Available online at www.sciencedirect.com
ScienceDirect
Procedia Computer Science 5 (2011) 165–172
Procedia Computer Science

The 2nd International Conference on Ambient Systems, Networks and Technologies (ANT-2011)

Challenges in the Implementation and Simulation for Wireless Side-Channel based on Intentionally Corrupted FCS

Ali Najafizadeh[a], Ramiro Liscano[a], Miguel Vargas Martin[a], Peter Mason[b], Mazda Salmanian[b]

[a]*University of Ontario Institute of Technology, 2000 Simcoe Street North, Oshawa, Ontario, Canada L1H 7K4*
[b]*Defence Research & Development Canada, Ottawa, Canada*

Abstract

We report on the challenges faced in the implementation and simulation of a side-channel communication based on frames with an intentionally corrupted Frame Check Sequence (FCS). Systematically corrupted FCSs can be used to enable covert communications between nodes that share the same algorithm for deciphering the FCS. In order to assess the possibility in detecting this side-channel communication it is necessary to have the ability to simulate it as well as to implement it on actual devices. Nearly all simulators drop corrupted frames before they reach their destination, making it impossible to simulate any side-channel communication based on intentionally corrupted FCS. We present an example of the modifications required to prevent this as applied to a well-known simulator called Sinalgo. We also discuss problems encountered when trying to intentionally corrupt the FCS on actual devices.

Keywords: Wireless network side-channel communications; hidden-channel communications;

1. Introduction

Wireless communication is prone to packet errors caused by signal fading, collision, or shadowing effects. In packet-based communications any corruption to a packet is detected by using a Cyclic Redundancy Check (CRC) on the frame payload and appending information derived from this CRC into a Frame Check Sequence (FCS) field in the frame. Thus the FCS and the CRC serve similar purposes but in different stages of the communication

This mechanism could also be used to establish a form of hidden channel communication between rogue nodes in a wireless network. These rogue nodes could simply use a different FCS algorithm than the other nodes to communicate with each other. The idea behind this form of side-channel communications is to use frames with intentionally corrupted FCSs as the means for transmitting information between two side-channel-enabled wireless devices that are within communication range of each other. This form of side-channel communication is difficult to detect because it is basically interpreted as noise to the other nodes and these corrupted frames will be dropped immediately by other nodes.

An intrinsic property that works in favour of side-channel communication frames of this type is that they hide amongst naturally corrupted frames thus making detection harder. In a communication system using this proposed side-channel mechanism, the complete set of frames transmitted in the wireless network is composed of frames

Selection and/or peer-review under responsibility of Prof. Elhadi Shakshuki and Prof. Muhammad Younas.
doi:10.1016/j.procs.2011.07.023

with: (a) non-corrupted FCS; (b) naturally corrupted FCS; (c) intentionally corrupted FCS for side-channel communication; and (d) naturally corrupted FCS of messages that had been intentionally corrupted for side-channel communication. Thus, the problem of detecting the existence of this side-channel becomes one of distinguishing between naturally and intentionally corrupted FCSs, or determining that more frames appear to be in error than expected. The difficulty of detection depends on a number of factors including the proportion of intentionally corrupted FCSs with respect to naturally corrupted FCSs and the traffic volumes.

The use of hidden side-channels such as this one has been proposed before (see e.g., Szczypiorski [1]) but to the best of our knowledge, there is no implementation in wireless devices reported in the literature. The reason is a lack of hardware implementation due to the fact that most devices are shipped with locked Layer 2 chipsets that are not meant to be tweaked. This limitation has led researchers to use simulators instead of actual devices.

Odor et al. [2] presented an algorithm that could be used for side-channel communications using an intentionally corrupted FCS in Layer 2 of the protocol stack and performed a simple simulation utilizing Matlab/Simulink. In order to build on Odor et al.'s work we wanted to design an experimental test-bed as well as to explore detection mechanisms using more conventional network simulators.

In terms of putting together an actual test-bed we report on the problems we faced in achieving this in Section 3. In Section 4 we present challenges in choosing the right simulator. In terms of the use of simulators we present extensions to the Sinalgo simulator [3] in Section 5 that is required in order to perform side-channel analysis. In Section 6, we provide some insight as to what scenarios are suitable for our side-channel communication and what the needs for gathering important information are.

2. Related work

Two fields of study related to our work are termed covert channel and side-channel communications. Based on Wagner [16] and Murdoch [14] there are four different channels for communication. According to Murdoch, covert channel can be described as a communication in a computer system where the sender and the receiver plan to leak information over a channel which is not designed for that communication to take place, in violation of a required access control policy. On the other hand, side-channel is a communication over a channel which fails to comply with the security requirements where the sender leaks information inadvertently and the receiver decides whether or not to take advantage and retrieve the information. The work we present in this paper, while more similar to the above definition of a covert channel rather than a side-channel, could also be categorized as steganographic method.

Steganography has a long history and many researchers have studied a plethora of aspects of it. Steganography can be used to hide data inside pictures, audio, and video files without changing the originally intended functionality of the file. Steganography has also been studied in the context of internet protocols. Jankowski et al. [8], studied new way of exchanging steganograms data in LAN environment where they use ARP protocol in conjunction with improper Ethernet frame padding, to provide localization and identification of the member of a hidden group. Handel et al. [13] studied steganography in the seven layers of the OSI Network model. In their paper they point out some of the reasons why the OSI Network model is susceptible to data hiding such as design errors, project deadlines, production shortcuts, etc.

Wolf [9] proposed a steganography method for IEEE 802.3 frames which makes it possible to have a hidden bandwidth of up to 45 bytes per frames. Fisk et al. [11] reported on a method that uses the padding of TCP and IP headers in the context of active warden to have 31 bits per packet for steganographic communication. IPv6 has also been studied for steganographic communication. Lucena et al. [12] also used padding in IPv6. This technique offers multiple channels with up to 256 bytes per packet for that type of communication.

Thus the biggest difference between a steganographic channel as opposed to a covert or side-channel is that a steganographic channel uses an open channel for communication where both the sender and the receiver plan to prevent observers from detecting that channel. On the other hand, the concept of a subliminal channel refers to a covert channel which is not detectable. Thus, a subliminal channel is a special case of a steganographic channel, and while a steganographic channel can be used to create a covert channel, not all covert channels fall into the category of steganographic channels; a simple example of that would be watermarking, where a message is hidden in a file, such as image, audio or video, so that its source can be tracked or verified.

The concept behind our covert communication channel can be illustrated in simple terms by the Alice and Bob analogy used in the context of cryptography. Assume that Alice and Bob are in prison and they plan to communicate with each other. Unfortunately, the only way they can communicate is through Walter, the warden. However, If Walter finds out that Alice and Bob are communicating, he will close their connection. Walter is the network manager and he monitors network traffic. Alice and Bob cannot use cryptography to hide their messages as Warden would detect there is a communication going on. They have two options; they either use steganography techniques to hide their information in valid network packets or they hide information in seemingly valid network packets. In our case, they choose the latter by corrupting the FCS to make packets seem naturally corrupted and thus be ignored and discarded by Warden.

Szczypiorski introduced HICCUPS (HIdden Communication system for CorrUPted networkS) [1], a steganographic system capable of establishing a side-channel in the data link layer in a wireless LAN. HICCUPS leverages natural fading and interference in wireless signals to mask side-channel messages. Szczypiorski suggested three features that make a network suitable for HICCUPS. First, the medium must be shared among all nodes and sender and receiver must have a high probability of intercepting the messages. The second one is having an environment with a published algorithm for link layer cipher. The last thing is that environment has to offer some mechanism to validate the integrity of messages. An example would be an FCS. If all these 3 features exist in a system then three types of hidden channels could be introduced. The first channel would be based on manipulation of the IVs (initialization Vector) in the link layer encryption. Forging MAC addresses physically in link layer hardware would be the second channel.. The third type of channel is based on the data integrity mechanism (e.g., CRC or FCS). These entire three features can be found in IEEE 802.11 with WEP encryption. Sbrusch [15] suggested that almost 25% of wireless transmission could be corrupted for a variety of reasons such as interference and noise in the environment. The high tolerance for error opens a door for sending secret messages.

Szczypiorski also introduced two methods for encoding and transporting covert messages. The “basic mode” utilizes covert messages in network interface MAC address and cipher IVs. Unfortunately, there are only 216 bits in total per frame that can be used to transfer messages. Because of limited transfer rates of side-channels, Szczypiorski suggested to use them to exchange control messages. The “Corrupted frame mode” is introduced to be used for higher bandwidth. In this mode the entire payload of a frame can be used to transfer the covert messages. Also the FCS, the last 4 bytes of each frame, can be altered in such a way to notify other HICCUPS members that a covert message is being delivered inside the payload. Szczypiorski estimated that in 802.11b, 44kbits per second and in 802.11g, 216 Kbits per second can be transmitted. Even though HICCUPS offers very high bandwidth, it suffers from a serious implementation drawback: to the best of our knowledge there is no commercial wireless card that actually can inject and alter FCSs in frames via an existing software and hardware.

3. Experimental test-bed

We explored the possibility of implementing the side-channel communications using commercial devices both based on 802.11 and 802.15.4. Fundamentally, the challenge is to find a network card with a communication chip that allows an application to use its own FCS algorithm. We discovered that the computation of the FCS is locked and hard-wired into the communications chip set, not allowing these devices to send a frame with a different FCS than those available by the manufacturer. Ironically older communication chip sets are better suited for this than the newer ones. For example, Neufeld et al. [4] developed SotMAC – A Flexible Wireless Research Platform based on the DLINK DWL-AG530 PCI card that uses the Atheros AR5212 chipset that supported a flexible MAC layer. This network card and chipset are no longer manufactured. As far as we know, there are no manufacturers that are developing a communication chip for 802.11 that support an application-based calculation of the FCS. We have tested several promising devices such as AIRPCAP that support the capturing and transmission of nearly any form of packet, however, because of the inability to by-pass the hardware computation of the FCS it is impossible to use this device as a test-bed.

We also investigated side-channel communications over 802.15.4 devices like Telos-B [5]. Telos-B is one of a few 802.15.4 communication devices that can support application computation of an FCS. The Telos-Bs utilized consists of the Texas Instruments CC2420 communication chip set that can support application computation of an FCS. To simplify development on the Telos-B we leveraged the TinyOS environment. A simple application was developed to communicate a short string using side-channel communication to another node. Unfortunately, after a certain number of frames have been sent and received over the side-channel, the wireless and serial communications began to fail due to complexity and some possible bugs in TinyOS. The mote which received the side-channel frame also failed to receive any further frames. Because of the complexity of working with TinyOS, at what is effectively the driver layer, further work on developing a test-bed were set aside and the focus was placed on the use of a simulation environment.

4. Choosing the right simulator

In a previous paper [2] the authors provide an outline of the limitations of simulation environments like NS-2, Qualnet, Omnet++ and MATLAB/Simulink for use in side-channel communications. As a result of this review the MATLAB/Simulink simulator was chosen over others primarily due to the fact that it was one of the only environments where corrupt frames could be received by a node. The nature of Simulink and the way we designed the simulation imposed a limit of two nodes which later made us decide to move to a different simulation platform, one that was closer to a network simulator as opposed to a communication channel simulator like Simulink.

Choosing the right simulator is challenging for the following reasons:

- Some simulators take a significant amount of time and effort to evaluate and understand how they can be programmed, our experience have shown that 4-6 months is not unusual. There are number of well-known network simulators to choose from for example, NS-2, Qualnet, Omnet++, and Opnet. Most of these are relatively sophisticated platforms because they simulate many layers in the OSI networking stack. Modifying one layer, in particular a lower layer requires very good understanding of its interaction with the other layers.
- Most of the network simulators do not allow the intentional corruption of frames into the communication packets and they certainly do not allow these packets to be sent to other nodes. Typically the simulator will decide for each round which packets are corrupted based on some well studied channel propagation model and discard the packet before placing it in the receiving nodes incoming buffer.

We ultimately chose to use a simulator called Sinalgo [3], a network simulator that would allow for the crafting of invalid FCSs. Sinalgo offers many features all of which are nicely wrapped in a simple and well-structured way. Furthermore, Sinalgo is open source and is written in Java, making it fully portable; and allowing for easy modifications and additions to its source code compared to other simulators such as Qualnet or NS-2. Additionally, Sinalgo comes with a complete and user-friendly set of documentation. Nevertheless, the most important limitation of Sinalgo, for our purposes, is that nodes are not able to receive corrupted frames; once a frame becomes corrupted, the frame is not even sent to its destination.

5. Adaptations to Sinalgo for Side-Channel Simulation

We were able to identify the two interfaces responsible for dropping corrupted frames in Sinalgo and modified these. The following is a brief description of what these interfaces are and what they do.

- Interference Model Interface

  This model uses an algorithm to determine if interference and the coexistence of all current packets signals will cause a packet to become corrupted. If so, the packet is deleted without further processing. The algorithm used is the received Signal to Interference plus Noise Ratio or SINR algorithm [5].
- Reliability Model Interface

  This model determines whether a frame should be lost in the communication channel due to various physical factors such as signal strength, failures, etc.

To adapt Sinalgo for our needs we modified its source code so that nodes may both send and receive corrupted frames. To achieve this some code had to be re-designed and we had to create new structures that integrate into the original source code. The first aspect that needed to be modified was the message structure. The structure of messages is composed as follows. First of all, since we are dealing with frames, the Message class was renamed as “Frame”. Each frame needs to have a section for the data, and the last 4 bytes of each frame are reserved for the FCS. The FCS can be stored into an unsigned integer variable.

In 802.11 frames, the CRC is the equivalent of the FCS. The calculation of the FCS (or CRC) is performed by a class name CRC32. This class has a table of all calculated variables needed for the CRC32 algorithm to perform its functionality faster. It also has a static method called GetCRC32 which is used by the class Frame for the calculation of the FCS. In order to introduce a new message class into the Sinalgo framework, Sinalgo provides a set of classes and interfaces to do so. Figure 1(a) illustrates the class hierarchy of the *Frame* and *Message* classes. Furthermore, the class Frame overrides the *Message.clone()* method.

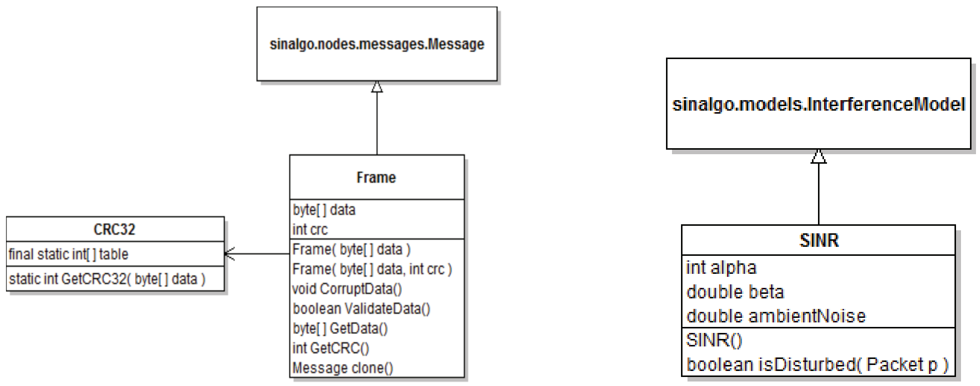
sinalgo.nodes.messages.Message
Frame
byte[ ] data
int crc
Frame(byte[ ] data)
Frame(byte[ ] data, int crc)
void CorruptData()
boolean ValidateData()
byte[ ] GetData()
int GetCRC()
Message clone()
CRC32
final static int[ ] table
static int GetCRC32(byte[ ] data)
sinalgo.models.InterferenceModel
SINR
int alpha
double beta
double ambientNoise
SINR()
boolean isDisturbed(Packet p)

Fig. 1. (a) Class Diagram of *Frame* class; (b) Class Diagram of SINR class

SINR also known as Signal Interference plus Noise Ratio is one of the best known interference models. It calculates a ratio $q = s / (n + i)$, which is the ratio of the received signal s and the sum of the ambient background noise n and the interference i caused by all simultaneously signals or transmission. The strength of the received signal is computed based on a simple exponential distance path loss model where the signal strength decreases exponentially based on the distance between the sender and receiving nodes. The transmission is successful if q is bigger than β, where β is a small threshold constant. However, because of the nature of Sinalgo, if any of those signals or frames becomes corrupted, it will be removed the packet from the system.

The next step was to modify the current implementation of SINR provided by the Sinalgo framework. We determined a way to change the SINR such that it delivers all messages to the target irrespective of the validity of the frame. Figure 1(b) depicts the SINR class diagram.

The SINR class has a method called *isDisturbed(Packet p)*. This method had to be overridden from the base class. Every time this method is called, Sinalgo passes one packet into it to see if that packet should be dropped or not. If method *isDisturbed(Packet p)* returns false, it means that the frame can be delivered to its destination, otherwise the frame is dropped. For our side-channel purposes we had to modify this method so that it always returns false. However how would Sinalgo know which packet is corrupted and which one is not? To address this issue, we implemented a new method inside the *Frame* class that if the packet becomes disturbed, the FCS in the frame will be changed to reflect that it is corrupted. Figure 2 is an activity diagram comparing the modified and original SINR algorithms.

Figure 2 shows the new implementation of the interference model based on SINR. The original SINR removes packets if they become corrupted, which runs counter to our purposes. We wanted to deliver the data to the destinations, corrupted or not. The concept behind the interference model is that once the *isDisturbed(Packet p)* method returns true, the back end implementation of Sinalgo will remove the packet from the system. In this case a corrupted packet is never delivered. In such a scenario a new implementation was required. So as the Figure shows, we are returning a false value in most cases that we need. The only way *isDisturbed(Packet p)* returns true is when the receiving node is out of range of the sender, represented by the value “Not in Range”.

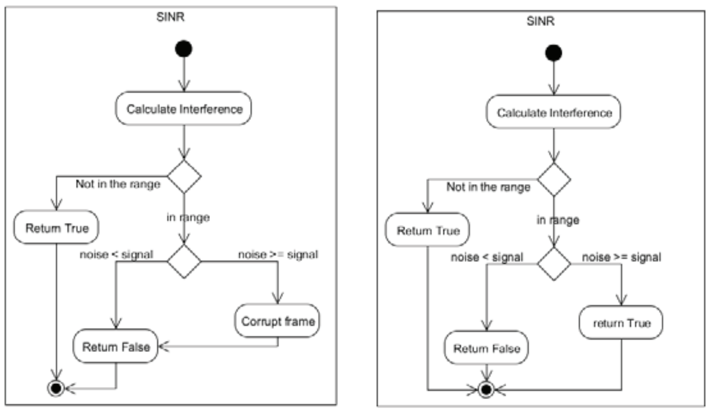

Fig. 2. Activity Diagram of the SINR implementation. The left hand side diagram is our modified version while the right hand side diagram shows the original Sinalgo implementation.

6. Simulation Scenarios

Thus far we have presented changes to the simulator to be able to manage the transmission of corrupted frames. This section presents our approach to the way we plan to use Sinalgo in order to detect and analyze side-channel communications.

For our scenarios we plan to have several wireless nodes communicating in a broadcast mode since we are primarily interested in Layer 2 communications. Collecting and analysing the data transmitted while executing a simulation is generally done by simply capturing and updating statistics at each node for each round in the simulation. However, this is not a realistic scenario for our situation, where we want to be able to defend a network from such communications, because this requires that each node would have to report its information to a coordinating node to determine if there is a potential side-channel communication. This implies that all nodes are trusted and will accurately report their statistics. A more realistic scenario is one in which some strategically placed “sniffer” nodes capture all communications within range and can cooperatively determine if side-channel communications are occurring. In order to do this an Agent Node has been created for this work whose sole purpose is to promiscuously monitor the air for packet transmissions. In summary, we define 3 types of nodes for our simulations that are:

- Normal Nodes: Normal nodes are a basic node. These kinds of nodes transmit and receive data using some distribution model such as Random or Poisson.
- Agent Nodes: Agent nodes are Normal Nodes that do not transmit data. They simply listen to the network and capture all packets their radio can hear. In addition they gather some statistical information for detecting anomalies in the area. Note that this type of node could not be supported on a regular simulator without the modifications we made because it would not receive any of the corrupted nodes.
- Malicious Nodes: As the name of the node implies, these nodes try to hide their communications by purposely corrupting the FCS of a frame. They can spoof any information such as MAC Address and they also act as a normal node. So at the start of communications an Agent Node would not notice a difference between Malicious Nodes and Normal Nodes. Again this node could not be supported if we had not made the changes to the simulator.

6.1. Messaging Distribution Models

One of the main challenges with broadcast communications is deciding when to communicate in order to minimize node interference. Because Sinalgo does not support any particular MAC layer communication models (for example 802.11 CSMA/CA) one quickly observes that if all the nodes send messages during the same simulation round that a large majority of the received frames will be corrupted because of signal interference resulting in all communications being ineffective. It is therefore necessary to come up with a simple communication strategy that can minimize collisions among the nodes without having to re-create the 802.11 MAC layer in Sinalgo.

A simple approach to reduce these collisions is to leverage a uniform random broadcast model. Some preliminary experiments showed that the results of using a uniform random approach still led to a significant number of packet collisions. We performed a simple experiment that commenced with 2 Normal nodes and an agent within range of each other and increased this to 15 Normal Nodes also all within communication range of each other. We then applied to these scenarios a desired message transmission rate (*r*) and used this rate as the desired rate for both a uniform and Poisson distribution model. The net effect is that the nodes are transmitting at a desired rate of communication *r* either in a uniform or Poisson distributed manner. One can easily see from Table 1 column 2 that using a uniform random distribution for communication has resulted in an 80% collision rate when 15 nodes communicate at a desired rate of 0.4 messages per round (our actual desired rate of message transmission was 40 messages / 100 rounds).

Table 1. Collision Rate using uniform Random over Poisson random.

# of nodes	Random Collision Rate	Poisson Collision Rate
2	0.5	0.1
10	0.75	0.3
15	0.79	0.4

A more realistic communication model is a Poisson Distribution Model because we have a large number of independent messages being sent, each message has the same chance of occurring, and the messages are of negligible length. The standard Poisson distribution is shown in Equation 1 where *k* is the number of occurrences (in our case this is the actual message transmission rate) and λ is the expected number of occurrences (in our case this is the desired message transmission rate *r*).

$$f(k, \lambda) = \frac{(\lambda^k e^{\lambda})}{k!} \tag{1}$$

To generate a random Poisson value from a uniform random value we use Knuth's algorithm [7]. We set the value of λ to be fixed for all the nodes in the simulation (this number represents the rate of message transmission of each node and was set to 40 in the simulations). At each simulation round we calculate a random Poisson value k using Knuth's algorithm as well as a random uniform value R between [0, 1]. We then compare the value of R against the Poisson probability value of k based on Equation 1. If the random value R is less than Poisson probability value of *k* then the node transmits a message otherwise it does not.

Table 1, column 3 displays the results of our experiments using a message rate of 40 messages / 1000 rounds (0.4 messages / round rate) for the Poisson distribution. It can be seen that the collision rate decreases by nearly half of that when using the uniform random distribution.

7. Conclusions

We have highlighted the challenges in creating a test-bed for the implementation of side-channeling. As discussed above, the lack of open communication chips that allow an application to compute their own FCS is really the most challenging obstacle in achieving an actual test-bed using commercial of the shelf products. It could be possible to re-visit using the Telos-B platform and instead of using TinyOS we could develop a driver using a micro-processor development kit for the Telos-B processor.

The lack of availability of adequate hardware forced us to move from a real test-bed into a virtual test-bed based on network simulators. This required a careful selection of an appropriate simulator along with adequate modifications so that corrupted frames would be passed onto any receiving nodes. We did this with the simulator called Sinalgo and we envision that the changes described in this paper can be applied to other simulators. The challenge is to determine the modules that compute if the frame is corrupt and add to the frame class a custom FCS calculation. Having this capability has helped us to generate more realistic scenarios that utilize a sniffer (Agent Node) that we developed and deployed to capture all packets within its communication range.

This work has established the foundations to pursue the design of appropriate intrusion and attack detection systems. We plan on designing and test some anomaly detection mechanisms in the near future as part of this on-going project.

Acknowledements

This research has been supported in part by the Natural Sciences and Engineering Research Council of Canada.

References

[1] K. Szczypiorski. "HICCUPS: Hidden Communication System for Corrupted Networks.", In Proc. International Multi-Conference on Advanced Computer Systems, Miedzyzdroje, Poland, October 22-24, 2003, pp. 31 – 40.

[2] M. Odor, B. Nasri, P. Mason, M. Salmanian, M. Vargas Martin, R. Liscano., "A Frame Handler Module for a Side-Channel in Mobile Ad Hoc Networks." In Proc. LCN Workshop on Security in Communications Networks (SICK), Zürich, Switzerland; 20-23 October 2009, pp. 930 – 936.

[3] Sinalgo, "Simulator for Network Algorithms.", (accessed March 28, 2011) [Online]. Available: http://www.disco.ethz.ch/projects/sinalgo/

[4] M. Neufeld, J. Fifield, C. Doerr, A. Sheth and D. Grunwald, "SoftMAC – Flexible Wireless Research Platform", In HotNets-IV, Nov. 2005.

[5] M. Patrick, "Implementation of side-channel communication in Telus motes.", Internal Undergraduate Report, Faculty of Businees and Information Technology, University of Ontario Institute of Technology, Oshawa, Ontario, 2010.

[6] A. F. Molisch, "Wireless Communications", John Wiley & Sons, 2005. ISBN: 047084888X

[7] D. E. Knuth, "The Art of Computer Programming", Volume 2, Addison Wesley, Seminumerical Algorithms.

[8] B. Jankowski, W. Mazurczyk, K. Szczypiorski - Information Hiding Using Improper Frame Padding - 14th International Telecommunications Network Strategy and Planning Symposium (Networks 2010), Warsaw, Poland, 2010.

[9] M. Wolf, "Covert Channels in LAN Protocols", Proc. Wksp. Local Area Network Security (LANSEC), 1989, pp. 91–101.

[10] O. Arkin, J. Anderson J., "Ethernet frame padding information leakage", Atstake report, 2003.

[11] G. Fisk, M. Fisk C. Papadopoulos, J. Neil, "Eliminating Steganography in Internet Traffic with Active Wardens", In Proc: 5th International Workshop on Information Hiding, Lecture Notes in Computer Science: 2578, 2002, str. 18–35.

[12] N.B. Lucena,, G. Lewandowski, S.J. Chapin, "Covert Channels in IPv6", Proc. Privacy Enhancing Technologies (PET), May 2005, pp. 147–66.

[13] G. Theodore, Handel and Maxwell T. Sandford II, "Hiding Data in the OSI Network Model", Weapon Design technology Group, Los Alamos National Laboratory, Los Alamos, NM 87545.

[14] Murdoch, S.J., "Covert Channel Vulnerabilities in Anonymity Systems", University of Cambridge, Decemberb2007 [Online]. Available: http://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-706.pdf

[15] R. Sbrusch, "Network Covert Channels: Subversive Secrecy", SANS Institute InfoSec Reading Room.

[16] D. Wagner. "Suggestions for the passing of passphrases.", Usenet posting to sci.crypt and alt.privacy, 2005.