

The Trusted Platform Module thus offers a mechanism towards the above ends.

6.2 The Fundamental Mathematics

6.2.1 The Initialization Phase

The initialization phase for Shamir's IBS scheme begins with the KGA selecting two large prime numbers $p, q \in \mathbb{P}$ followed by the computation of $n = pq$ and $\phi(n) = (p - 1)(q - 1)$. The KGA then selects a random value for e such that $1 < e < \phi(n)$ and e is co-prime with $\phi(n)$.

Using e and $\phi(n)$, the KGA computes d such that $d = e^{-1} \bmod \phi(n)$. The pairs (d, n) and (e, n) are the master private and public key respectively. (e, n) is made public, together with a pre-selected one-way function f such that $f : \{0, 1\}^* \times \{0, 1\}^* \rightarrow \{0, 1\}^l$ which will be used during the signing and verifying process, and $H : \{0, 1\}^* \rightarrow \{0, 1\}^k$, a cryptographic hash function used to hash signer identities. (d, n) is stored in secured storage for later use during signature key generation.

It can be seen that the master key pair generation process is essentially that of generating an RSA key pair.

6.2.2 The Key Generation Operation

Compared to the initialization phase, the key generation operation is more straightforward. A signer wishing to obtain a signer key would submit a request containing its identity μ to the KGA. Upon satisfied that the identity in the request belongs to the entity submitting it, the KGA proceeds to generate the signer key.

The KGA begins the signer key generation by first hashing μ using the cryptographic hash function H to break the inherent RSA homomorphism (where $\alpha^d \cdot \beta^d \equiv (\alpha\beta)^d \pmod{n}$). This step is vital as it prevents multiple signers from colluding later to generate fake signer identities that possess valid signer keys.

Upon obtaining the identity digest $i_\mu = H(\mu)$, the KGA performs an RSA signing operation on it using the master private key (d, n) to get $g_\mu = i_\mu^d \bmod n$. g_μ is the signer private key and is returned to the signer, who can then start using

it to generate signatures.

6.2.2.1 Using the Signer Private Key

To generate a signature for a message m , a signer μ first computes $t = r^e \bmod n$ using a random $r \in \mathbb{Z}_n$ and the master public key (e, n) . The signer then generates $s = g_\mu \cdot r^{f(t,m)} \bmod n$ using the signer private key g_μ , the one-way function f , the values t and r , and the message m . (t, s) is the signature for message m .

6.2.2.2 Verifying a Signature

The signer's identity μ must first be hashed using H to produce i_μ . Signature verification is successful if, for a message m and signature (t, s) , the statement $s^e \equiv i_\mu \cdot t^{f(t,m)} \pmod{n}$ holds.

6.3 Why Use a TPM as KGA?

The use of a TPM as KGA in the Shamir IBS scheme has its own benefits, some of which cannot be replicated on non-TPM systems.

6.3.1 Low Overall Code Complexity

In the previous section, it was highlighted that the master key pair and signer private key generation operations are that of an RSA key pair generation and RSA signing operation respectively. These are operations that can be performed by a TPM.

A TPM can be used to generate a TPM Signing Key (which is an RSA key pair) for use as a KGA master key pair, and subsequently using that key, the TPM can perform signing operations to generate signer private keys for individual signers. As these operations are fundamental TPM operations (albeit not performed in a single step), the code complexity for implementing them are relatively low.

6.3.2 Security of the TPM Key Generation Process

Given that the KGA has authority over the generation of master and signer keys, and that a compromised KGA or leaked master private key could lead to unauthorized signer key generation, it is imperative that the KGA be a secure and trusted authority, and that the key generation operations and master private key be kept secure. In this sense, using a TPM to generate, manage, and use the master key pair would afford the signature scheme more protection than using a non-TPM approach.

Generating a TPM key entails first generating an empty key object, and then populating it with an actual key blob. Given that for the latter, the actual RSA key generation process happens within a TPM, and the output private key blob is encrypted to the key's parent key prior to leaving the TPM, the chances of the actual RSA private key being known to or intercepted by any entity other than the TPM itself remains virtually non-existent. Moreover, since the actual RSA key generation process happens in isolation within the TPM, it would also be free from any external influences.

The process of a TPM key generation, from preliminaries to the key generation itself and subsequent optional key migration¹ is shown in Figure 6.2.

6.3.3 Security from the Use of TPM Keys

The security that is afforded from the use of TPM keys can be viewed from two points. Firstly, as mentioned in the previous subsection, the private portion of a TPM key is never exposed in cleartext outside the TPM. Secondly, the conditions for loading and migrating a TPM key can be set and strictly enforced, therefore restricting the environment where the keys can be used. In order to understand such concepts, one has to first understand the way TPM keys are stored and

¹Key migration allows a TPM key to be migrated out of its created branch. If key migration is not required or the TPM key is marked as non-migratable, the migration process does not need to be run. The migration technique shown in Figure 6.2 is a straightforward migration method (direct migration), which is one of a few possible ways for key migration. This and other methods of key migration are described in [125, 126].

Figure 6.2: Generating a TPM key.

managed.

A TPM that is enabled, active, and owned would have at least two RSA keys - an Endorsement Key (EK) which identifies the TPM, and a Storage Root Key (SRK) which forms the root for transitive protection of other TPM keys. Besides those keys, a TPM can generate and load a few other types of TPM keys, among

them Binding Keys and Signing Keys. On top of that there are also Storage Keys, which are used to encrypt and protect other keys (the Storage Root Key can be thought of as a special Storage Key).

TPM keys are stored as a hierarchy of keys with the SRK at the root. It is from this hierarchical structure that protection is afforded to TPM keys. In a TPM key hierarchy, all keys except the SRK and EK are protected (encrypted) by a respective parent key. This parent key is in turn protected by its own parent key, and this transitive protection is recursive all the way back to the SRK. At the root, the SRK and EK are protected by the TPM itself; both are non-migratable, and the private portion of these keys are never exposed outside the TPM, even in encrypted form.

This means that in order to load a TPM key, one has to load its parent key first, and in order to load that parent key, its parent key is required, and so on, ultimately loading the SRK itself. Naturally, this also means that in order to generate a TPM key at a location within the hierarchy, all keys from the SRK leading to that new key location will have to be loaded. Figure 6.3 depicts this key hierarchy.

Since the private portion of a TPM key is never exposed in cleartext outside of a TPM, and extraction of such private portion while the key is loaded in a TPM is extremely impractical, it can be said that the private portion of the TPM keys remain secure from intercept.

In addition to the security of TPM private keys themselves, the conditions for loading (and thus use of) a TPM key can be set and strictly enforced. For example, a TPM key can be bound to a set of Platform Configuration Register (PCR) values, therefore ensuring that the key can only be loaded into the TPM if the corresponding PCR values during load-time matches that of the bound PCR values. Since PCR values represent the state of measured entities within a system, if such a PCR-bound key is used as the master key, assurance can be provided that the key is loadable only if the state of measured entities are equal to that represented by the bounding PCR values. In other words, it means that the

Figure 6.3: A TPM key hierarchy example.

measured entities, and its sequence, must have not been tampered with.

Migration of a TPM key (i.e. exporting a TPM key for use in another TPM) can also be strictly controlled, or disallowed altogether. By generating and using a non-migratable TPM Signing Key as master key for example, the KGA can ensure that the master key is loadable only into its own TPM, and that any attempt to export and use the key outside of its TPM will fail. On the other hand, generating a migratable TPM Signing Key as master key allows the key to be loaded into other TPMs, provided such TPMs are authorized to do so by the generating TPM. This provides a form of control over key migration, in that only authorized entities may load the migrated key.

6.3.4 Assurance to External Parties

From a signer's point-of-view, assurance that the KGA is a trustworthy authority is something important to have before requesting for a signer private key. Fortunately, it is possible to perform attestation of a TPM-enabled system remotely using a concept known as remote attestation. Such attestation framework could

be custom-deployed at the KGA to allow signers to know the state of the KGA. Note though that the attestation scheme is independent of the Shamir IBS scheme.

The concept of attestation relies on the TPM forming a Root of Trust for Measurement, from which a chain of trust can be built. In a typical TPM-enabled system, this chain extends from the Core Root of Trust for Measurement (CRTM) up to the BIOS, and if implemented, into the bootloader, operating system and finally the executable application codes.

Consequently at any one time, a system's state can be established by its set of PCR values, and using those values (and available measurement lists), a decision can then be made on whether or not a system is free of tampered, or undesirable yet activated components which in turn forms a basis on whether or not the system can be trusted. In the case of the KGA, a signer would then be able to decide whether or not to proceed with signer key generation.

6.4 Using a TPM as KGA for Shamir's IBS Scheme

6.4.1 Programming for the TPM and Key Generation Primer

To control a TPM and make it work, one has to first know how to program for a TPM. Programming for a TPM can be done using a TCG Software Stack (TSS) - sets of codes that provide an interface for accessing and executing TPM functions (through the operating system's TPM driver). The Trusted Computing Group (TCG) defines a specification for this stack, called the TSS Specification [127] (references to functions, objects, and flags will be to those defined within this specification, unless otherwise stated).

One relatively popular implementation of this TSS is called TrouSerS [128]. TrouSerS is an open source implementation of the TSS first developed for Linux, but was recently ported over to Windows [129]. It is available as prebuilt packages for popular Linux distributions like Ubuntu, Fedora, openSUSE, and Debian.

Programming for a TPM is relatively straightforward once one is familiar with

the TSS API functions and the sequence in which to call them. Some good tutorials for learning to program for a TPM can be found at [130, 131].

TPM key objects can be created by calling the **Tspi_Context_CreateObject** function and passing into it the **TSS_OBJECT_TYPE_RSAKEY** type definition as well as a set of initialization flags that specifies the type (e.g. Signing, Binding, Storage, etc.) and characteristics (e.g. key length, migratability, etc.) of the key. For example to generate a 2048-bit TPM Signing Key, the initialization flags **TSS_KEY_SIZE_2048** and **TSS_KEY_TYPE_SIGNING** are used.

Once a TPM key object is created and returned, it remains merely a container and has to be populated by an encrypted TPM key blob before it can be of any real use. This can be done by either loading it with a pre-existing encrypted TPM key blob (using **Tspi_SetAttribData**), or by generating a new encrypted TPM key blob (using **Tspi_Key_CreateKey**). When that is done, the key object containing the (encrypted) key blob can be loaded into the TPM for use where the TPM will automatically decrypt the key blob using its parent key (provided that is already loaded).

6.4.2 Generating the Master Key Pair using a TPM

In this example, a 2048-bit, non-migratable TPM Signing Key protected by the SRK that can be used as a KGA master key will be generated. The full code snippet for generating this key is presented in Figure 6.4.²

The first step in generating this TPM key, or any TPM key as a matter of fact, is to define the initialization flags to be used. Here, for a 2048-bit non-migratable signing key usable without authorization, the flags **TSS_KEY_TYPE_SIGNING**, **TSS_KEY_SIZE_2048**, **TSS_KEY_NO_AUTHORIZATION**, and **TSS_KEY_NOT_MIGRATABLE** are used.

With the flags defined and the **TSS_OBJECT_TYPE_RSAKEY** type definition, the function **Tspi_Context_CreateObject** can be called, which generates

²The generated master key pair *hKey* is protected by the SRK.

```

TSS_HCONTEXT hContext;

TSS_HKEY hSRK, hKey;
TSS_HPOLICY hSRKp;
TSS_UUID hSRKu = TSS_UUID_SRK;

TSS_FLAG hKeyf = TSS_KEY_TYPE_SIGNING | TSS_KEY_SIZE_2048 |
    TSS_KEY_NO_AUTHORIZATION | TSS_KEY_NOT_MIGRATABLE;

/* Replace with actual SRK secret */
BYTE hSRKs[] = TSS_WELL_KNOWN_SECRET;

/* Create and connect to context */
Tspi_Context_Create(&hContext);
Tspi_Context_Connect(hContext, NULL);

/* Load SRK and set SRK secret */
Tspi_Context_LoadKeyByUUID(hContext, TSS_PS_TYPE_SYSTEM, hSRKu, &hSRK);
Tspi_GetPolicyObject(hSRK, TSS_POLICY_USAGE, &hSRKp);
Tspi_Policy_SetSecret(hSRKp, TSS_SECRET_MODE_SHA1, sizeof(hSRKs), hSRKs);

/* Create and load signing key */
Tspi_Context_CreateObject(hContext, TSS_OBJECT_TYPE_RSAKEY, hKeyf,
    &hKey);
Tspi_SetAttribUint32(hKey, TSS_TSPATTRIB_KEY_INFO,
    TSS_TSPATTRIB_KEYINFO_SIGSCHEME,
    TSS_SS_RSASSAPKCS1V15_SHA1);

Tspi_Key_CreateKey(hKey, hSRK, 0);
Tspi_Key_LoadKey(hKey, hSRK);

/* Unload key */
Tspi_Key_UnloadKey(hKey);

/* Free and close context */
Tspi_Context_FreeMemory(hContext, NULL);
Tspi_Context_Close(hContext);

```

Figure 6.4: Generate, load, and unload the KGA master key using a TPM.

an empty (no RSA key present) TPM key object.

After the TPM key object is generated, the key's signature scheme attribute can be set, which lets the TCG Software Stack (TSS) and TPM know how to perform signing. For this KGA master key, the signature will be performed using

the RSASSA-PKCS1-v1.5 scheme [132] defined in PKCS #1 with SHA-1 as the hash algorithm, and thus the key's signature scheme will be set to that using the flag ***TSS_SS_RSASSAPKCS1V15_SHA1***. The key's signature scheme attribute, set via the **Tspi_SetAttribUint32** function, must be set before calling the **Tspi_Key_CreateKey** function as no TPM key attribute (except for the key blob) can be changed or set after **Tspi_Key_CreateKey** returns successfully.

The RSA key within the TPM key object can then be created using the **Tspi_Key_CreateKey** function. This function requires the key and its parent key object handle (in this case the SRK) to be specified, as well as an optional handle to a Platform Configuration Register (PCR) composite object (which would bind the key to the PCR values within this object). For this example, a null value is passed instead; the key would not be bound to any PCR values.

Upon completion of **Tspi_Key_CreateKey**, one would have an encrypted 2048-bit, non-migratable TPM Signing Key blob protected by the SRK that uses the RSASSA-PKCS1-v1.5 scheme with SHA-1 as the hash algorithm, ready to be used as a KGA master key. Unlike the original Shamir IBS scheme, one cannot select the public exponent e for this KGA master key, which, if in compliance with the ISO/IEC Standard 11889, shall be set to $2^{16} + 1$. The use of such a value though should not pose a security issue to the IBS scheme.

This encrypted key blob is still located outside the TPM, and one would have to load it back into the TPM before use.

If this key pair is to be used immediately, the **Tspi_Key_LoadKey** function can be called to load the encrypted key blob into the TPM (generated key blobs do not get loaded automatically into the TPM). Else if the key is to be stored for use later, the **Tspi_GetAttribData** function can be called to extract the encrypted key blob from the key object. Also, the public portion of the TPM Signing Key can be distributed to verifiers for use as the master public key. This is done again using the **Tspi_GetAttribData** function.

Although this example does not generate a migratable TPM key, it is possible to generate one using roughly the same way plus migratable key related steps.

With key migration, it means that a KGA can securely delegate the task of signer key generation to other TPM-enabled systems, knowing that only the recipient TPM can load and use the key migration blob generated.

6.4.3 Generating the Signer Private Key using a TPM

With the KGA master key generated, the KGA can now proceed to generate signer keys for individual signers. During this process, the KGA accepts a signer key generation request and the identity (μ) of the signer. The code snippet for generating a signer key is given in Figure 6.5.³

```
TSS_HCONTEXT hContext;
TSS_HKEY hKey;
TSS_HHASH hHash;

BYTE *bId;
UINT32 bIdLength;

BYTE *bIdSigKey;
UINT32 bIdSigKeyLength;

...

Tspi_Context_CreateObject(hContext, TSS_OBJECT_TYPE_HASH,
    TSS_HASH_SHA1, &hHash);

Tspi_Hash_UpdateHashValue(hHash, bIdLength, bId);

Tspi_Hash_Sign(hHash, hKey, &bIdSigKeyLength, &bIdSigKey);
```

Figure 6.5: Generating a signer key *bIdSigKey* for the identity object *bId*.

Using the previously created KGA master key as an example, the first step to generate a signer key (g_μ) would be to create a hash object. This is done using the **Tspi_Context_CreateObject** function with the **TSS_OBJECT_TYPE_HASH** type definition and **TSS_HASH_SHA1** initialization flag (this initialization flag

³The code snippet is used to generate a signer key *bIdSigKey* for the identity object *bId*, using the KGA master key pair *hKey*. *hKey* would have to be loaded into the TPM before the **Tspi_Hash_Sign** function is called.

sets the hash object's algorithm attribute to SHA-1). Creating the object with ***TSS_HASH_SHA1*** instead of ***TSS_HASH_OTHER*** allows the use of the **Tspi_Hash_Sign** function later to sign the hash object using the TPM-generated KGA master key.

After the hash object is generated, it has to be updated with the identity of the signer. This is done using the **Tspi_Hash_UpdateHashValue** function, passing to it μ . When this function returns successfully, the hash value in the hash object will be $\text{SHA-1}(\mu)$. Now containing the hashed signer identity, the hash object is ready for signing.

To sign the hash object, one loads the KGA master key blob into the TPM and calls the **Tspi_Hash_Sign** function to sign the hash using it. With the KGA master key's signature scheme (RSASSA-PKCS1-v1_5 scheme with SHA-1 as the hash algorithm) and the hash object's algorithm (SHA-1), what happens during this function call is that the TPM will be handed the hash value (the *digest*, $\text{SHA-1}(\mu)$), which it then uses to construct a DER-encoded DigestInfo type [132]. The TPM then applies the necessary PKCS #1 signature padding (the process together with the DigestInfo construction is known as EMSA-PKCS1-v1_5), and performs the RSA signing operation using the KGA master key. The use of EMSA-PKCS1-v1_5 represents a minor addition to the original signer key generation process where one just performs RSA signing on the identity digest (i.e. $\text{SHA-1}(\mu)$).

The signature generated and returned by the TPM (which can be represented as

$\text{RSA}_{\text{SIGN}}(\text{EMSA-PKCS1-v1.5}(\text{SHA-1}(\mu)))$) would be the signer key g_μ , and can be used immediately once returned to the signer; message signature generation and verification do not require the use of a TPM. Nevertheless, it is important to note that during verification, an identity would have to be hashed and encoded to form $\text{EMSA-PKCS1-v1.5}(\text{SHA-1}(\mu))$.

6.4.4 Using the Signer Private Key

Once the signer receives its signer private key from the KGA, it could proceed to use it to generate signatures. The signature generation (and verification) process does not require the use of a TPM.

For a signature to be generated, the signer (μ) would need to perform the signing operation of the Shamir IBS scheme. For a message m , this entails generating the value $t = r^e \bmod n$ using a random r and the master public key (e, n) , and generating $s = g_\mu \cdot r^{f(t,m)}$ using the signer private key g , r , the one-way function f , t , together with the message m . (t, s) is then the signature for message m .

The signature generation process (and verification process later) can be realized using readily available arbitrary-precision arithmetic libraries, like the GNU Multiple Precision Arithmetic Library [123].

6.4.5 Verifying a Signature Generated using a Signer Private Key

Signature verification is performed as per the Shamir IBS scheme (test if $s^e = i_\mu \cdot t^{f(t,m)} \bmod n$ holds). However due to the use of hashing (SHA-1) and encoding (EMSA-PKCS1-v1_5) in generating the signer key, the same hashing and encoding must be applied to the signer identity during verification. These can be done using readily available cryptographic libraries like OpenSSL.

To start, the signer's identity μ must first be hashed and encoded to produce $i_\mu = \text{EMSA-PKCS1-v1_5}(\text{SHA-1}(\mu))$. This hashed and encoded identity i_μ can then be used to verify the signer's signature (t, s) for a message m .

Signature verification is successful if $s^e = i_\mu \cdot t^{f(t,m)} \bmod n$.

6.5 Real-World Implementation and Performance

Since TPMs are quite ubiquitous in OEM systems nowadays [126, 133, 134], and the required libraries and TCG Software Stack are readily available especially for

the Linux platform (Trousers, OpenSSL, and GNU Multiple Precision Arithmetic Library), the entire code base for TPM key generation (for the KGA master key), TPM signing (to generate IBS signer keys), Shamir IBS signing, and signature verification can be realized in less than 1000 lines. This makes it practical for use either standalone or through integration into larger systems that would benefit from the use of IBS schemes.

Nevertheless, one should know the limitations of the TPM. A TPM is not a crypto-accelerator. It is a crypto-processor (among one of its functions), meaning it performs, not accelerates cryptographic operations. TPM operations can be relatively slow compared to modern day CPUs, though still within acceptable limits.

To understand the performance difference between a TPM and CPU when used as KGA for the Shamir IBS scheme, consider the results of a benchmark conducted on both a TPM and CPU performing KGA-related operations as tabulated in Table 6.1.

Table 6.1: Performance Benchmark between TPM and CPU

Operation	TPM ^a	CPU ^b
Shamir IBS master key generation ^c (<i>seconds</i> , average after 100 rounds)	53.778840	0.143150
Shamir IBS signer key generation ^d (<i>seconds</i> , average after 500 rounds)	0.930326	0.007717

^aBroadcom TPM 1.2, accessed using Trousers 0.3.7-2ubuntu1.

^bIntel Core 2 Duo E6700, running at 2.66GHz. Demonstrator utilizes GMP for large number processing.

^cRSA key pair generation.

^dRSA sign operation.

One can immediately see that a TPM is about 350 times slower than a CPU when generating a master key pair, and about 120 times slower than a CPU when generating signer keys, though in practical terms it is not that bad (about a minute to generate a master key, and one second to generate a signer key), considering master keys are rarely generated, and signer private keys are generated once per signer for use multiple times.

Even though slower than a CPU, a TPM provides better security assurances compared to a CPU, which is the main factor for using a TPM in the first place. By using a TPM as KGA, one can for example be better assured that the master key generation process is secure from external influence given that some degree of tamper resistivity is usually present in a TPM implementation. Subsequently if the key is non-migratable, one can also be better assured that the signer key generation process can only happen at the KGA, since the private portion of a TPM key, which the KGA master private key is, will never leave the TPM in clear form.

6.6 Conclusion

A Trusted Platform Module is an interesting chip (or depending on how it is implemented, a module). It can be found in many systems, and offers the benefits of a secure crypto-processor, yet is often underutilized or not utilized at all in spite of security being a major cause of concern these days.

This chapter presents one way of using a TPM by using it as (literally) a Key Generating Authority in a Shamir Identity-Based Signature Scheme implementation. It is hoped that many more ways of using this little module will be found, and it could just be that this currently underutilized module would gain more importance as more uses for it come to light.

This page is intentionally left blank.

Chapter 7

Future Works and Conclusion

7.1 Future Works

7.1.1 On Hindering Forensics against Information

In Chapter 3, the encryption of incriminating data is tied to the platform configuration. It means that if the platform is damaged, or is tampered irreversibly, then the data could no longer be decrypted. It would be interesting to work on allowing for the data to be migrated when necessary, while still maintaining exclusivity to the data owners. This would mean attempting to replace the concept of data sealing which is used now, with data binding instead (for key migration purposes, since sealing keys cannot be migrated).

Finally taking a step further, the system could be made to handle multiple users. Currently, it only supports one user.

7.1.2 On Hindering Forensics in Mobile Systems

Although preliminary evaluation demonstrates the feasibility of running the Android operating system in RAM to hinder forensics, it would be good to further test the feasibility of positioning script-based triggers at other trigger points so to find out where script-based triggers are effective. With more knowledge of such locations, further customization can be provided to strengthen the anti-forensic

capability of the operating system.

At the same time, works could also be done on the applications, which are ultimately the entities of interest (as described early in the section). This can be done by designing applications to resist forensics, perhaps via techniques like securely hiding the functions of interest, use of off-site binary segments, or crowd-based cryptographic hiding (e.g. using questionable encryption [135]).

7.1.3 On Threat Mitigation in Disruption Tolerant Networks

Though from a theoretical point-of-view, the protocol works as expected and is implementable practically where it could be utilized to better filter and discard unwanted packets, it would be interesting to find out the performance of the protocol in other real-world scenarios under more varied conditions.

In addition, some aspects of the protocol involves operator intervention (i.e. deleting of translucent set secrets), and might be more effective if such processes within the protocol can be automated in order to minimize the risk of human error, while increasing the level of plausible deniability allowed by the protocol.

7.2 Conclusion

This dissertation presented novel ideas on the design and analysis of techniques to provide security while being minimally conspicuous. The author has identified areas in which such techniques can be used, and evaluated the performance of the techniques when used. The contributions addressing each of these segments are as follows:

7.2.1 Hindering Forensics against Information

A system designed to hinder forensics against information was presented in Chapter 3. The system

- Is designed from the perspective of a criminal, with the idea being that a criminal would more likely find itself in a situation where forensics retrieval of information could result in severe consequences.
- Utilizes a Trusted Platform Module and Integrity Measurement Architecture in its construction.
- Provides information confidentiality capability to conceal incriminating information, and plausible deniability to reduce (and in best case eliminate) suspicion of the existence of the incriminating information.
- Is shown that if operated properly, the system (in conjunction with the user) should reduce the suspicion that incriminating information exists.

Although designed from a criminal's point-of-view, ideas are presented in the chapter on how such a system could be used instead for good and legitimate purposes.

7.2.2 Hindering Forensics in Mobile Systems

In Chapter 4, a method of deploying Android to run in RAM to hinder forensics in mobile devices was presented. The method entails having the operating system copy itself from persistent storage to RAM and continue execution from there, then deleting the sensitive components and certain other entities from persistent storage, including parts of the scripts to trigger such deletion. This therefore eliminates any evidence of the sensitive components ever being in persistent storage, and evidence of scripts designed to engage in such deletion.

It has been shown how such a method would be disadvantageous to first responders in particular and forensics in general, as it is almost unlikely for first responders to know if an acquired device is actually engaging in measures to remove sensitive components.

Evaluation of an x86-based Android distribution running in RAM on a laptop has shown that the method worked as intended for the anti-forensic measures

proposed.

7.2.3 Threat Mitigation in Disruption-Tolerant Networks

In Chapter 5, a protocol to authenticate beacon message information that could also be used for authenticating routing data using the Shamir identity-based signature (IBS) scheme was presented. The signatures contain a hidden, translucent code used in conjunction with the signer to allow the verifier to know if the signer is compromised.

On top of that, the chapter shows how the authentication information, once obtained by the verifier, can be used in schemes to secure the DTN from attacks. Two attacks have been presented - flooding and spoofing, which have been shown to severely impact delivery rates in routing protocols that are not protected. Affording protection to the protocols resulted in successful mitigation of the impact of the attacks.

The protection schemes outlined are inconspicuous, in that an adversary launching an attack would not know at the time of attack whether or not its attack is successful. In particular, the schemes react as per normal under an attack, such that an attacker would assume that the attack is going well. Only after contact with the attacker ends that measures are executed to manage the consequence of the attack. It has been shown that performance is not affected by not reacting directly against the attacker.

7.2.4 TPM as KGA for the Shamir IBS Scheme

Chapter 6 presents an implementation of a key generating authority (KGA) for the Shamir IBS scheme using a TPM. It demonstrates how TPMs can be used for alternative, and sometimes more lightweight causes that extends beyond the documented, specification-outlined uses. In addition, since the protocol for beacon message information authentication in Chapter 5 requires a KGA, the implementation can be used there as well.

Author's Publications

Journals

- W. Goh, P. C. Leong, and C. K. Yeo, "A Plausibly-Deniable, Practical Trusted Platform Module Based Anti-Forensics Client-Server System," *IEEE Journal on Selected Areas in Communications*, vol. 29, no. 7, pp. 1377-1391, Aug. 2011.
- W. Goh and C. K. Yeo, "Teaching an Old TPM New Tricks: Repurposing for Identity-Based Signatures," *IEEE Security & Privacy Magazine*, vol. 11, no. 5, pp. 28-35, Sep.-Oct. 2013.

Conferences

- W. Goh and C. K. Yeo, "Anonymity-Preserving Identity-Based Multisignature Scheme with Provision for Origin Self-Revelation," accepted for *2013 IEEE Global Telecommunications Conference (GLOBECOM 2013)*.
- W. Goh and C. K. Yeo, "Threat Mitigation in Tactical-Level Disruption Tolerant Networks," in *2012 IEEE Global Telecommunications Conference (GLOBECOM 2012)*, Anaheim, CA, Dec. 2012, pp. 997-1003.
- F. C. Lee, W. Goh, and C. K. Yeo, "A Queuing Mechanism to Alleviate Flooding Attacks in Probabilistic Delay Tolerant Networks," in *2010 Sixth Advanced International Conference on Telecommunications (AICT 2010)*, Barcelona, Spain, May 2010, pp. 329-334.

- W. Goh, P. C. Leong, and C. K. Yeo, “A Trusted Platform Module Based Anti-Forensics System,” in *IFIP International Conference on Network and Service Security, 2009 (N2S ‘09)*, Paris, France, Jun. 2009, pp. 1-5.
- W. Goh and C. K. Yeo, “Cryptanalyzing the Efficient Identity-Based RSA and GQ Multisignature Schemes,” submitted to *2014 IEEE International Conference on Communications (ICC 2014)*.

Bibliography

- [1] S. Goldwasser and S. Micali, “Probabilistic encryption & how to play mental poker keeping secret all partial information,” in *Proceedings of the 14th Annual ACM Symposium on Theory of Computing (STOC ‘82)*, San Francisco, CA, USA, 1982, pp. 365-377.
- [2] Y. Watanabe, J. Shikata, and H. Imai, “Equivalence between Semantic Security and Indistinguishability against Chosen Ciphertext Attacks,” in *Proceedings of the 6th International Workshop on Theory and Practice in Public Key Cryptography: Public Key Cryptography (PKC 2003)*, Miami, FL, USA, Jan. 2003, ser. Lecture Notes in Computer Science, vol. 2567, Y. Desmedt, Ed., Berlin / Heidelberg, Germany: Springer-Verlag, 2002, pp. 71-84.
- [3] B. Grundy and J. Garris, “One Policy Approach Regarding Digital Evidence Acquisition and Analysis,” National Aeronautics and Space Administration, Aug. 2007. [Online]. Available: ftp://ftp.hq.nasa.gov/pub/ig/ccd/procedures/1_Approach_Digital_Evid.pdf. [Accessed: Mar. 20, 2010]
- [4] *Searching and Seizing Computers and Obtaining Electronic Evidence in Criminal Investigations*, 3rd ed., Office of Legal Education Executive Office for United States Attorneys, 2009. [Online]. Available: <http://www.justice.gov/criminal/cybercrime/docs/ssmanual2009.pdf>. [Accessed: Apr. 17, 2010].

- [5] *Electronic Crime Scene Investigation: A Guide for First Responders*, 2nd ed., National Institute of Justice, 2008. [Online]. Available: <http://www.nij.gov/pubs-sum/219941.htm>. [Accessed: Jan. 30, 2011].
- [6] E. Casey and R. Dunne, *Digital evidence and computer crime: forensic science, computers and the Internet*. Amsterdam; Boston: Elsevier Academic Press, 2004.
- [7] *Forensic Examination of Digital Evidence: A Guide for Law Enforcement*, National Institute of Justice, 2004. [Online]. Available: <http://www.nij.gov/pubs-sum/199408.htm>. [Accessed: Jan. 30, 2011].
- [8] M. Reith, C. Carr, and G. H. Gunsch, "An Examination of Digital Forensic Models," *International Journal of Digital Evidence*, vol. 1, no. 3, pp. 1-12, 2002.
- [9] S. Ciardhuáin, "An Extended Model of Cybercrime Investigations," *International Journal of Digital Evidence*, vol. 3, no. 1, pp. 1-22, 2004.
- [10] S. Mocas, "Building theoretical underpinnings for digital forensics research," *Digital Investigation*, vol. 1, no. 1, pp. 61-68, Feb. 2004.
- [11] K. Kent, S. Chevalier, T. Grance, and H. Dang "Guide to Integrating Forensic Techniques into Incident Response," National Institute of Standards and Technology, NIST SP 800-86, Aug. 2006. [Online]. Available: <http://csrc.nist.gov/publications/nistpubs/800-86/SP800-86.pdf>. [Accessed: Aug. 23, 2009].
- [12] N. L. Beebe and J. G. Clark, "A hierarchical, objectives-based framework for the digital investigations process," *Digital Investigation*, vol. 2, no. 2, pp. 147-167, Jun. 2005.
- [13] M. M. Pollitt, "An Ad Hoc Review of Digital Forensic Models," in *Proceedings of the 2nd International Workshop on Systematic Approaches to Digital Forensic Engineering (SADFE '07)*, Washington, DC, USA, 2007, pp. 43-54.

- [14] B. D. Carrier and E. H. Spafford, "Categories of digital investigation analysis techniques based on the computer history model," *Digital Investigation*, vol. 3, Supplement, no. 0, pp. 121-130, Sep. 2006.
- [15] S. L. Garfinkel, "Digital forensics research: The next 10 years," *Digital Investigation*, vol. 7, Supplement, no. 0, pp. S64-S73, Aug. 2010.
- [16] B. D. Carrier and J. Grand, "A hardware-based memory acquisition procedure for digital investigations," *Digital Investigation*, vol. 1, no. 1, pp. 50-60, Feb. 2004.
- [17] B. Schatz, "BodySnatcher: Towards reliable volatile memory acquisition by software," *Digital Investigation*, vol. 4, Supplement, no. 0, pp. 126-134, Sep. 2007.
- [18] J. A. Halderman, S. D. Schoen, N. Heninger, W. Clarkson, W. Paul, J. A. Calandrino, A. J. Feldman, J. Appelbaum, and E. W. Felten, "Lest we remember: cold-boot attacks on encryption keys," *Commun. ACM*, vol. 52, no. 5, pp. 91-98, May 2009.
- [19] N. Mikus, "An Analysis of Disc Carving Techniques," Master's thesis, Dept of Computer Science, Naval Postgraduate School, Monterey, CA, USA, 2005.
- [20] M. Chen, N. Zheng, M. Xu, Y. Lou, and X. Wang, "Validation Algorithms Based on Content Characters and Internal Structure: The PDF File Carving Method," in *2008 International Symposium on Information Science and Engineering (ISISE '08)*, 2008, vol. 1, pp. 168-172.
- [21] D.-G. Park, S.-J. Park, J.-C. Lee, S.-Y. No, and S.-Y. Shin, "A File Carving Algorithm for Digital Forensics," in *International Conference on Computational Science and Its Applications (ICCSA 2009)*, Seoul, Korea, Jul. 2009, ser. Lecture Notes in Computer Science, vol. 5592, O. Gervasi, D. Taniar, B. Murgante, A. Laganà, Y. Mun, and M. Gavrilova, Eds., Berlin / Heidelberg, Germany: Springer-Verlag, 2009, pp. 615-626.

- [22] S. L. Garfinkel, A. Parker-Wood, D. Huynh, and J. Migletz, “An Automated Solution to the Multiuser Carved Data Ascription Problem,” *IEEE Transactions on Information Forensics and Security*, vol. 5, no. 4, pp. 868-882, Dec. 2010.
- [23] J.-S. Kim, D.-G. Kim, and B.-N. Noh, “A fuzzy logic based expert system as a network forensics,” in *Proceedings: 2004 IEEE International Conference on Fuzzy Systems*, 2004, vol. 2, pp. 879-884.
- [24] M. Ponc, P. Giura, H. Brönnimann, and J. Wein, “Highly efficient techniques for network forensics,” in *Proceedings of the 14th ACM Conference on Computer and Communications Security*, Alexandria, VA, USA, Oct. 2007, pp. 150-160.
- [25] “Federal Rules of Evidence.” [Online]. Available: <http://www.law.cornell.edu/rules/fre/>. [Accessed: Jan. 7, 2012].
- [26] L. V. Romano, “Electronic Evidence and the Federal Rules,” *Loyola of Los Angeles Law Review*, vol. 38, no. 4, pp. 1745-1801, Jul. 2005.
- [27] J. D. Frieden and L. M. Murray, “The admissibility of electronic evidence under the federal rules of evidence,” *Rich. JL & Tech.*, vol. 17, no. 2, pp. 5-16, 2011.
- [28] M. K. Rogers, “Anti-Forensics,” presented at Lockheed Martin, San Diego, CA, Sep. 2005. [Presentation Slides]. Available: <http://www.cyberforensics.purdue.edu/presentations.aspx>. [Accessed: May. 23, 2011].
- [29] C. S. J. Peron and M. Legary, “Digital anti-forensics: emerging trends in data transformation techniques,” in *Proceedings of 2005 E-Crime and Computer Evidence Conference*, 2005.

- [30] R. Harris, “Arriving at an anti-forensics consensus: Examining how to define and control the anti-forensics problem,” *Digital Investigation*, vol. 3, Supplement, no. 0, pp. 44-49, Sep. 2006.
- [31] R. Chandramouli, M. Kharrazi, and N. Memon, “Image Steganography and Steganalysis: Concepts and Practice,” in *Digital Watermarking, Second International Workshop, IWDW 2003, Revised Papers*, Seoul, Korea, Oct. 2003, ser. Lecture Notes in Computer Science, vol. 2939, T. Kalker, I. Cox, and Y. Ro, Eds., Berlin / Heidelberg, Germany: Springer-Verlag, 2004, pp. 204-211.
- [32] N. Provos and P. Honeyman, “Hide and Seek: An Introduction to Steganography,” *IEEE Security & Privacy*, vol. 1, no. 3, pp. 32-44, May 2003.
- [33] N. Cvejic, “Algorithms for audio watermarking and steganography,” Academic Dissertation, Department of Electrical and Information Engineering, Information Processing Laboratory, University of Oulu, Oulu, Finland, 2004.
- [34] V. M. Potdar, S. Han, and E. Chang, “A survey of digital image watermarking techniques,” in *3rd IEEE International Conference on Industrial Informatics (INDIN '05)*, 2005, pp. 709-716.
- [35] K. Bennett, “Linguistic Steganography: Survey, Analysis, and Robustness Concerns for Hiding Information in Text,” Center for Education and Research in Information Assurance and Security, Purdue University, West Lafayette, IN, CERIAS Tech Report 2004-13, 2004. [Online]. Available: https://www.cerias.purdue.edu/assets/pdf/bibtex_archive/2004-13.pdf. [Accessed: Jul. 2, 2010].
- [36] T. Handel and M. Sandford, “Hiding data in the OSI network model,” in *Proceedings of the First International Workshop on Information Hiding*, Cambridge, UK, May 1996, ser. Lecture Notes in Computer Science, vol. 1174, R. Anderson, Ed., Berlin / Heidelberg, Germany: Springer-Verlag, 1996, pp. 23-38.

- [37] G. J. Simmons, "Results concerning the bandwidth of subliminal channels," *IEEE Journal on Selected Areas in Communications*, vol. 16, no. 4, pp. 463-473, May 1998.
- [38] K. Ahsan and D. Kundur, "Practical Data Hiding in TCP/IP," in *Proceedings of the 4th ACM Workshop on Multimedia & Security*, Juan Les Pins, France, 2002.
- [39] G. J. Simmons, "The Prisoners' Problem and the Subliminal Channel," in *Advances in Cryptology, Proceedings of CRYPTO '83*, Santa Barbara, California, USA, Aug. 1983, David Chaum, Ed., New York, USA: Plenum Press, 1984, pp. 51-67.
- [40] R. Anderson, S. Vaudenay, B. Preneel, and K. Nyberg, "The Newton channel," in *Proceedings of the First International Workshop on Information Hiding*, Cambridge, UK, May 1996, ser. Lecture Notes in Computer Science, vol. 1174, R. Anderson, Ed., Berlin / Heidelberg, Germany: Springer-Verlag, 1996, pp. 151-156.
- [41] S. Zander, G. Armitage, and P. Branch, "A survey of covert channels and countermeasures in computer network protocols," *IEEE Communications Surveys & Tutorials*, vol. 9, no. 3, pp. 44-57, quarter 2007.
- [42] H. Berghel and N. Brajkovska, "Wading into alternate data streams," *Commun. ACM*, vol. 47, no. 4, pp. 21-27, Apr. 2004.
- [43] E. Casey, "Network traffic as a source of evidence: tool strengths, weaknesses, and future needs," *Digital Investigation*, vol. 1, no. 1, pp. 28-43, Feb. 2004.
- [44] B. J. Nikkel, "Generalizing sources of live network evidence," *Digital Investigation*, vol. 2, no. 3, pp. 193-200, Sep. 2005.
- [45] "Android." [Online]. Available: <http://www.android.com/>. [Accessed: Oct. 8, 2012].

- [46] “Apple - iOS 6” [Online]. Available: <http://www.apple.com/ios/>. [Accessed: Oct. 8, 2012].
- [47] “BlackBerry - BlackBerry Device Software - US.” [Online]. Available: <http://us.blackberry.com/software/smartphones.html>. [Accessed: Oct. 8, 2012].
- [48] “Windows Phone (United States).” [Online]. Available: <http://www.windowsphone.com/en-us>. [Accessed: Oct. 8, 2012].
- [49] “Android Technical Information — Android Open Source.” [Online]. Available: <http://source.android.com/tech/index.html>. [Accessed: Jul. 9, 2012].
- [50] “Android Developers.” [Online]. Available: <http://developer.android.com/about/index.html>. [Accessed: Jul. 9, 2012].
- [51] A. Hoog, *Android Forensics: Investigation, Analysis and Mobile Security for Google Android*. Elsevier, 2011.
- [52] J. Lessard and G. Kessler, “Android Forensics: Simplifying Cell Phone Examinations,” *Small Scale Digital Device Forensics Journal*, vol. 4, no. 1, Oct. 2010.
- [53] R. Andriatsimandefitra, S. Geller, and V.V.T. Tong, “Designing information flow policies for Android’s operating system,” in *2012 IEEE International Conference on Communications (ICC 2012)*, Ottawa, ON, Canada, 2012, pp. 976-981.
- [54] S. Fiorillo, “Theory and practice of flash memory mobile forensics,” in *Proceedings of the 7th Australian Digital Forensics Conference*, Perth, WA, Australia, 2009, pp. 52-84.
- [55] W. Jansen and R. Ayers, “Guidelines on cell phone forensics,” National Institute of Standards and Technology, NIST SP SP 800-101,

- May 2007. [Online]. Available: <http://csrc.nist.gov/publications/nistpubs/800-101/SP800-101.pdf>. [Accessed: Jan. 16, 2011].
- [56] “TCG Architecture Overview, Version 1.4,” *Trusted Computing Group, Incorporated*, Aug. 2007. [Online]. Available: http://www.trustedcomputinggroup.org/resources/tcg_architecture_overview_version_14. [Accessed: Mar. 3, 2010].
- [57] *Information technology – Trusted Platform Module – Part 1: Overview*. ISO/IEC Standard 11889-1:2009.
- [58] *Information technology – Trusted Platform Module – Part 2: Design principles*. ISO/IEC Standard 11889-2:2009.
- [59] *Information technology – Trusted Platform Module – Part 3: Structures*. ISO/IEC Standard 11889-3:2009.
- [60] *Information technology – Trusted Platform Module – Part 4: Commands*. ISO/IEC Standard 11889-4:2009.
- [61] “Understand the TPM Endorsement Key,” *Microsoft TechNet*. [Online]. Available: <http://technet.microsoft.com/en-us/library/cc770443.aspx>. [Accessed: Aug. 14, 2011].
- [62] “Understand the TPM Storage Root Key,” *Microsoft TechNet*. [Online]. Available: <http://technet.microsoft.com/en-us/library/cc753560>. [Accessed: Aug. 14, 2011].
- [63] “Trusted Computing Group - Developers - Glossary,” *Trusted Computing Group*. [Online]. Available: <http://www.trustedcomputinggroup.org/developers/glossary>. [Accessed: Nov. 14, 2011].
- [64] R. Sailer, X. Zhang, T. Jaeger, and L. van Doorn, “Design and implementation of a TCG-based integrity measurement architecture,” in *Proceedings of*

the 13th conference on USENIX Security Symposium, Berkeley, CA, USA, 2004, vol. 13, pp. 16-16.

- [65] K. Fall, "A delay-tolerant network architecture for challenged internets," in *Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM '03)*, Karlsruhe, Germany, Aug. 2003, pp. 27-34.
- [66] S. Burleigh, A. Hooke, L. Torgerson, K. Fall, V. Cerf, B. Durst, K. Scott, and H. Weiss, "Delay-tolerant networking: an approach to interplanetary Internet," *IEEE Communications Magazine*, vol. 41, no. 6, pp. 128-136, Jun. 2003.
- [67] V. Cerf, S. Burleigh, A. Hooke, L. Torgerson, R. Durst, K. Scott, K. Fall, and H. Weiss, "Delay-Tolerant Networking Architecture." [Online]. Available: <http://tools.ietf.org/html/rfc4838>. [Accessed: Jan. 16, 2011].
- [68] K. Scott and S. Burleigh, "Bundle Protocol Specification." [Online]. Available: <http://tools.ietf.org/html/rfc5050>. [Accessed: Jan. 16, 2011].
- [69] T. Spyropoulos, K. Psounis, and C. S. Raghavendra, "Efficient Routing in Intermittently Connected Mobile Networks: The Single-Copy Case," *IEEE/ACM Transactions on Networking*, vol. 16, no. 1, pp. 63-76, Feb. 2008.
- [70] S. Jain, K. Fall, and R. Patra, "Routing in a delay tolerant network," *SIGCOMM Comput. Commun. Rev.*, vol. 34, no. 4, pp. 145-158, Aug. 2004.
- [71] E. P. C. Jones, L. Li, and P. A. S. Ward, "Practical routing in delay-tolerant networks," in *Proceedings of the 2005 ACM SIGCOMM workshop on Delay-tolerant networking*, Philadelphia, PA, USA, Aug. 2005, pp. 237-243.
- [72] T. Spyropoulos, K. Psounis, and C. S. Raghavendra, "Single-copy routing in intermittently connected mobile networks," in *2004 First Annual IEEE*

- Communications Society Conference on Sensor and Ad Hoc Communications and Networks, 2004 (SECON 2004)*, Santa Clara, CA, USA, Oct. 2004, pp. 235-244.
- [73] A. Balasubramanian, B. Levine, and A. Venkataramani, "DTN routing as a resource allocation problem," *SIGCOMM Comput. Commun. Rev.*, vol. 37, no. 4, pp. 373-384, Aug. 2007.
 - [74] T. Spyropoulos, K. Psounis, and C. S. Raghavendra, "Efficient Routing in Intermittently Connected Mobile Networks: The Multiple-Copy Case," *IEEE/ACM Transactions on Networking*, vol. 16, no. 1, pp. 77-90, Feb. 2008.
 - [75] A. Vahdat and D. Becker, "Epidemic Routing for Partially-Connected Ad Hoc Networks," Duke University, Durham, NC, USA, Technical Report CS-2000-06, 2006.
 - [76] J. Burgess, B. Gallagher, D. Jensen, and B. N. Levine, "MaxProp: Routing for Vehicle-Based Disruption-Tolerant Networks," in *Proceedings IEEE INFOCOM 2006*, Apr. 2006, pp. 1-11.
 - [77] B. Burns, O. Brock, and B. N. Levine, "MV routing and capacity building in disruption tolerant networks," in *Proceedings IEEE INFOCOM 2005*, Mar. 2005, vol. 1, pp. 398-408 vol. 1.
 - [78] A. Lindgren, A. Doria, and O. Scheln, "Probabilistic Routing in Intermittently Connected Networks," in *Service Assurance with Partial and Intermittent Resources*, Fortaleza, Brazil, ser. Lecture Notes in Computer Science, vol. 3126, P. Dini, P. Lorenz, and J. de Souza, Eds., Berlin / Heidelberg, Germany: Springer-Verlag, 2004, pp. 239-254.
 - [79] T. Spyropoulos, K. Psounis, and C. S. Raghavendra, "Spray and wait: an efficient routing scheme for intermittently connected mobile networks," in

Proceedings of the 2005 ACM SIGCOMM Workshop on Delay-Tolerant Networking, Philadelphia, PA, USA, Aug. 2005, pp. 252-259.

- [80] S. Farrell and V. Cahill, "Security considerations in space and delay tolerant networks," in *2nd IEEE International Conference on Space Mission Challenges for Information Technology, 2006 (SMC-IT 2006)*, Pasadena, CA, USA, Jul. 2006, pp. 29-38.
- [81] V. Paxson, "An analysis of using reflectors for distributed denial-of-service attacks," *SIGCOMM Comput. Commun. Rev.*, vol. 31, no. 3, pp. 38-47, Jul. 2001.
- [82] S. Symington, S. Farrell, H. Weiss, and P. Lovell, "Bundle Security Protocol Specification," [Online]. Available: <http://tools.ietf.org/html/rfc6257>. [Accessed: Mar. 11 2012].
- [83] A. Seth and S. Keshav, "Practical security for disconnected nodes," in *1st IEEE ICNP Workshop on Secure Network Protocols, 2005 (NPSec)*, Boston, MA, USA, Nov. 2006, pp. 31-36.
- [84] N. Asokan, K. Kostianen, P. Ginzboorg, J. Ott, and C. Luo, "Applicability of identity-based cryptography for disruption-tolerant networking," in *Proceedings of the 1st International MobiSys Workshop on Mobile Opportunistic Networking (MobiOpp '07)*, Puerto Rico, Jun. 2007, pp. 52-56.
- [85] N. Asokan, K. Kostianen, P. Ginzboorg, J. Ott, and C. Luo, "Towards Securing Disruption-Tolerant Networking," Nokia Research Center, Helsinki, Finland, Tech. Rep. NRC-TR-2007-007, Mar. 2007. [Online]. Available: <http://research.nokia.com/files/tr/NRC-TR-2007-007.pdf>. [Accessed: Feb. 15, 2009].
- [86] A. Kate, G. M. Zaverucha, and U. Hengartner, "Anonymity and security in delay tolerant networks," in *Third International Conference on Security*

and Privacy in Communications Networks and the Workshops, 2007 (SecureComm 2007), Nice, France, Sep. 2007, pp. 504-513.

- [87] R. Sakai, K. Ohgishi, and M. Kasahara, "Cryptosystems based on pairing," in *2000 Symposium on Cryptography and Information Security*, Okinawa, Japan, 2000.
- [88] H. Zhu, X. Lin, R. Lu, X. Shen, and P.-H. Ho, "BBA: An Efficient Batch Bundle Authentication Scheme for Delay Tolerant Networks," presented at the *Proceedings of the IEEE Global Telecommunications Conference, 2008 (GLOBECOM 2008)*, New Orleans, LA, USA, Nov. 2008, pp. 1-5.
- [89] H. Zhu, X. Lin, R. Lu, and X. Shen, "A secure incentive scheme for delay tolerant networks," presented at the *3rd International Conference on Communications and Networking in China, 2008 (ChinaCom 2008)*, Aug. 2008, pp. 23-28.
- [90] R. Su and Z. Cao, "An efficient anonymous authentication mechanism for delay tolerant networks," *Computers & Electrical Engineering*, vol. 36, no. 3, pp. 435-441, May 2010.
- [91] W. Itani, A. Tajeddine, A. Kayssi, and A. Chehab, "Slow but certain wins the race: authenticated bundle communication in delay tolerant networks," in *Proceedings of the 6th ACM Workshop on QoS and Security for Wireless and Mobile Networks*, Bodrum, Turkey, Oct. 2010, pp. 90-97.
- [92] S. Farrell, H. Weiss, S. Symington, and P. Lovell, "Delay-Tolerant Networking Security Overview." [Online]. Available: <http://tools.ietf.org/html/draft-irtf-dtnrg-sec-overview-06>. [Accessed: Nov. 19, 2011].
- [93] S. R. Das, E. M. Belding-Royer, and C. E. Perkins, "Ad hoc On-Demand Distance Vector (AODV) Routing." [Online]. Available: <https://tools.ietf.org/html/rfc3561>. [Accessed: Nov. 19, 2011].

- [94] P. J. <philippe.jacquet@inria.fr>, “Optimized Link State Routing Protocol (OLSR).” [Online]. Available: <https://tools.ietf.org/html/rfc3626>. [Accessed: Nov. 19, 2011].
- [95] B. Kannhavong, H. Nakayama, N. Kato, Y. Nemoto, and A. Jamalipour, “Analysis of the node isolation attack against OLSR-based mobile ad hoc networks,” in *2006 International Symposium on Computer Networks*, Istanbul, Turkey, Jun. 2006, pp. 30-35.
- [96] B. Kannhavong, H. Nakayama, and A. Jamalipour, “NIS01-2: A Collusion Attack Against OLSR-based Mobile Ad Hoc Networks,” in *Proceedings of the Global Telecommunications Conference, 2006. GLOBECOM '06*, San Francisco, CA, USA, Nov. 2006, pp. 1-5.
- [97] Y.-C. Hu, A. Perrig, and D. B. Johnson, “Wormhole attacks in wireless networks,” *IEEE Journal on Selected Areas in Communications*, vol. 24, no. 2, pp. 370-380, Feb. 2006.
- [98] F. C. Choo, M. C. Chan, and E.-C. Chang, “Robustness of DTN against routing attacks,” presented at the 2010 Second International Conference on Communication Systems and Networks (COMSNETS), Bangalore, India, Jan. 2010.
- [99] J. Burgess, G. D. Bissias, M. D. Corner, and B. N. Levine, “Surviving attacks on disruption-tolerant networks without authentication,” in *Proceedings of the 8th ACM International Symposium on Mobile Ad-hoc Networking and Computing (MobiHoc '07)*, Montreal, Canada, Sep. 2007, pp. 61-70.
- [100] D. Ellard and D. Brown, “DTN IP Neighbor Discovery (IPND),” [Online]. Available: <http://tools.ietf.org/html/draft-irtf-dtnrg-ipnd-01>. [Accessed: Nov. 19, 2011].

- [101] D. Ellard, R. Altmann, A. Gladd, D. Brown, “DTN IP Neighbor Discovery (IPND),” [Online]. Available: <http://tools.ietf.org/html/draft-irtf-dtnrg-ipnd-02>. [Accessed: Mar. 19, 2013].
- [102] *IEEE Standard for Information Technology Telecommunications and information exchange between systems Local and metropolitan area networks Specific requirements. Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. Amendment 4: Protected Management Frames*. IEEE Std. 802.11w-2009, Sep. 2009.
- [103] *IEEE Standard for Information Technology- Telecommunications and Information Exchange Between Systems- Local and Metropolitan Area Networks- Specific Requirements Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications Amendment 6: Medium Access Control (MAC) Security Enhancements*. IEEE Std. 802.11i-2004, Aug. 2004.
- [104] C. He, J.C. Mitchell, “Analysis of the 802.11i 4-way handshake,” in *Proceedings of the 3rd ACM workshop on Wireless security (WiSe '04)*, Philadelphia, PA, Sep. 2004, pp. 43-50.
- [105] M. Eian, “Fragility of the Robust Security Network: 802.11 Denial of Service,” in *Proceedings of the 7th International Conference on Applied Cryptography and Network Security (ACNS '09)*, Paris-Rocquencourt, France, Jun. 2009, ser. Lecture Notes in Computer Science, vol. 5536, M. Abdalla, D. Pointcheval, P.-A. Fouque, and D. Vergnaud, Eds., Berlin / Heidelberg, Germany: Springer-Verlag, 2009, pp. 400-416.
- [106] J. Arkko, J. Kempf, B. Zill, P. Nikander, “SEcure Neighbor Discovery (SEND),” [Online]. Available: <http://tools.ietf.org/html/rfc3971>. [Accessed: Mar. 19, 2013].
- [107] T. Aura, “Cryptographically Generated Addresses (CGA),” [Online]. Available: <http://tools.ietf.org/html/rfc3972>. [Accessed: Mar. 19, 2013].

- [108] A. Shamir, "Identity-Based Cryptosystems and Signature Schemes," in *Advances in Cryptology, Proceedings of CRYPTO '84*, Santa Barbara, CA, USA, Aug. 1984, ser. Lecture Notes in Computer Science, vol. 196, G. Blakley and D. Chaum, Eds., Berlin / Heidelberg, Germany: Springer-Verlag, 1985, pp. 47-53.
- [109] M. Bellare, C. Namprempre, and G. Neven, "Security Proofs for Identity-Based Identification and Signature Schemes," *Journal of Cryptology*, vol. 22, no. 1, pp. 1-61, 2009.
- [110] P.C. Leong, "Some impacts of trusted computing on digital forensic," Cyber Crime Investigation Workshop 2006, Singapore, Nov. 2006. [Invited Talk]
- [111] L. Gong, R. Needham, and R. Yahalom, "Reasoning about belief in cryptographic protocols," in *Proceedings, 1990 IEEE Computer Society Symposium on Research in Security and Privacy*, Oakland, CA, USA, May 1990, pp. 234-248.
- [112] N.H. Cogan, *The complete Bill of Rights : the drafts, debates, sources, and origins*, New York: Oxford University Press, 1997.
- [113] In Re Boucher 2007. Case No. 2:06-mj-91, document 35, WL 4246473, 11/29/2007, United States District Court for the District of Vermont.
- [114] M. Ward, "Campaigners hit by decryption law," *BBC*, Nov. 2007. [Online]. Available: <http://news.bbc.co.uk/2/hi/technology/7102180.stm> [Accessed: Jan. 10, 2010].
- [115] C. Williams, "UK jails schizophrenic for refusal to decrypt files," *The Register*, Nov. 2009. [Online]. Available: http://www.theregister.co.uk/2009/11/24/ripa_jf1/ [Accessed: Jan. 10, 2010].
- [116] "Integrity Measurement Architecture (IMA)," *IBM Research*. [Online]. Available: http://domino.research.ibm.com/comm/research_people.nsf/pages/sailer.ima.html. [Accessed: Jan. 10, 2010].

- [117] S. Munetoh, “Trusted GRUB.” [Online]. Available: <http://trousers.sourceforge.net/grub.html>. [Accessed: Sep. 11, 2008].
- [118] A. M. Dunn, O. S. Hofmann, B. Waters, and E. Witchel, “Cloaking malware with the trusted platform module,” in *Proceedings of the 20th USENIX conference on Security*, San Francisco, CA, USA, 2011, pp. 26-26.
- [119] “Android-x86 - Porting Android to x86.” [Online]. Available: <http://www.android-x86.org/>. [Accessed: May 5, 2011].
- [120] R. Canetti, C. Dwork, M. Naor, and R. Ostrovsky, “Deniable Encryption,” in *Advances in Cryptology - CRYPTO '97*, Santa Barbara, CA, USA, Aug. 1997, ser. Lecture Notes in Computer Science, vol. 1294, B. Kaliski, Ed., Berlin / Heidelberg, Germany: Springer-Verlag, 1997, pp. 90-104.
- [121] O. Goldreich and L. A. Levin, “A hard-core predicate for all one-way functions,” in *Proceedings of the 21st Annual ACM Symposium on Theory of Computing (STOC '89)*, Seattle, WA, USA, May 1989, pp. 25-32.
- [122] J. Håstad and M. Näslund, “The security of all RSA and discrete log bits,” *Journal of the ACM*, vol. 51, no. 2, pp. 187-230, Mar. 2004.
- [123] “The GNU MP Bignum Library.” [Online]. Available: <http://gmplib.org/>. [Accessed: May 30, 2012].
- [124] A. Keränen, J. Ott, and T. Kärkkäinen, “The ONE Simulator for DTN Protocol Evaluation,” in *Proceedings of the 2nd International Conference on Simulation Tools and Techniques (SIMUTools '09)*, Rome, Italy, Mar. 2009.
- [125] A.-R. Sadeghi, “Trusted Platform Module,” lecture slides for Secure, Trusted and Trustworthy Computing, Technische Universität Darmstadt, Germany. [Online]. Available: http://www.trust.informatik.tu-darmstadt.de/fileadmin/user/_upload/Group/_TRUST/LectureSlides/STC-WS2011/

- Chap3_-_Trusted_Platform_Module.pdf.pdf. [Accessed: Jun. 3, 2012].
- [126] T. Hardjono and G. Kazmierczak, “Overview of the TPM Key Management Standard,” *Trusted Computing Group*. [Online]. Available: http://www.trustedcomputinggroup.org/resources/overview_of_the_tpm_key_management_standard. [Accessed: Jun. 3, 2012].
 - [127] “TCG Software Stack (TSS) Specification,” *Trusted Computing Group, Incorporated*, Mar. 2007. [Online]. Available: http://www.trustedcomputinggroup.org/resources/tcg_software_stack_tss_specification. [Accessed: Jun. 1, 2012].
 - [128] “Trousers - The open-source TCG Software Stack.” [Online]. Available: <http://trousers.sourceforge.net/>. [Accessed: Jun. 1, 2012].
 - [129] “TORSEC - Trousers for Windows.” [Online]. Available: <http://security.polito.it/trusted-computing/trousers-for-windows/>. [Accessed: Jun. 1, 2012].
 - [130] D. Challener, “Programming With Trousers,” Johns Hopkins University, Applied Physics Laboratory, Laurel, MD. [Online]. Available: <http://www.cylab.cmu.edu/tiw/slides/challener-handout.pdf>. [Accessed: Jun. 1, 2012].
 - [131] D. Challener, K. Yoder, R. Catherman, D. Safford, and L. V. Doorn, *A Practical Guide to Trusted Computing*, 1st ed. IBM Press, 2008.
 - [132] R. L. <pkcs-editor@rsasecurity.com>, “Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1.” [Online]. Available: <http://tools.ietf.org/html/rfc3447>. [Accessed: May 30, 2012].
 - [133] “U.S. Army requires trusted computing.” [Online]. Available: <http://www.securityfocus.com/brief/265>. [Accessed: June 1, 2012].

- [134] W. Ashford, “2012: Will this be the year TPM finally comes of age?” [Online]. Available: <http://www.computerweekly.com/news/2240157874/Analysis-2012-Will-this-be-the-year-TPM-finally-comes-of-age>. [Accessed: Jun. 18, 2012].
- [135] A. L. Young and M. M. Yung, “On Fundamental Limitations of Proving Data Theft,” *IEEE Transactions on Information Forensics and Security*, vol. 1, no. 4, pp. 524-531, Dec. 2006.