8-2014

A Layer 2 Protocol to Protect the IP Communication in a Wired Ethernet Network

Reiner Augusto Campillo Terrero

A Layer 2 Protocol to Protect the IP Communication in a Wired Ethernet Network

By

Reiner Augusto Campillo Terrero

Thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Networking and System Administration

Rochester Institute of Technology

B. Thomas Golisano College of Computing and Information Sciences

Department of Information Sciences and Technologies

August 2014

Rochester Institute of Technology

B. Thomas Golisano College
of
Computing and Information Sciences

Master of Science in Networking and System Administration

Thesis Approval Form

Student Name:	Reiner Augusto Campillo Terrero
Thesis Title:	A Layer 2 Protocol to Protect the IP Communication in a Wired Ethernet Network

Thesis Committee

Name	Signature	Date
Tae Oh, Ph.D. Chair		
Sharon Mason, M.S. Committee Member		
Bruce Hartpence, M.S. Committee Member		

DEDICATION

To my parents Sheila and Cesar, my sister Yolena and my wife Soribel. Thank you for your unconditional love and support.

ACKNOWLEDMENTS

It is very gratifying when we can look at the results of the combined effort of many people and institutions that contributed to our success. I would like to express my gratitude to all those who directly and indirectly supported me during my days as grad student; my greatest appreciation to:

- The government of The Dominican Republic, represented by the Ministry of Higher Education, Science and Technology (MESCYT), for granting me the scholarship that allowed me to initiate and complete my master's studies.
- Rochester Institute of Technology for accepting me as a graduate student and for the exceptional support and guidance they gave me. I would like to highlight the wonderful assistance given by Mrs. Diane Ellison.
- Professor Dr. Tae Oh for being a great mentor to me, for trusting in my ideas and for always encouraging me to continue with my research. As a thesis advisor, you always gave the extra mile. I'm also grateful to my committee members Professor Sharon Mason and Professor Bruce Hartpence; thank you for your accurate observations and input.
- The College of Computing and Information Sciences for the exceptional advisory given to me while I was enrolled as a grad student. I would like to particularly thank my academic advisor Mrs. Susan Herzberg. Regardless of how complicated a particular situation could be, your huge spirit and sweet understanding were always a fresh air that always put me on the right track.
- The Dominican Advanced Network for Research and Education (RADEI) for allowing me to use one of their servers and the virtual environment where we ran most of the tests shown in this thesis. Special thanks to Dr. Jose Pedro Diaz Gonzalez for his constant support and for his wonderful advices. Jose, I feel really lucky for having the support of such a veteran as you.
- Dr. Felix Farias for reviewing the structure of my thesis and advising me in regards to the form and style of this. Your advices were always very useful and straightforward.

- My cousin, Rilke Ulloa (Xenomuta), for always clearing up my doubts about IT security and for the incredible advices related to the theory and practice of the Linux Kernel, and the programming tools to manipulate Ethernet frames. You have always been a motive of inspiration for my researches in the IT field.
- My roommates: Alberth, Marlenny, Evelyn and Raysa. There are not enough words to describe the wonderful moments we spent together, supporting us each other to stay always on track. You were and still are part of my family. I also want to thank all new friends I made at RIT.
- Mr. James Stefano for allowing me to work on campus as his assistant and for always encouraging me to keep working hard. I will never forget the relaxing conversations we had at lunch break.

There are also those who have been with me during the good and bad days of my life, those who have always believed in me and have always encouraged me to move ahead and pursue my dreams:

- Thank you God almighty, for giving me the wisdom, strength and perseverance to achieve my goals.
- To my wonderful parents Sheila and Cesar: You have been a great inspiration for me to never give up, to go ahead and to fight for what I want. You taught me that we must always do things right, assume responsibility for our actions and always love what we do. I remember as a kid how you constantly confronted the most difficult of the situations to assure a better future for my sister and me, and how you sacrificed everything for us. I don't have enough words to express my love and how thankful I'm to you. All I am now is because of your constant support, love and dedication to my sister and me. Thank you for being my parents.
- To my great sister Yolena: Ever since we were kids, you never stopped surprising me in regards to your perseverance to hold and defend your decisions and ideas. I, as your elder brother, have always admired that determination of yours and many times in my life, I have thought about you when I have needed to hold my position about something, as was necessary in this thesis project. Thank you very much.

- To my beloved wife Soribel: You have always been by my side inspiring me and motivating me to give the best of myself. The way you assumed the sacrifice of us being separated for so long while I was doing my master's studies was a remarkable demonstration of your love and understanding. I remember the moments when I was explaining to you some of the images I present in this thesis, and how your input was a huge help for me to make them more understandable. I remember the moments of frustration during the testing phase when things got really complicated and how your voice and the soft touch of your hand kept me calmed. When the time came to defend this thesis in front of my committee members, you were my inspiration to prepare my defense as a lawyer preparing and defending its case in court. You brought the necessary equilibrium to make this project a reality. Thank you for your love and support.
- To my lovely grandmother Elena Galarza: Your example of honesty, commitment, perseverance, love, bravery, discipline and hard work has stayed impregnated within all your family. Thank you for all your teachings and your eternal love.
- To my great uncles, aunts and cousins: Thank you for your unconditional support during all my life. Your actions are the true representation of love and family unity.
- To my mother-in law, brothers-in law and sisters-in law: Thank you for believing in me and for being my second family.
- To all my friends that have always been there for me: Thank you very much.

ABSTRACT

A Layer 2 Protocol to Protect the IP Communication in a Wired Ethernet Network

Reiner Augusto Campillo Terrero

Supervising Professor: Dr. Tae Oh

The IP protocol is the preferred data communication mechanism used nowadays. Data encapsulated using IP can be compromised if it is sent in clear text or without integrity protection, and even using known protocols to protect the confidentiality, integrity and authenticity of this data, the EtherType field of the Ethernet frames and the header of the IP packets in a wired Ethernet network still remain exposed opening possibilities for an attacker to gain knowledge of the network, cause a denial of service attack or steal information.

In this thesis, we propose a new protocol that protects the confidentiality, integrity and authenticity of the IP communication in a wired Ethernet network. This new protocol operates in the layer 2 of the OSI model, and for each Ethernet frame, it encapsulates the EtherType field and the entire IP packet into a new PDU structure that is partially encrypted. Integrity and authenticity are assured by an HMAC value or a digital signature calculated over the entire frame. We ran several tests to analyze the security characteristics and performance impact of our proposed solution; the results of these tests demonstrate that all traffic is effectively protected and that an attacker or eavesdropper wouldn't know the type of protocols, IP addresses or any other data travelling across the network. It is also demonstrated that under certain conditions, performance is not highly impacted and is feasible to protect the network communication with our new protocol.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1. INTRODUCTION

The IP protocol is one of the key elements that make communication possible in today's data networks; its main purpose is to provide an addressing mechanism for the delivery of data between two hosts regardless of their physical location [1]. Data communicated using this protocol is encapsulated into IP packets [2] that, in their basic structure, are sent over the network in clear text allowing any eavesdropper to read the entire content of what has been transmitted; moreover, the IP communication was designed without taking into consideration any need of confidentiality [3], and by default, all data transmission between two hosts can be compromised.

Known protocols, as for example TLS, SSH and IPSec, have been developed to protect the confidentiality and integrity of the information transmitted over IP [4], but fail to offer this protection to the header of the IP packets leaving it exposed in clear text and opening possibilities for an attacker to gain knowledge of the network, disrupt the communication or steal information. In the case of wireless Ethernet networks, this problem has been thoroughly approached, and protocols such as WPA and WPA2 were developed as a solution to protect the confidentiality and integrity of the entire IP packets, including the IP header and the transmitted data [5]; however, the problem is still present in wired Ethernet networks. In order to protect the entire IP communication in these networks, it is necessary to increase security at the OSI's layer 2 level by offering confidentiality protection to the EtherType field of the Ethernet frame and the encapsulated IP packet, and integrity protection to the entire Ethernet frame.

Some solutions have been proposed to address the lack of confidentiality and/or integrity of the transmitted data over wired Ethernet networks. The PPP Encryption Control Protocol, for example, defines a standard method to encrypt the information in a PPP link, but it doesn't specify any method to protect the integrity or authenticity of the transmitted data, providing only confidentiality protection [6]. Another solution was proposed by Dr. Kwei Tu [7] providing authentication, integrity, confidentiality and replay attack protection at the link layer level to the communication between two nodes; however, his solution is tied only to AES as the encryption algorithm and HMAC-SHA-2 as the hashing algorithm; it doesn't provide flexibility to use other algorithms. The IEEE also proposed a solution with the standard IEEE 802.1AE called MACSEC [8] and provides confidentiality, integrity and authenticity protection at the layer 2 level; however, its implementation can represent a huge investment in new hardware, limiting the scenarios where it can be implemented. Another solution was proposed by Yves Igor Jerschow, Christian Lochert, Bjorn Scheuermann and Martin Mauve [9] with a protocol called Cryptographic Link Layer (CLL) which provides authentication, integrity, confidentiality and replay attack protection to the IP packets in the link layer; however, this protocol strongly relies on digital certificates to authenticate hosts. In addition, encryption of data is optional and doesn't include ARP, broadcast or DHCP packets.

In this thesis, we propose a new layer 2 protocol called Packet Security Protocol (PSP) to protect the confidentiality, integrity and authenticity of the IP communication in a wired Ethernet network. Our proposed solution is designed for flexibility, allowing the use of multiple encryption and hashing algorithms as well as multiple digital certificate standards. When protecting data with PSP, the ethertype field of the Ethernet frame is replaced with a new value that indicates the PSP protocol; the protected data is then encapsulated into a new PDU structure that includes the original ethertype field of the Ethernet frame and the entire IP packet, both encrypted with a symmetric key, and an integrity check value that can be either an HMAC value or a digital signature calculated over the entire Ethernet frame. Our proposed protocol also includes multiple options to protect the communication against reply attacks.

To demonstrate the proof of concept of our solution, we will develop a program in C++ that implements the new protocol in a Linux operating system. This demonstration will be based on a methodology consisting of two parts. First, we will analyze the security of our proposed protocol by performing different attacks to compromise the communication between two hosts, and later, we will protect the same communication with our solution; we will also analyze the impact on security caused by the use of different modes of operation of a block cipher. Second, we will run multiple tests to 7 different protocols to measure the performance of the network and the performance of the hosts using our proposed solution.

The results of our experiments will have a clear answer to the following questions:

1. What are the real consequences of leaving the IP header unprotected in a wired Ethernet Network?
2. When encrypting information, what is the impact caused by different modes of operation of a block cipher in terms of security?
3. Is the network and host performance highly impacted when protecting the confidentiality and integrity of the entire IP packet?
4. Is it convenient for any kind of scenario to offer such level of protection?

The main contribution of this thesis is to present a new protocol that protects the confidentiality, integrity and authenticity of the entire IP packets in a wired Ethernet network and is also flexible to be used by any host, under any networking scenario.

The rest of this work is presented in the following order: Chapter 2 thoroughly elaborates on the problem statement. Chapter 3 gives a broad explanation about the related work. Chapter 4 presents all details of the proposed solution. Chapter 5 talks about the implementation process, including the methodology and tests used for our experiments. Chapter 6 presets the results of the experiments, a security and performance analysis of our proposed protocol, and the answer to all the questions formulated in this chapter. Chapter 7 presents the general conclusions and the future work.

2. PROBLEM STATEMENT

Several mechanisms and protocols have been developed to protect the confidentiality and integrity of the IP communication. An example of these protocols is IPsec, a suit of protocols that provides mechanisms to protect the confidentiality, integrity and authenticity of the payload of an IP packet. It also can, in its tunnel mode, encapsulate the entire IP packet (header + payload), add a new unencrypted IP header, and prevent the original header information from being exposed [10].

In a wireless Ethernet network, the header and payload of an IP packet is encrypted and integrity protected at the OSI's layer 2 level using protocols such as WPA and WPA2 [5] preventing unauthorized users from understanding the frames traveling across the network. This scenario is different in a wired Ethernet network because existing security protocols, as for example IPsec, SSH or TLS, can protect the transmitted data; however, this protection occurs in the network and upper layers of the OSI model [4], leaving the EtherType field of the Ethernet frames and the header of the IP packets exposed in clear text absent from any confidentiality and integrity protection. To better understand this problem, it is necessary to talk about layered models.

Data communication between two hosts is based on the OSI and the TCP/IP layered models [11]. If a layer is protected but then encapsulated into a less secure layer, a weak link in the security of the communication is created giving the attackers the possibility to gain knowledge of the network, disrupt the communication, cause a denial of service, impersonate one of the involved parties or steal information. This is exactly the existing problem in wired Ethernet networks. The IP protocol operates at the layer 3 or Internet layer of the TCP/IP model [1]; In order to protect the entire IP communication, it is necessary to increase security at the layer 2 level by offering confidentiality protection to the ethertype field of the Ethernet frame and the encapsulated layer 3, and integrity protection to the entire Ethernet frame. To demonstrate this statement, we'll generate in chapter 6 an attack that will allow us to gain access to an SSH server by exploiting a vulnerability of a layer 2 protocol; we will later neutralize this attack by protecting the communication with our proposed solution. We also make reference to Kenneth G. Paterson and Arnold K. L. Yau [12] who demonstrated 3 types of attacks that allow compromising the information transmitted over an IPSec tunnel using ESP encryption without integrity protection. Ventzislav Nikov [13] also demonstrated a DoS attack against an encryption only IPsec ESP without integrity protection. These attacks are possible because attackers can have access to an unencrypted IP header and can tamper the captured packets because these are not integrity protected. With a full encryption of the EtherType field and the encapsulated IP packets in the data link layer or layer 2, these specific attacks could be avoided even if the IPsec ESP doesn’t have integrity protection.

3. RELATED WORK

Research and development of protocols and technics to protect, in the layer 2 of the OSI model, the confidentiality or integrity of the information transmitted over a wired network expands to several solutions that, to our understanding, don't offer a complete protection to the information.

The Point to Point Protocol (PPP) is a well-known solution used to transmit other protocol's packets in a point to point link. Several methods have been defined to increase the functions of this protocol, including the ability to encrypt the entire encapsulated packet as specified in the PPP Encryption Control Protocol (ECP) [6]. This protocol defines a standard method to encrypt information in a PPP link and is open to any encryption algorithm, but it doesn't specify any method to protect the integrity or authenticity of the information being transmitted, providing only confidentiality protection [6]. Another disadvantage of the ECP protocol is that it can be used just after the Link Establishment Phase and the Authentication Phase of the PPP protocol, and not before. It means that an attacker can sniff the entire Link establishment phase of PPP in clear text and send multiple attacks to avoid a successful link negotiation.

Any protection to the information transmitted in a PPP link must be done through additional protocols. Confidentiality protection is achieved through ECP, but authentication protection must be done through different authentication protocols that can send the information in clear text or as a cryptographic hash. The big flaw of PPP and ECP is that there isn't a defined protocol to protect the integrity of the information [6][14]. It means that an attacker can manipulate and modify the data and the receiving end won't realize that it was modified.

Other defined protocols to protect encapsulated traffic in a PPP link are PPP Triple-DES Encryption (PPP-3DES) and Microsoft Point to Point Encryption (MPPE). The first uses 3DES as encryption algorithm, and the second uses the RC4 encryption algorithm, but both protocols are affected by the PPP and ECP design flaws mentioned in the paragraphs above [6][14][15]. In the case of Microsoft Point to Point Encryption, the RC4 encryption algorithm is vulnerable to key attacks and an attacker can retrieve the key, just as demonstrated by Fluhrer, Mantin and Shamir [16].

Dr. Kwei Tu [7] proposed a protocol to provide authentication, integrity, confidentiality and replay attack protection at the link layer level to the communication between two nodes. This protocol uses a shared secret data and time codes between the communicating devices. It also uses HMAC-SHA-2 as the only algorithm for data authentication, integrity and session key generation; and AES as the only data encryption protocol. Dr. Tu explains the implementation of this protocol on a CCSDS (Consultative Committee on Space Data Systems) Data format but does not specify if it could be used to protect Ethernet, PPP or any other layer 2 protocol. Dr. Tu's solution is limited to only one encryption and hashing algorithm. His protocol is limited to the vulnerabilities found on the only algorithm it uses. It also uses time codes for the communication which could be either convenient or not depending on the scenario where it is implemented.

The IEEE developed the IEEE 802.1AE standard known as the media access control security (MacSec) standard for local and metropolitan area networks. It provides confidentiality and integrity protection to trusted network hosts. MacSec uses the terms Mac Security Entity (SecY) to define the host or network element that uses MacSec, and Secure Association Key (SAK) to define the secret key used between two hosts that have established a Secure Association [8]. Even though MacSec defines encryption and integrity protection, it has to rely on the standard IEEE 802.1X-2010 for authentication and key management [17].

MacSec combined with IEEE 802.1X-2010 offers a good protection to the layer 2 frames; however, it must be supported by the physical hardware of the network including network switches. This condition may force to make investment in new hardware, limiting the number of scenarios where it can be adopted.

Yves Igor Jerschow, Christian Lochert, Bjorn Scheuermann and Martin Mauve [9] proposed a protocol called Cryptographic Link Layer (CLL) which provides authentication, integrity, confidentiality and replay attack protection to the IP packets in the link layer. The Cryptographic Link Layer relies on digital certificates and HMAC values to authenticate the frames transmitted by a host. In the case of digital certificate, it binds the Mac and IP address of the host and the receiver can always validate the authenticity of the frame based on these parameters.

CLL encapsulates the IP packet and adds a new header and a HMAC value or digital signature to the protected frame. It works in the following way: When a host needs to communicate with another host, it must use the ARP protocol to map the MAC and IP address. These ARP packets have a nonce and a timestamp and are authenticated using a digital certificate issued by a certificate authority. The next step is to establish a security association between both hosts. In this stage, hosts negotiate different security parameters and a common symmetric key to calculate the HMAC of the frames and use this value to authenticate and protect the integrity of the frames instead of using digital certificates. Hosts can optionally decide to encrypt the IP packet as well.

We consider CLL to have the closest approach to a complete and efficient protection of the IP packet at the data link layer level in a wired Ethernet network; however, it has certain limitations that can result in an administrative burden to the network administrator. It also has other disadvantages that can allow an attacker to gain knowledge of the network, especially all information contained in the IP headers of the packets, and find new ways to disrupt the communication and compromise the data.

CLL uses timestamps to protect the ARP and security association packets from reply attacks. It also relies exclusively on digital certificates to authenticate ARP, DHCP and broadcast packets and doesn’t include another way to authenticate these. CLL is not open to disable timestamps or digital certificates, which requires the network to have a certificate authority and a way to keep all hosts with their clocks synchronized. These two requirements, depending on the scenario, can significantly increase the administrative tasks of the network.

CLL offers optional confidentiality protection. Data is transmitted in clear text, and even if an encryption algorithm is used to encrypt the IP packet, the packet type field is always exposed in clear text. Something to note about CLL is that only unicast IP packets can be encrypted, and it occurs only after two hosts have established a security association.

CLL doesn’t offer the option to encrypt ARP, DHCP or broadcast packets which, in addition to exposing the packet type field in clear text, allows eavesdroppers to always know the type of traffic that is being transmitted and the IP addresses and all other fields of the IP header of the packets. This exposed information can open new possibilities for attackers to negatively impact the network.

4. PACKET SECURITY PROTOCOL

In this thesis, we propose the creation of a new data link protocol that will provide a secure transmission preserving the confidentiality, integrity and authenticity of the information sent over a wired Ethernet network. From this point on, we will refer to our proposed protocol as Packet Security Protocol or PSP.

We designed PSP considering the following characteristics:

- It must provide a mechanism to protect the confidentiality, integrity and authenticity of the information being transmitted.
- It must protect the communication from replay attacks.
- It must act as a security layer regardless of the network protocol being used.
- It must encrypt the information contained in all fields of the header of the used communication protocol.
- It must not send relevant information in clear text at any time.
- It must support multiple hashing and encryption algorithms.
- It must support multiple digital certificate standards.
- It must be completely open to allow a network administrator to choose the security parameters used to protect the communication.
- It must be compatible with existing network equipments.

The ethertype field of an Ethernet frame and the header of the IP communication protocol in its versions IPv4 and IPv6 are transmitted in clear text even if the entire IP packet is encrypted and tunneled by any known network security protocol. This lack of protection makes the transmitted information more vulnerable to any kind of attack.

As a way to protect the IP communication and avoid attacks based on information disclosed by any field in clear text, packet injection or data tampering, PSP encrypts the ethertype field and the whole network packet and encapsulates it into a new protocol data unit (PDU) that contains a new header, the encrypted data and an integrity check value calculated for the entire layer 2 frame.

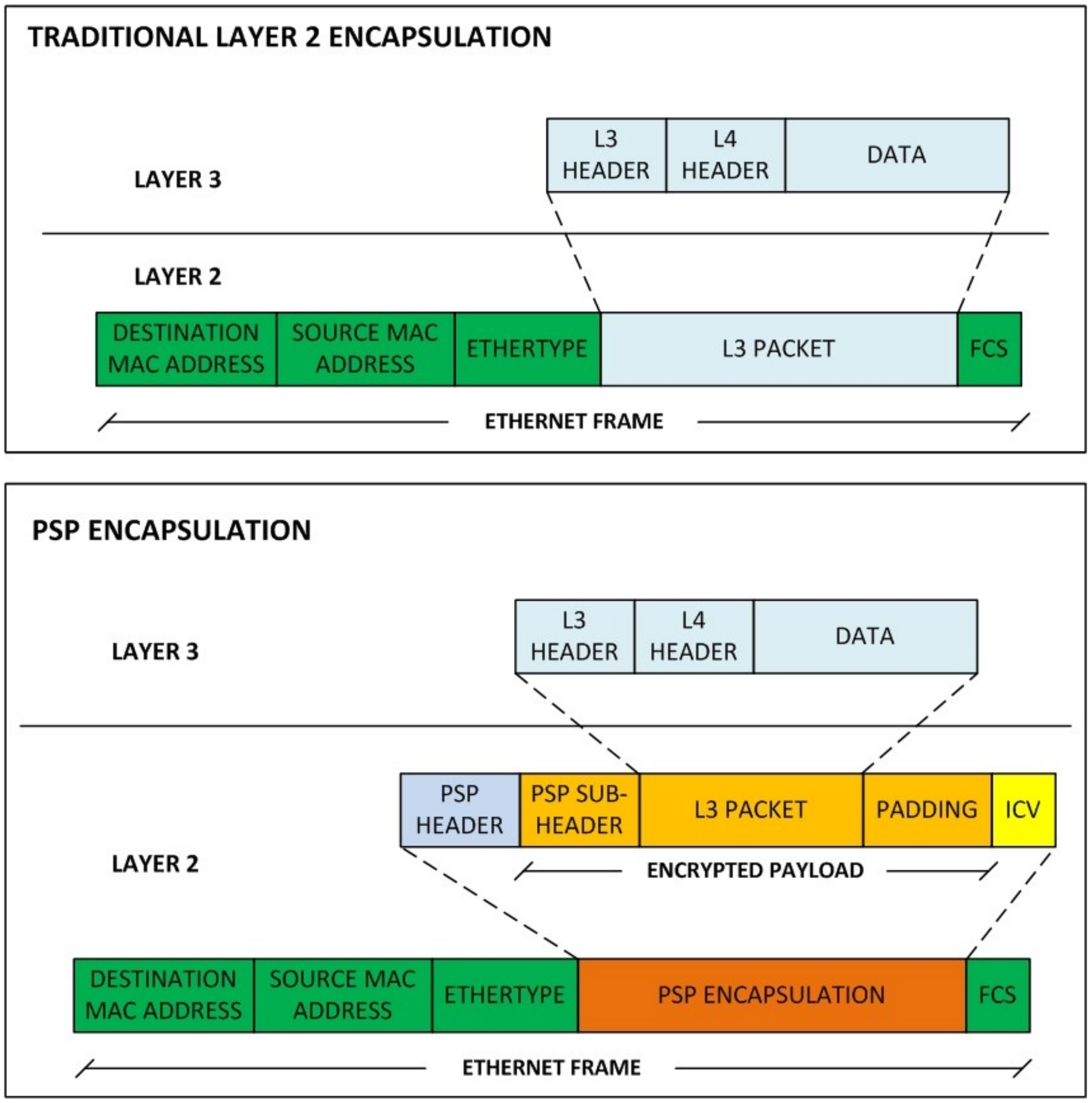
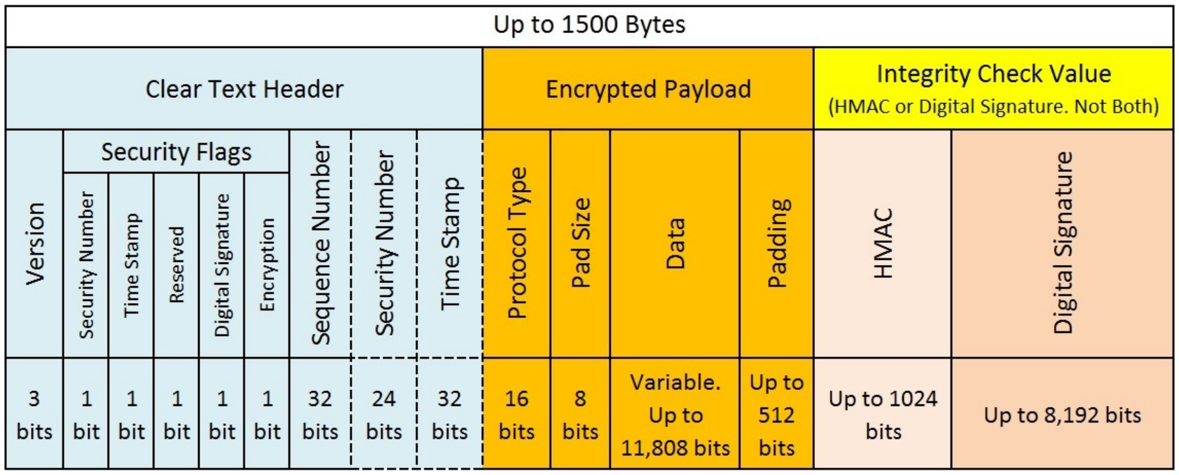


Figure 4.1. Traditional Layer 2 vs. PSP Encapsulation

A general overview of the traditional layer 2 encapsulation vs. the PSP encapsulation is depicted in Figure 4.1. Taking the OSI model as a reference in a regular data network communication, layer 3 packets are encapsulated into a layer 2 frame and then transmitted over a physical medium. PSP respects and works together with the traditional encapsulation process, but instead of encapsulating the network packet directly into a layer 2 frame, it is first encapsulated into the PSP Protocol Data Unit, and then this PSP PDU is encapsulated into a layer 2 frame before being transmitted over a physical medium.

4.1. PDU Structure

The PSP Protocol Data Unit (PDU) is composed of 3 sections: clear text header, encrypted payload and integrity check value.



Up to 1500 Bytes														
Clear Text Header									Encrypted Payload				Integrity Check Value (HMAC or Digital Signature. Not Both)	
	Security Flags													
Version	Security Number	Time Stamp	Reserved	Digital Signature	Encryption	Sequence Number	Security Number	Time Stamp	Protocol Type	Pad Size	Data	Padding	HMAC	Digital Signature
3 bits	1 bit	1 bit	1 bit	1 bit	1 bit	32 bits	24 bits	32 bits	16 bits	8 bits	Variable. Up to 11,808 bits	Up to 512 bits	Up to 1024 bits	Up to 8,192 bits

Figure 4.2. PSP Protocol Data Unit Structure

Each section of the Packet Security Protocol PDU structure, along with the fields they contain, is shown in Figure 4.2.

The clear text header has information about the PSP version and the different security parameters used to protect the data. It contains 3 mandatory fields and 2 others that are optional:

- **Version (3 bits):** It Indicates the PSP version. The Packet Security Protocol proposed in this thesis is version 0.
- **Security Flags (5 bits):** This field contains 4 bits that indicate the reply attack protection mechanism, confidentiality and integrity protection of the encapsulated data. There is a fifth bit that is reserved for future purposes.
    - **Security Number Flag:** If set to 1, a security number will be used as reply attack protection mechanism combined with a sequence number. This is an optional feature and is set to 0 by default.
    - **Time Stamp Flag:** If set to 1, a time stamp will be used as reply attack protection mechanism combined with a sequence number. It can also be

combined with a security number. This is an optional feature and is set to 0 by default.

- **Reserved Flag:** This bit is reserved for future security purposes.
- **Digital Signature Flag:** This flag is used to indicate if either an HMAC or Digital signature is used to preserve the authenticity and integrity of the protected data. 0 indicates HMAC and 1 indicates Digital Signature.
- **Encryption Flag:** This flag indicates if the protected data will be either encrypted or not. By default, this flag is set to 1 (encryption on).

- **Sequence Number (32 bits):** It is a number used to keep track of the order of the frames. It also works as a protection against replay attacks. This number is increased every time a frame is sent.
  If a frame is received with a sequence number lower or equal to the sequence number of the last received frame, it is automatically discarded. The sequence number of the first transmitted frame always starts with a random value.
- **Security Number (Optional. 24 bits):** This is an optional replay attack protection mechanism consisting of a random generated number that changes every certain time. All hosts must be synchronized with the same number and change it at the same time. When used, the security number flag is set to one. If a frame is received with a security number different than the value generated by the receiving host, the frame is automatically discarded.
  For this number to be used there has to be a method to effectively install it on the host. It can be done manually or through a server.
- **Time Stamp (Optional. 32 bits):** This is an optional replay attack protection mechanism that indicates the exact time at which a frame was transmitted. All hosts must be synchronized with the same time. When used, the time stamp flag is set to one. If a frame is received with a time stamp different than the current time of the receiving host, the frame is automatically discarded.

The encrypted section guarantees the confidentiality of the protected data. It contains a sub header, the protected data and padding bytes necessary to complete a block size when a block cipher is used for encryption. All fields in this section are encrypted with a defined cryptographic key using an encryption algorithm and can only be extracted or analyzed once the section has been decrypted.

- **Protocol Type (16 bits):** This field indicates the protocol that is being protected. When constructing the PSP PDU, the Ether Type field of the layer 2 frame is replaced with 0x1982 to indicate PSP as the encapsulated protocol. The replaced ether type value is then added to the Protocol Type field of the PSP PDU. This field can also indicate a sub protocol of PSP, like the Session Establishment Protocol.
- **Padding Size (8 bits):** It indicates the size of the Padding field in bytes.
- **Data (Variable):** This field contains the data (header and payload of an IP packet or any other protocol) to be protected. Depending on the security flags, encryption algorithm and integrity protection used, this field can be of up to 11,808 bits (1,476 bytes).
- **Padding (Up to 512 bits):** This field contains the padding bytes necessary to complete the size of a full block when a block cipher algorithm is used for encryption. The padding bytes can be all 0x00, 0xFF or any defined value. The size of this field can be of up to 512 bits (64 bytes).

The integrity check value (ICV) section guarantees the integrity and authenticity of the frame. This section contains either an HMAC value or a Digital Signature calculated for the entire layer 2 frame, but not both.

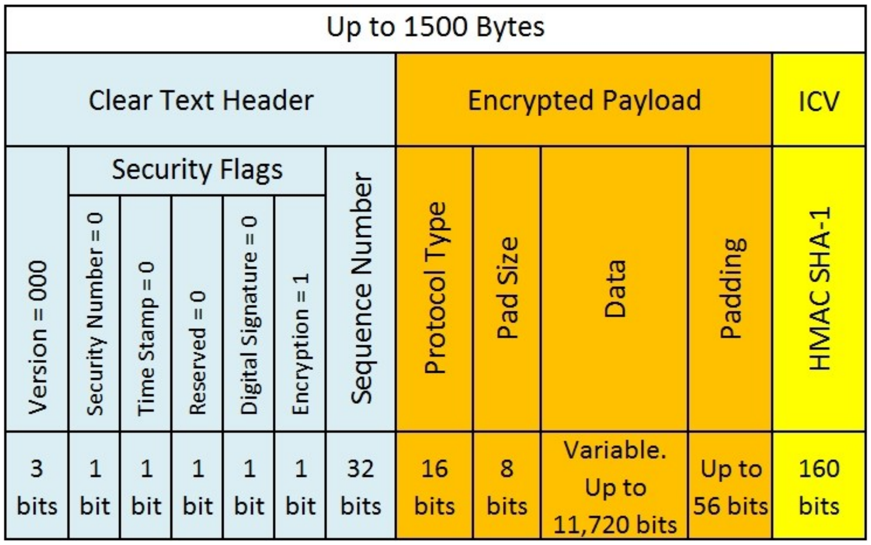
- **HMAC (Up to 1024 bits):** If the Digital Signature Flag is set to 0, an HMAC is calculated for the entire layer 2 frame. This kind of ICV can be used for any kind of data to be protected, including unicast, broadcast and multicast packets.
- **Digital Signature (Up to 8,192 bits):** If the Digital Signature Flag is set to 1, the entire layer 2 frame will be digitally signed using a private key. This kind of ICV is to be used exclusively for broadcast and multicast messages, and for unicast packets between hosts that haven't yet established a communication session. Digital certificates prevent attacks from legitimate users; however, it is necessary to use a Certificate Authority or a Public Key Infrastructure.

4.2. Default Parameters

By default, PSP uses only the sequence number as reply attack protection. It also encrypts the data with a shared symmetric key using the Blowfish encryption algorithm in Cipher Block Chaining mode, and calculates an HMAC value using the SHA-1 hashing algorithm to protect the integrity and authenticity of the frame. PSP encrypts the data and calculates the HMAC values with the same symmetric key. All default parameters of PSP are shown in Table 4.1.

Table 4.1. Default Parameters of PSP

Clear Text Header	Encrypted Payload	Integrity Check Value
Version = 0	Blowfish-CBC as encryption algorithm.	HMAC-SHA1 as hashing algorithm.
Security Number Flag = 0		
Time Stamp Flag = 0		
Reserved Flag = 0		
Digital Signature Flag = 0		
Encryption Flag = 1		
Sequence Number = Random value		



Up to 1500 Bytes											
Clear Text Header							Encrypted Payload				ICV
	Security Flags										
Version = 000	Security Number = 0	Time Stamp = 0	Reserved = 0	Digital Signature = 0	Encryption = 1	Sequence Number	Protocol Type	Pad Size	Data	Padding	HMAC SHA-1
3 bits	1 bit	1 bit	1 bit	1 bit	1 bit	32 bits	16 bits	8 bits	Variable. Up to 11,720 bits	Up to 56 bits	160 bits

Figure 4.3. PSP Protocol Data Unit Structure Using Default Parameters

PSP with its default parameters has the PDU structure shown in Figure 4.3. The first byte is equal to 0x01, which is the hexadecimal value representation of the PSP version and the Security Flags. The following 4 bytes (32 bits) correspond to the Sequence

Number, and it always starts at a random value. The next bytes correspond to the Encrypted Payload and there is not a fixed length for this section. The last 20 bytes (160 bits) correspond to the calculated HMAC-SHA1 value.

Even though, PSP uses Blowfish-CBC and HMAC-SHA1 as default encryption and hashing algorithms, it has been designed to support other algorithms as well as multiple digital certificate standards. Section 4.4.2 explains how two hosts establish a communication session and how they negotiate the encryption and hashing algorithms used to protect that session.

4.3. PSP Encapsulation Process in an Ethernet Environment

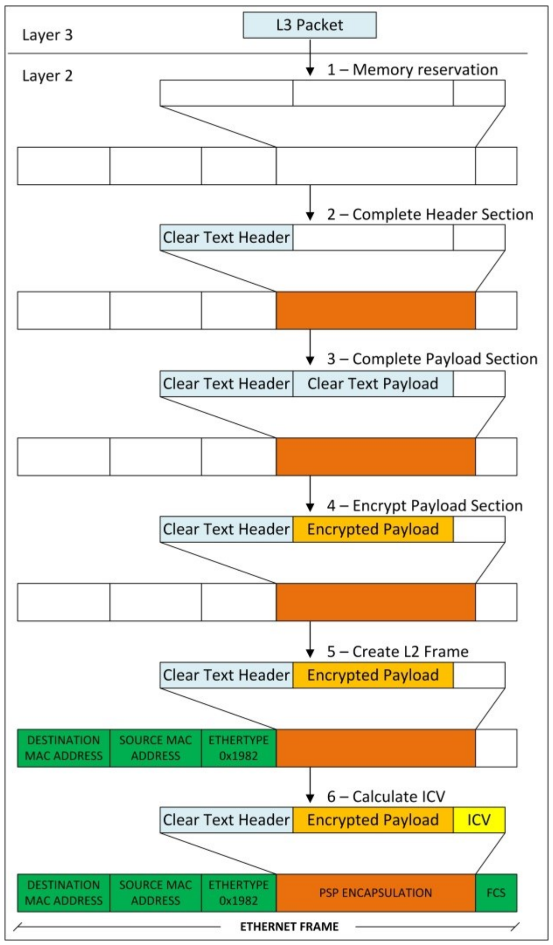


Figure 4.4. PSP Encapsulation Process in an Ethernet Environment

The PSP encapsulation process consists of 6 steps and is directly linked to the layer 2 protocol over which it is encapsulated. As depicted in Figure 4.4, when the layer 3 packet is sent to layer 2, the first step PSP takes in the encapsulation process is to reserve buffer memory to create the layer 2 frame. At this point, memory is reserved for each one of the fields of the frame, including the PSP PDU structure based on the security parameters used to protect the data.

Once memory has been reserved, the second step is to complete the PSP header with all necessary information. The PSP version, Security flags and the Sequence Number fields are added. If the Security Number and Time Stamp are necessary, these two fields are added as well.

After completing the PSP header, the third step is to complete the Payload section adding the layer 3 protocol into the Protocol Type, Pad size, Layer 3 packet fields and the padding bytes necessary to complete a block size in case a block cipher is used. Something very important to consider about this step is that the section is still in clear text, no data has been encrypted yet.

The fourth step is to encrypt the payload section using a defined encryption algorithm. In the case of algorithms that require an initialization vector (IV), PSP uses a unique IV/cryptographic key pair every time it encrypts the data to avoid the formation of patterns of blocks that have the same clear text. Every time a new symmetric cryptographic key is defined, PSP calculates the HMAC-SHA1 of this new key and uses the resulting value as a Base IV Number to calculate the initialization vector (IV), which is defined by a mathematical function of the Base IV Number (BIVN) and the Sequence Number (SN):

$$\mathrm{IV} = f(BIVN, SN)$$

With this approach, an attacker would never know the real IV used to encrypt the data.

By default, PSP uses addition as the mathematical function to calculate the IV:

$$\mathrm{IV} = BIVN + SN$$

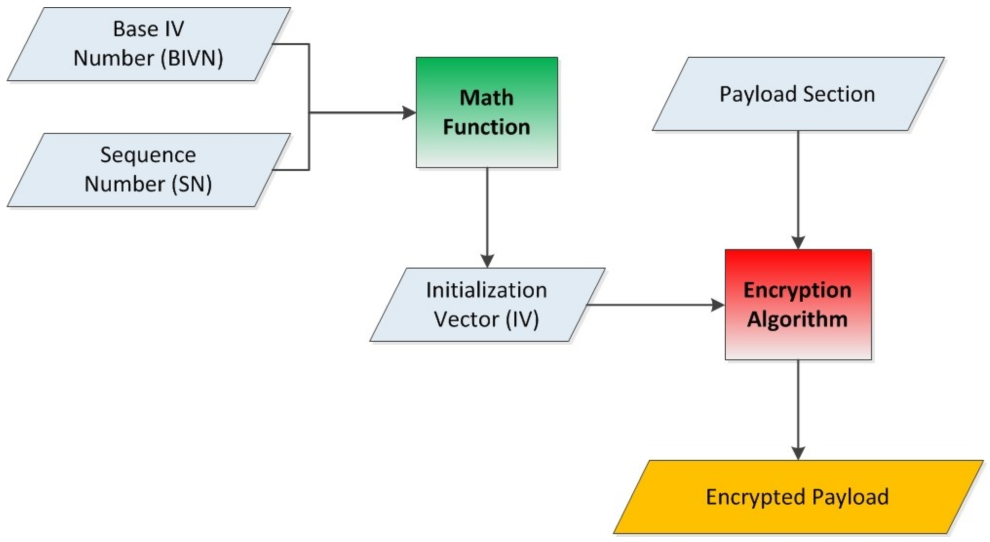


Figure 4.5. PSP Encryption Process

Once the Initialization Vector has been calculated, as depicted in Figure 4.5, it is passed to the encryption algorithm to encrypt the payload section.

The fifth step in the encapsulation process is the formation of the layer 2 frame. PSP adds the Destination and Source Mac Address, the Ethertype field, and the Encrypted Payload section. We have established 0x1982 as the value of the Ethertype field to identify PSP as the encapsulated protocol.

The last step corresponds to the calculation of the Integrity Check Value (ICV) and the Frame Check Sequence (FCS). PSP calculates the ICV for the entire layer 2 frame to guarantee the authenticity and integrity of each byte. Depending on the state of the Digital Certificate Flag indicated in the Header section, the ICV is either the HMAC or digital signature of the frame. After completing the ICV section, the frame is ready to be transmitted over the network. The FCS is either calculated by PSP or by the network adapter of the host.

4.4. PSP Communication Process

PSP works at the layer 2 of the OSI model. It must be configured, along with a common default Pre-Shared Symmetric Key, on all devices that will talk directly to each other in a communication network. All communication between two devices configured with PSP is encrypted and authenticated.

PSP uses a process of three sequential steps that will allow two devices to communicate with each other: Address Mapping, Communication Session Establishment, and Session Key Mapping.

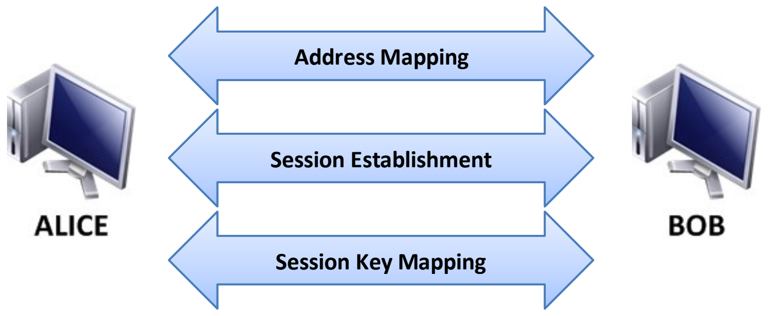


Figure 4.6. PSP Communication Process

As depicted in Figure 4.6, the first process is to map the network address of the peer host with its hardware address. Once the address mapping is done, PSP uses a Session Establishment Protocol to establish a communication session and be able to exchange data with the peer host; during this process, a session key is created to encrypt the communication and avoid using the default Pre-Shared Symmetric Key. After the communication session has been established, both hosts do a Session Key to Hardware Address Mapping. With this process, by creating an association between a session key and the hardware address of the peer, a device can know what session key to use to decrypt or encrypt data. This last process is necessary when a device has established communication sessions with more than one device.

Each process is explained in more details below.

4.4.1. Address Mapping

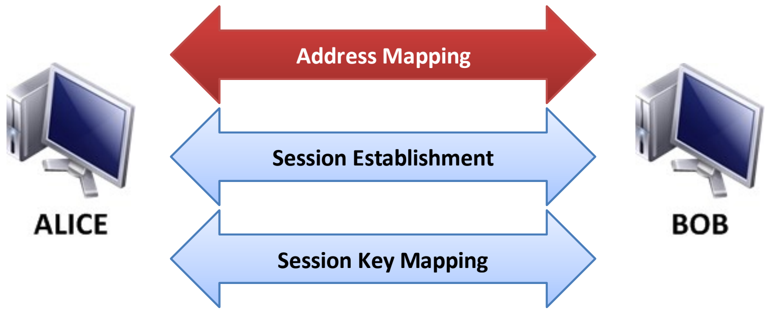


Figure 4.7. PSP Communication Process – Step 1

All network devices have two types of addresses: Hardware address, and Logical or Network address. Routing of packets in a communication network as we know it today, is possible because of the Logical or Network address of the devices involved. A Network Address indicates the location of a device and also, because of its logical characteristic and protocols associated, multiple routes or paths can be used to reach it [18]. When we talk about a Network address, we can think about a device location, network and route or path.

A Hardware address is the unique identifier of the network interface module of a device. This address is not used for routing of information, but to identify the device in a physical network. It means that, all devices connected to the same physical local network, can send frames to each other using their hardware address [18]. In other words, we can say that a hardware address is the address used by a device to send a frame to another device or devices that are in the same physical local network.

Regardless of the Network address and the protocols being used in the upper layers of the OSI model, the Hardware address is always necessary to reach a device in the same physical local network.

All data communication needs to map network and hardware addresses. To better understand this process; let's review the OSI model and the ARP protocol.

4.4.1.1. OSI Model and Address Resolution Protocol

Before explaining PSP Network to Hardware Address Mapping in more details, it is necessary to have a good understanding of how the Data is processed through the OSI model since its generation to its delivery. It is also necessary to understand how address mapping protocols work. For this purpose, we will explain the Address Resolution Protocol. If you already understand how the OSI model and the Address Resolution Protocol work, you can skip this section.

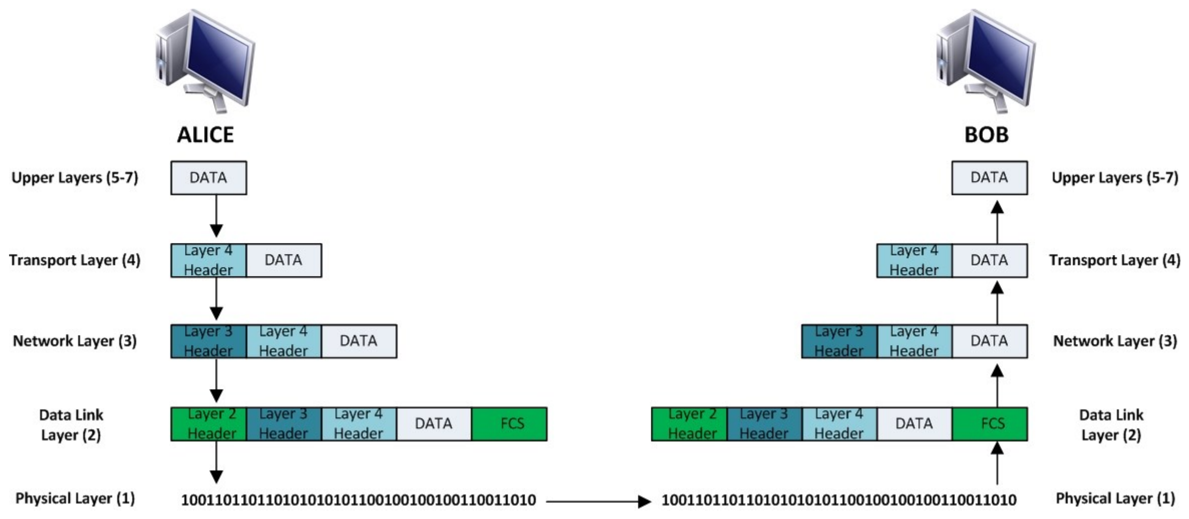


Figure 4.8. Data Processed Through the OSI Model

In the scenario shown in Figure 4.8, Alice wants to send a unicast packet to Bob – a unicast is a packet sent to one destination [19]-. This data, which is generated in the upper layers of the OSI model, is encapsulated in a transport layer that will append a header indicating the source and destination port, the transport protocol and some other parameters that are not relevant for the purpose of this explanation. This new segment is passed to the network layer and a new header is appended indicating the version of the IP protocol, the source and destination IP addresses, and some other parameters. This packet is passed to the Data link layer, which appends another header indicating the source and destination MAC addresses, and other parameters. Finally, the data is sent over to Bob as bits traveling through the media used to connect the equipment to the network (Cable, Air, Fiber, etc.) [20].

Bob receives the bits at the physical layer and removes the header of each additional layer until it gets the data that was sent.

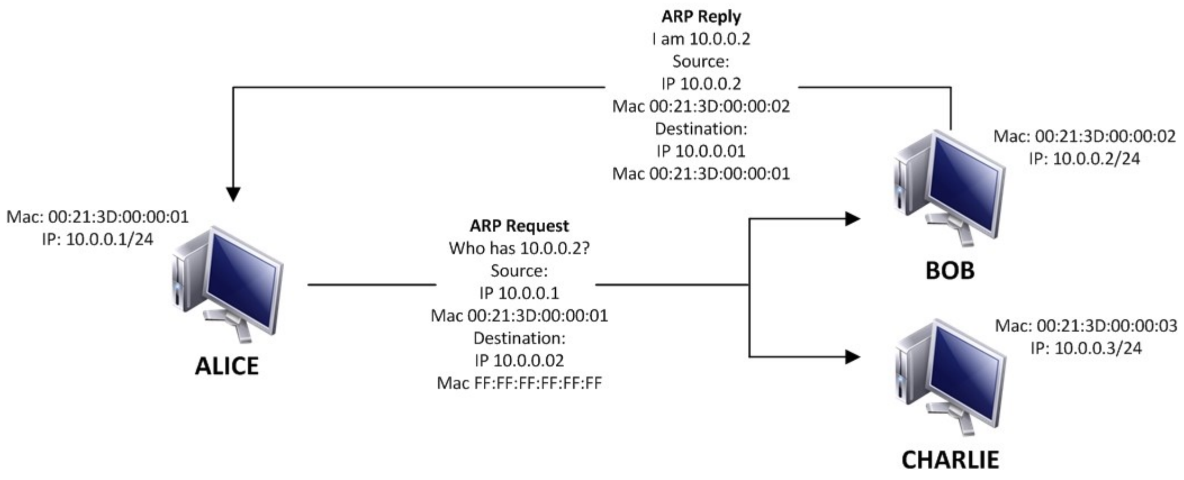


Figure 4.9. ARP Request and Reply

Alice and Bob can communicate because both hosts know the Mac and IP addresses of each other (Hardware and Network addresses). If this information is missing, it is necessary to use the Address Resolution Protocol to map both addresses [21]. In the previous example of Alice sending a unicast packet to Bob, if Alice doesn't know the Mac address, she won't know how to structure the layer 2 header. What Alice will do, as depicted in Figure 4.9, is to send a layer 2 broadcast asking "What is the Mac address of Bob" – a broadcast is a packet sent to all possible destinations at the same time [19]-. All hosts connected to the same Hub or Switch will receive the broadcast but only Bob, if he is in the same layer 3 subnet as Alice, will reply with his own Mac address".

4.4.1.2. PSP Address Mapping Explained

Now that we understand how Data is processed through the OSI model and how an address mapping protocol works, we can explain how address mapping occurs in PSP.

PSP creates an address mapping table from any PSP packet that enters the interface where the protocol is configured. The packet must belong to the same subnet.

There are multiple ways to get the source hardware and network addresses:

- We can get the source hardware and network addresses by using the ARP or NDP protocols, depending if the protected protocol is either IPv4 or IPv6.
- Manually created ARP or NDP tables.
- Incoming broadcast or multicast packets.

When a PSP packet is received, if there is not a communication session established, the device tries to decrypt it using the installed common pre-shared symmetric key. If successful, it will look into the source and destination address fields of the network protocol that is being protected. If the network address is in the same subnet, and if the destination address is the receiver's address, a multicast or broadcast address; it will be mapped with the source hardware address specified in the layer 2 header. If a device is able to decrypt a PSP packet, and it is not using digital certificate protection to authenticate the sender, it will be assumed that it was sent by a trusted source.

4.4.2. Communication Session Establishment

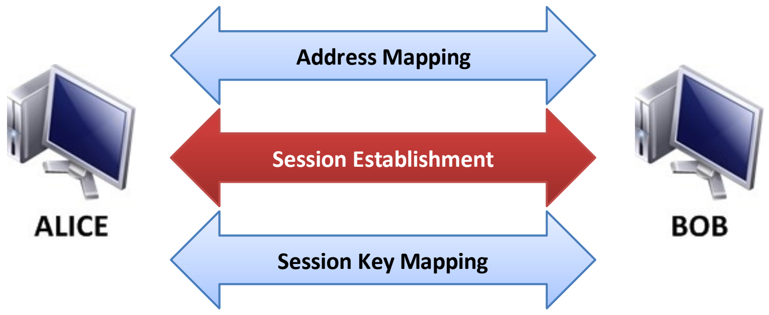


Figure 4.10. PSP Communication Process – Step 2

The second step in the PSP communication process is the establishment of a communication session that is unique to the hosts trying to communicate with each other. This step is handled by the PSP Session Establishment Protocol (PSP SEP) and occurs only if two conditions are met: 1) the initiator has mapped the network and hardware addresses of the receiver. 2) The sender wants to send a packet to the receiver.

The PSP Session Establishment Protocol (PSP SEP) is responsible for establishing a communication session between two hosts that wants to communicate with each other. The protocol negotiates the security parameters of the communication session including encryption and hashing algorithms, unique session symmetric key, Base Initialization Vector Number (BIVN), mathematical function to calculate the IV, communication session expiration time, communication session reestablishment and communication session finalization.

PSP SEP consists of 3 stages to establish a communication session: Session Establishment Request, Hash and Encryption Negotiation, and Session Parameters Negotiation.

We will retake the names "Alice" and "Bob" to represent the devices involved in a PSP communication and make this explanation more understandable.

Once Alice has mapped the network and hardware addresses of Bob, she will try to establish a PSP communication session. For this purpose, she will use the PSP Session Establishment Protocol.

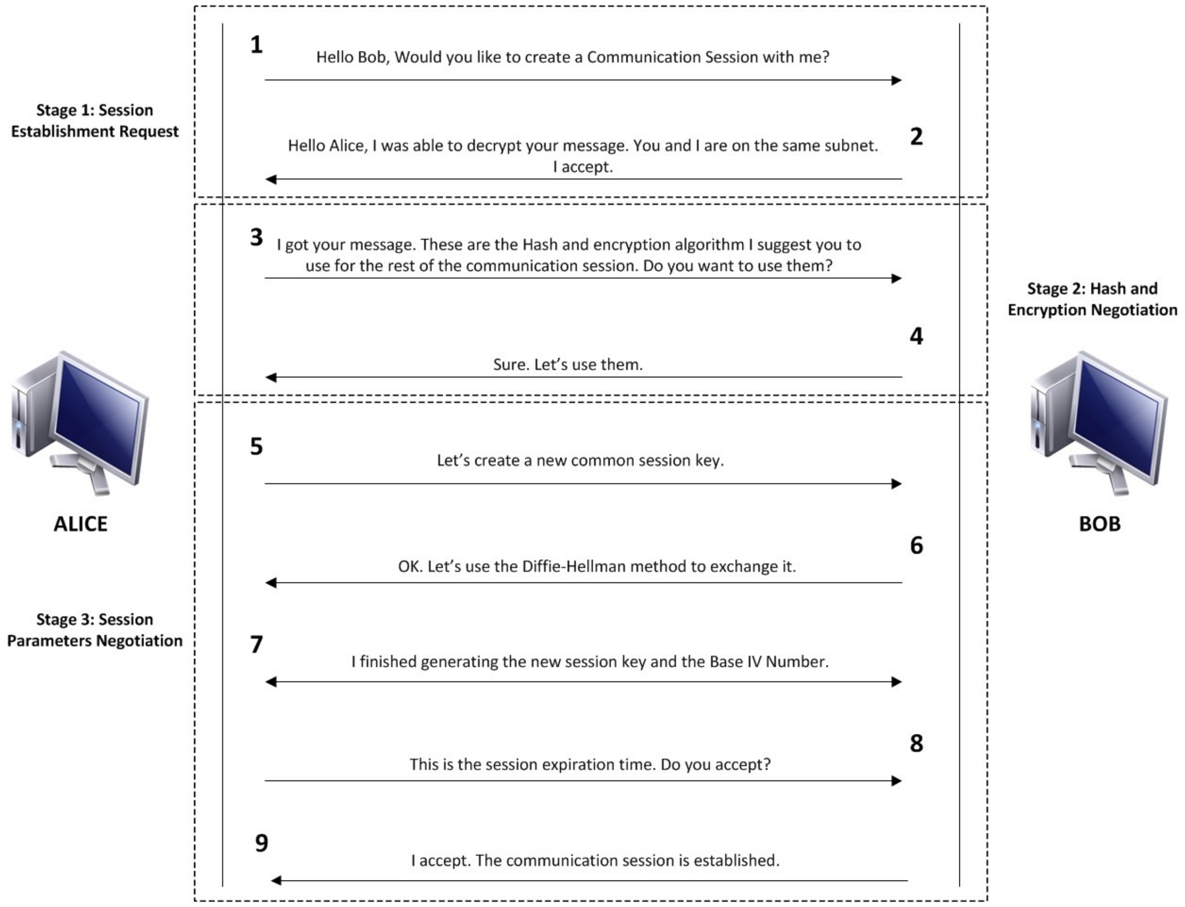

Stage 1: Session Establishment Request
1
Hello Bob, Would you like to create a Communication Session with me?
2
Hello Alice, I was able to decrypt your message. You and I are on the same subnet. I accept.
3
I got your message. These are the Hash and encryption algorithm I suggest you to use for the rest of the communication session. Do you want to use them?
Stage 2: Hash and Encryption Negotiation
4
Sure. Let's use them.
ALICE
BOB
5
Let's create a new common session key.
6
OK. Let's use the Diffie-Hellman method to exchange it.
Stage 3: Session Parameters Negotiation
7
I finished generating the new session key and the Base IV Number.
8
This is the session expiration time. Do you accept?
9
I accept. The communication session is established.

Figure 4.11. PSP Communication Session Establishment Process

Each step of the Communication Session Establishment is depicted in Figure 4.11.

Stage 1: Session Establishment Request

1. Alice will send a communication session establishment request to Bob.

2. Bob verifies that Alice's Network address is on the same subnet as his, and that the destination address is Bob's. If Alice and Bob are in different subnets, or if the destination address of the request sent by Alice is not Bob's network address, Bob will drop the request packet and won't reply to Alice. If both devices are in the same subnet, Bob will reply accepting the request.

Stage 2: Hash and Encryption Negotiation

If Bob accepts the communication session establishment request, Alice and Bob will negotiate the encryption and hashing algorithms to use from that point on.

3. Alice sends an offer of encryption and hashing algorithms to Bob.
4. Bob replies accepting or rejecting the offer. If both hosts agree upon which encryption and hashing and algorithms to use, the session establishment process continues.

Stage 3: Session Parameters Negotiation

Once both hosts have agreed upon encryption and hashing algorithms for their communication, they negotiate a session key, a mathematical function to calculate the Initialization Vector of the encryption algorithm and a session expiration time.

5. Alice requests to establish a new common session key to encrypt future communication.
6. Bob confirms the message and starts a key exchange using the Diffie-Hellman method.
7. Once both Hosts have agreed upon a new common symmetric key, they calculate the HMAC-SHA1 of the new key to generate the Base IV Number used to calculate the Initialization Vector.
8. Alice sends a session expiration time.
9. Bob replies accepting or rejecting the expiration time. From this point on, the communication session has been established. All traffic between Alice and Bob is protected using the negotiated encryption and hashing algorithms and the new symmetric session key and Base IV Number.

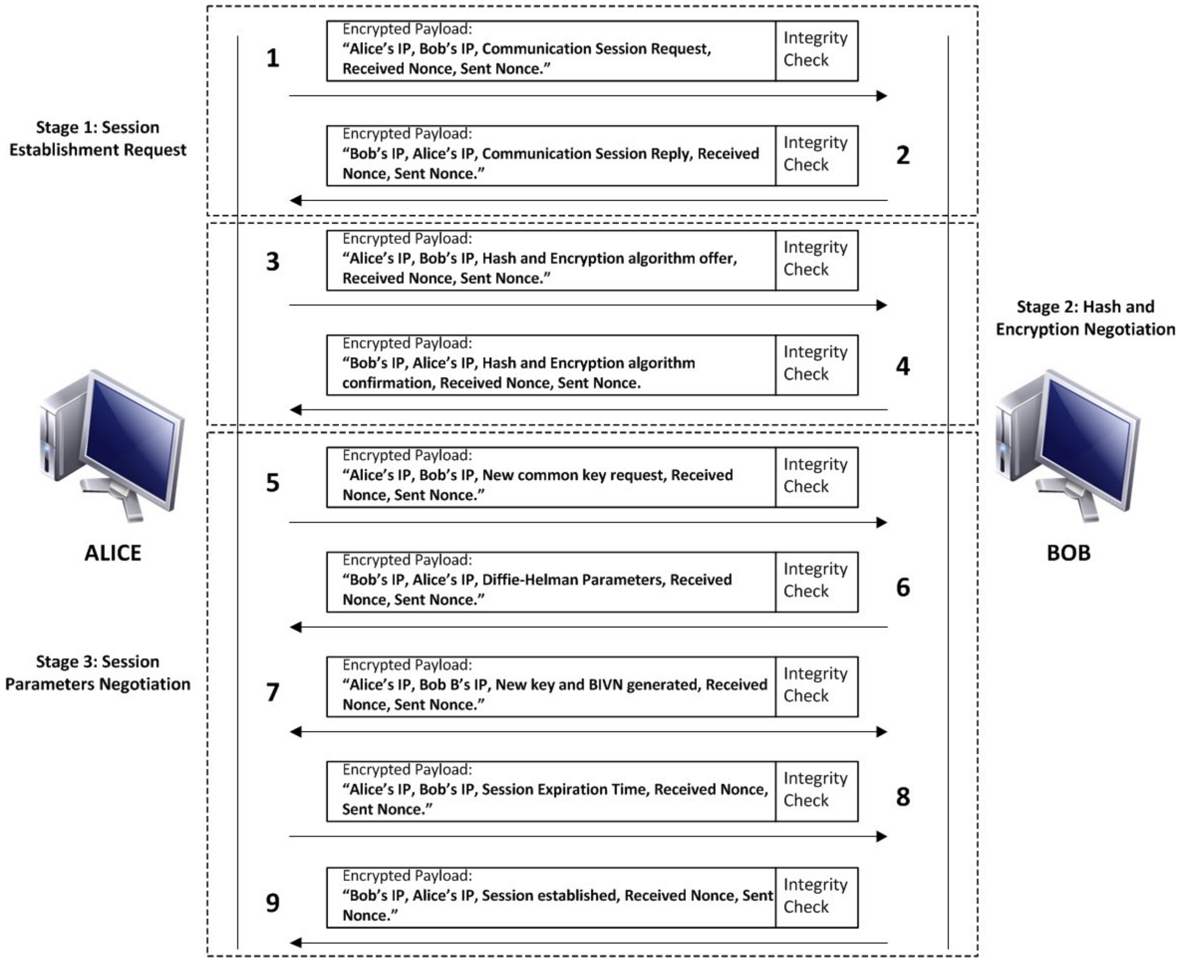

Stage 1: Session Establishment Request
1
Encrypted Payload:
"Alice's IP, Bob's IP, Communication Session Request, Received Nonce, Sent Nonce."
Integrity Check
2
Encrypted Payload:
"Bob's IP, Alice's IP, Communication Session Reply, Received Nonce, Sent Nonce."
Integrity Check
Stage 2: Hash and Encryption Negotiation
3
Encrypted Payload:
"Alice's IP, Bob's IP, Hash and Encryption algorithm offer, Received Nonce, Sent Nonce."
Integrity Check
4
Encrypted Payload:
"Bob's IP, Alice's IP, Hash and Encryption algorithm confirmation, Received Nonce, Sent Nonce.
Integrity Check
ALICE
BOB
Stage 3: Session Parameters Negotiation
5
Encrypted Payload:
"Alice's IP, Bob's IP, New common key request, Received Nonce, Sent Nonce."
Integrity Check
6
Encrypted Payload:
"Bob's IP, Alice's IP, Diffie-Helman Parameters, Received Nonce, Sent Nonce."
Integrity Check
7
Encrypted Payload:
"Alice's IP, Bob B's IP, New key and BIVN generated, Received Nonce, Sent Nonce."
Integrity Check
8
Encrypted Payload:
"Alice's IP, Bob's IP, Session Expiration Time, Received Nonce, Sent Nonce."
Integrity Check
9
Encrypted Payload:
"Bob's IP, Alice's IP, Session established, Received Nonce, Sent Nonce."
Integrity Check

Figure 4.12. PSP Communication Session Establishment Process – PDU Structure

A PDU structure perspective of the PSP communication session establishment process is depicted in Figure 4.12. We have omitted some header fields to make the image more understandable.

4.4.3. Session key Mapping

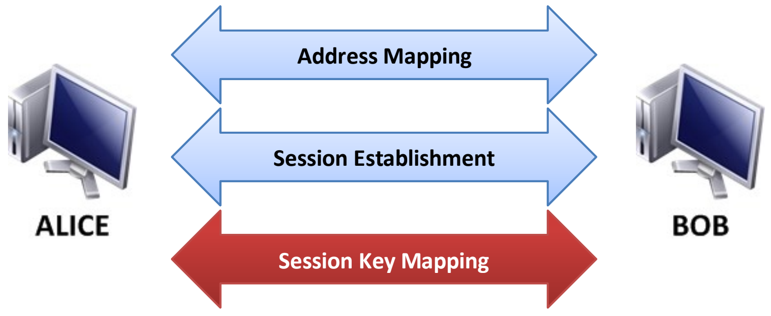


Figure 4.13. PSP Communication Process – Step 3

When two hosts establish a communication session, a unique symmetric key is generated and an encryption and hashing algorithm, along with a mathematical function to calculate the IV are negotiated to protect the communication between them. Session key mapping is the association of the session key and communication session parameters with the Hardware address of a host based on the layer 2 protocol that is used in the communication: Ethernet, HDLC, Frame Relay, PPP, etc.

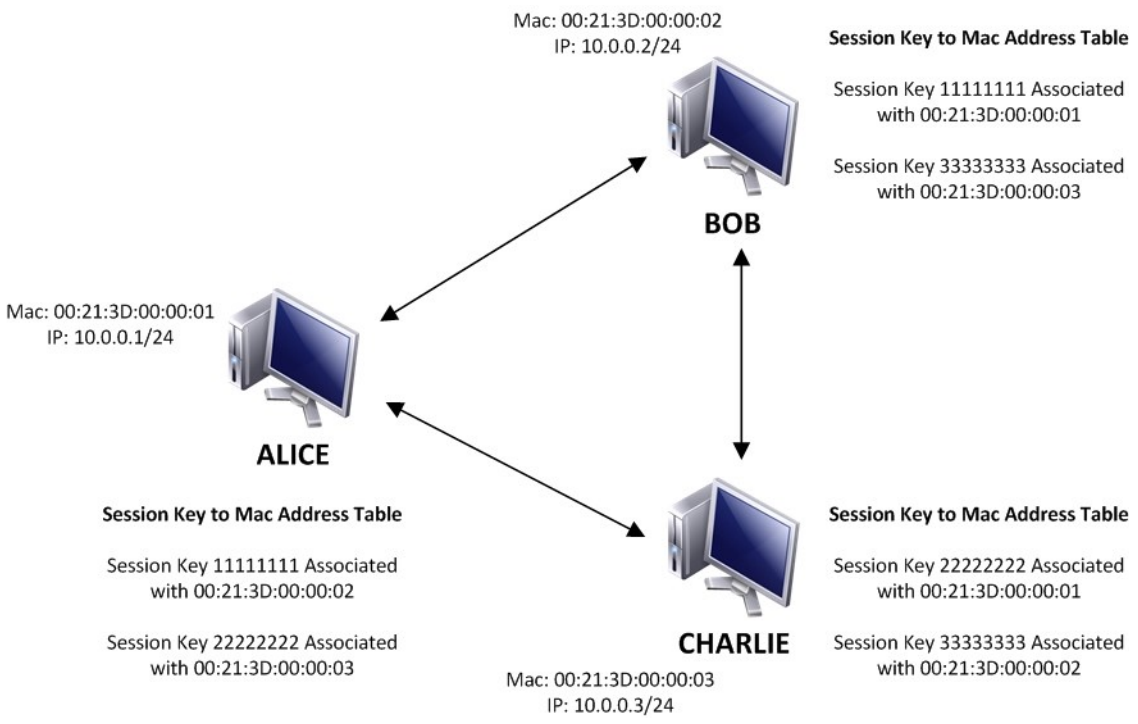

Mac: 00:21:3D:00:00:02
IP: 10.0.0.2/24
BOB
Session Key to Mac Address Table
Session Key 11111111 Associated with 00:21:3D:00:00:01
Session Key 33333333 Associated with 00:21:3D:00:00:03
Mac: 00:21:3D:00:00:01
IP: 10.0.0.1/24
ALICE
Session Key to Mac Address Table
Session Key 11111111 Associated with 00:21:3D:00:00:02
Session Key 22222222 Associated with 00:21:3D:00:00:03
CHARLIE
Mac: 00:21:3D:00:00:03
IP: 10.0.0.3/24
Session Key to Mac Address Table
Session Key 22222222 Associated with 00:21:3D:00:00:01
Session Key 33333333 Associated with 00:21:3D:00:00:02

Figure 4.14. PSP Session Key Mapping

Let's say that Alice, Bob and Charlie use Ethernet as layer 2 protocol. Figure 4.14 shows that Alice has established a communication session with Bob and a second communication session with Charlie. Every time Alice receives a packet, she will know which session key and encryption and hashing algorithms must use to decrypt the packet because of the Hardware address of the sender (in this case, the Mac address). If the packet's source mac address is Bob's, Alice will know that she will have to use the session key created with Bob and not the session key created with Charlie.

4.5. PSP Communication Process in an Ethernet Environment

This section explains the PSP communication process in an Ethernet environment. Let's use the same example shown in section 4.4.1.1 where Alice wants to send a packet to Bob. Both hosts are using IPv4 or IPv6 but this time, they want to protect the communication using PSP.

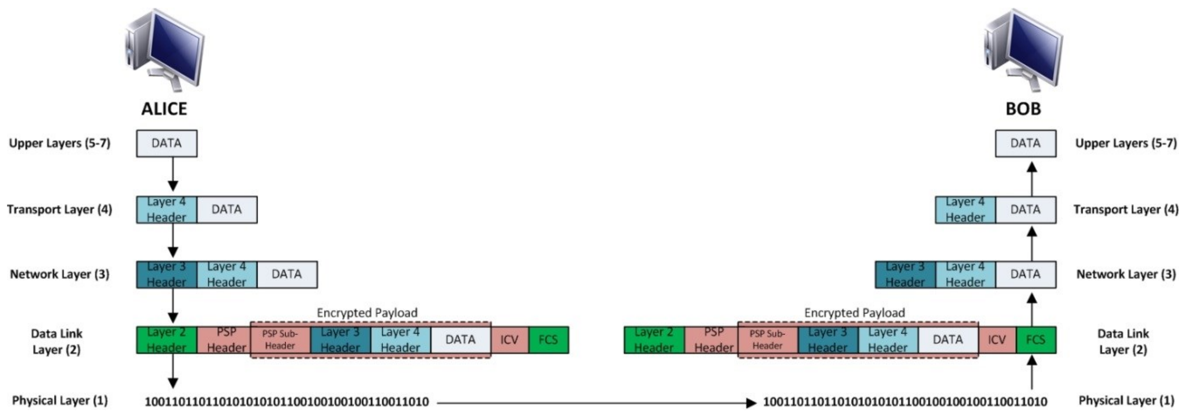


Figure 4.15. Data Processed Through the OSI Model with PSP Protection

Figure 4.15 depicts how PSP fits into the OSI layered model. The entire IPv4/6 datagram (header and payload) will be encapsulated into a PSP PDU without disclosing any kind of information about the packets being transmitted.

Before exchanging data with Bob, Alice has to complete each step of the PSP communication process: Address Mapping, Communication Session Establishment, and Session Key Mapping. We will describe what happens at each step in an Ethernet environment.

4.5.1. Address Mapping

Most of LAN networks today use Ethernet as layer 2 protocol. When a host is configured with IPv4 as Network protocol, it uses ARP to map network and hardware addresses. ARP supports the IPv4 protocol but is not part of it [22], and information about the source and destination network addresses is transmitted in clear text. To avoid disclosing network address information, all ARP packets are encapsulated into PSP.

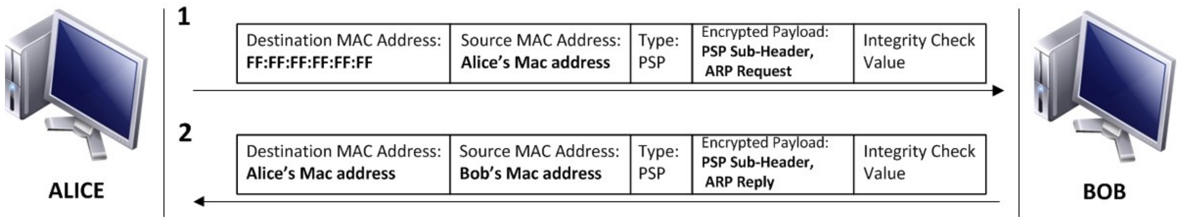

1
Destination MAC Address: FF:FF:FF:FF:FF:FF
Source MAC Address: Alice's Mac address
Type: PSP
Encrypted Payload: PSP Sub-Header, ARP Request
Integrity Check Value
2
Destination MAC Address: Alice's Mac address
Source MAC Address: Bob's Mac address
Type: PSP
Encrypted Payload: PSP Sub-Header, ARP Reply
Integrity Check Value
ALICE
BOB

Figure 4.16. ARP Over PSP

As depicted in Figure 4.16, instead of directly broadcasting an ARP request in clear text asking for the Mac address of Bob, Alice will encrypt the ARP request using the default pre-shared key and will create a layer 2 frame with the Ethertype field as a PSP packet. All hosts in the same physical segment of the network will receive the packet and check the Ethertype field in the Ethernet frame, but only hosts protected by PSP and using the same default pre-shared key as Alice will be able to decrypt it. All hosts protected by PSP will first calculate the Integrity Check Value of the frame. If the calculated value is the same as the value in the ICV field of the received frame, they will proceed to decrypt and extract the payload. By default the PSP protocol uses Blowfish and HMAC-SHA1 as encryption and hashing algorithms.

Once the packet is decrypted, all hosts will extract the ARP request and only Bob will send an ARP reply encapsulated into PSP to Alice.

With PSP, hosts can map Mac and IP addresses in a secure way without disclosing IP information.

4.5.2. Session Communication Establishment and Session Key Mapping

Once Alice has received the ARP reply from Bob, she will establish a communication session with him. For this purpose, Alice will use the PSP Session Establishment Protocol (PSP SEP), which will negotiate the security parameters of the communication session including encryption and hashing algorithms, unique session symmetric key, Base Initialization Vector Number (BIVN), mathematical function to calculate the IV and communication session expiration time. After establishing a communication session, Alice and Bob can communicate with each other.

Alice and Bob will use Session Key Mapping to identify their communication session. Every time Bob receives a frame from Alice, he will know what session key and encryption and hashing algorithms to use thanks to Alice's Mac address.

If we look at the communication process from a security perspective, it is safe from Man-in-the-Middle attacks or eavesdropping. From the beginning of the communication, which starts off with the ARP request, all messages are encrypted. The only information that is disclosed are the layer 2 and PSP headers, but the communication messages and IPv4 header are completely encrypted.

The only way to compromise the establishment of a PSP communication session between two hosts is by stealing the encryption key that the hosts use.

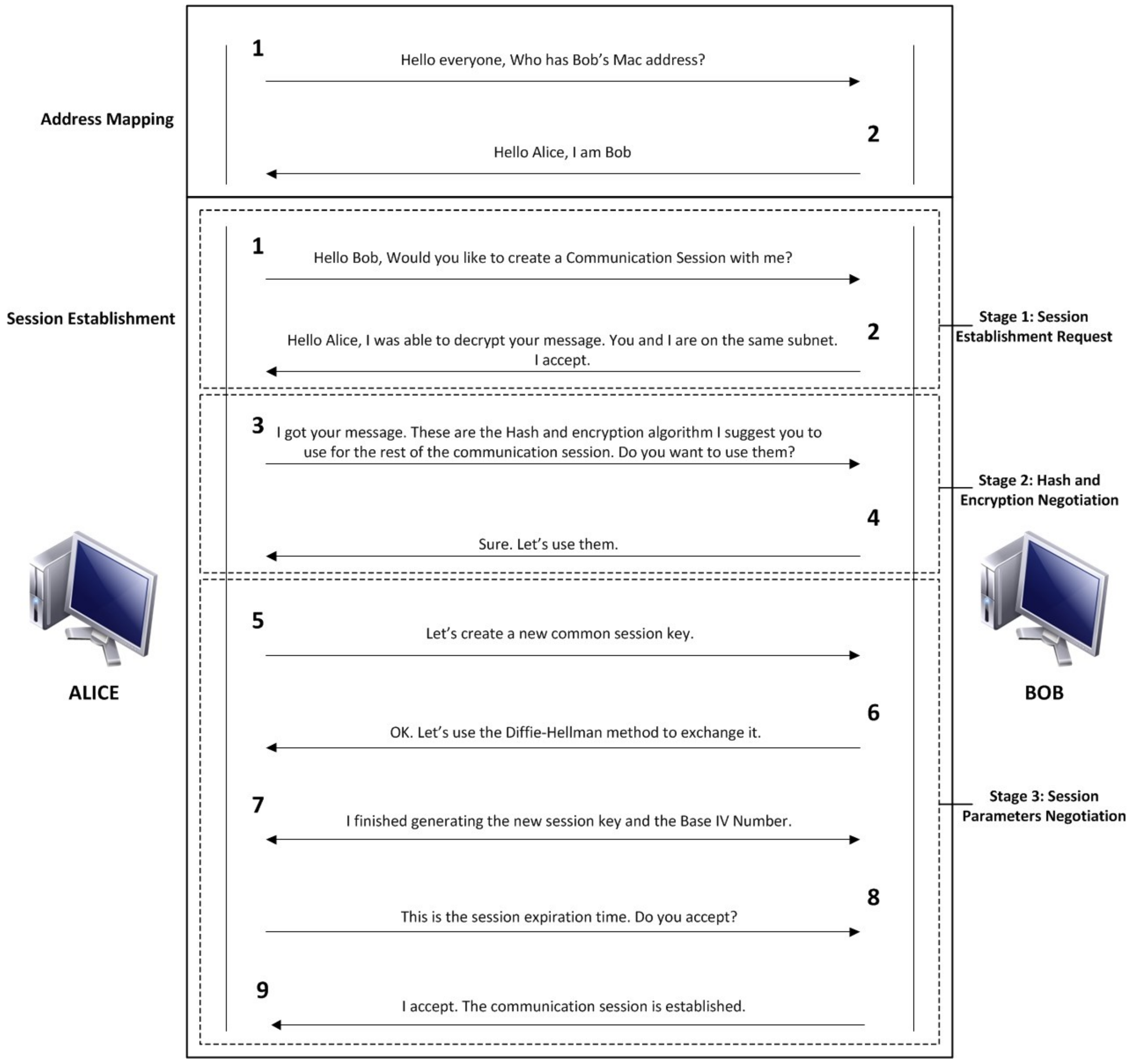

Address Mapping
1
Hello everyone, Who has Bob's Mac address?
2
Hello Alice, I am Bob
Session Establishment
1
Hello Bob, Would you like to create a Communication Session with me?
2
Hello Alice, I was able to decrypt your message. You and I are on the same subnet. I accept.
Stage 1: Session Establishment Request
3
I got your message. These are the Hash and encryption algorithm I suggest you to use for the rest of the communication session. Do you want to use them?
4
Sure. Let's use them.
Stage 2: Hash and Encryption Negotiation
ALICE
BOB
5
Let's create a new common session key.
6
OK. Let's use the Diffie-Hellman method to exchange it.
7
I finished generating the new session key and the Base IV Number.
8
This is the session expiration time. Do you accept?
9
I accept. The communication session is established.
Stage 3: Session Parameters Negotiation

Figure 4.17. Complete PSP Communication Process in an Ethernet Environment

The entire PSP communication process in an Ethernet environment is depicted in Figure 4.17.

5. IMPLEMENTATION OF THE PROPOSED SOLUTION

This section explains the methodology, scenarios, tested protocols and services, test categories and the computer program developed to demonstrate the proposed Packet Security Protocol.

5.1. Methodology and Scenario

We have developed a computer program in C++ to demonstrate the proof of concept of our proposed solution. This demonstration was based on a methodology consisting of a security and a performance analysis. First, we analyzed the security of our proposed protocol by performing different attacks to compromise the communication between two hosts, and later, we protected the same communication with our solution; we also analyzed the impact on security caused by the use of different modes of operation of a block cipher. Second, we ran multiple tests to 7 different protocols to measure the performance of the network and the performance of the hosts using our proposed solution.

The main objective of this methodology is to measure the security, performance and reliability of the Packet Security Protocol based on the following characteristics:

- Data confidentiality: Verifies that the payload and IP headers are fully encrypted and an attacker is not able to decrypt them or recognize data patterns.
- Data integrity: Verifies that the Ethernet frame is not successfully modified by an attacker.
- Network performance: Reflects the network latency, data throughput and bandwidth of the protocol.
- System performance: Reflects the CPU and memory usage on the systems running the proposed protocol.

All security and performance tests were executed in a scenario with a network topology depicted in Figure 5.1.

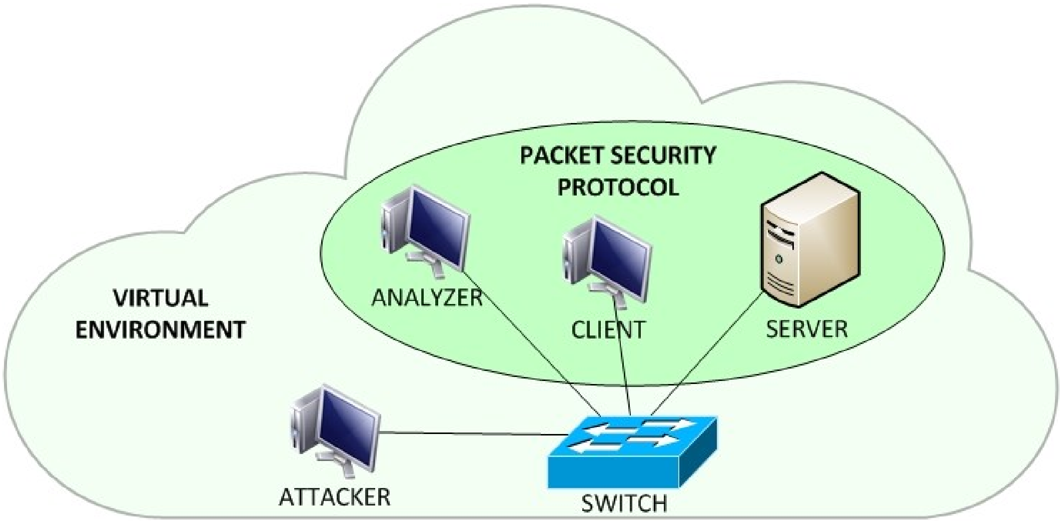


Figure 5.1. Network Topology

Three virtual machines - a network protocol analyzer capturing traffic for data analysis purposes, one client and one server all running the proposed Packet Security Protocol - are connected to a virtual switch configured in promiscuous mode. An attacker not running the proposed Packet Security Protocol is connected to the same virtual switch and is sniffing the traffic of the network.

5.2. Security Analysis

We performed an ARP spoofing and Man in the Middle attack and installed an SSH honeypot to capture user credentials of the server. The scenario is the same depicted in Figure 5.1 but without protecting the network with our protocol. We later tried to replicate the same attacks to the network depicted in Figure 5.1 protected with our proposed solution.

The second part of the security analysis evaluates the possible security implications derived from using our solution with a block cipher in Electronic Codebook mode (ECB).

5.3. Performance Analysis

A total of 7 network protocols and services were tested in 4 different categories to determine the impact on network and system performance caused by our solution. All tested protocols and services correspond to a Client-Server scenario like the one depicted in Figure 5.1, where the Client tries to reach a service provided by the Server.

5.3.1. Tested Protocols and Services

The following protocols and services were tested: ARP, ICMP, VoIP, Skype, Iperf-TCP, Iperf-UDP and SCP**.**

For Skype tests, we used a different scenario as depicted in Figure 5.2.

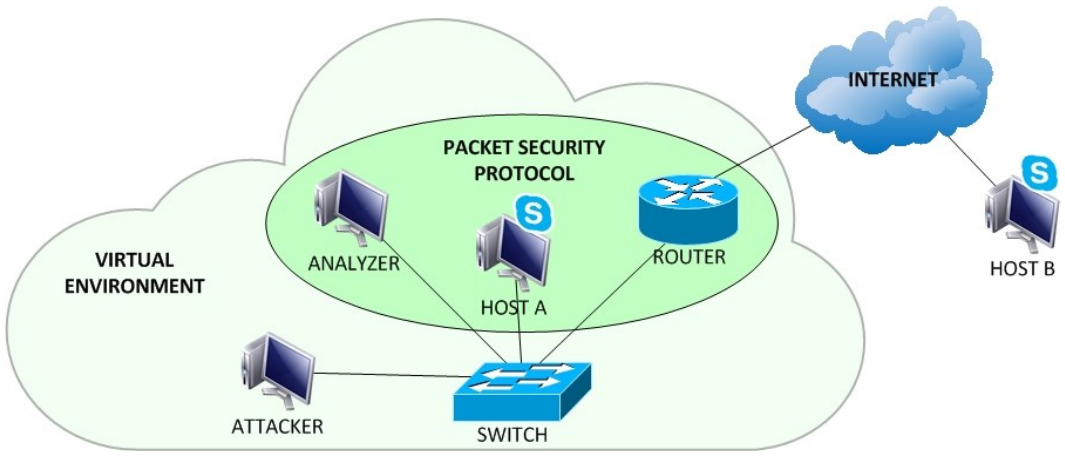


Figure 5.2. Network Topology for Skype Tests

Host A establishes a Skype video conference with Host B. Host A is running the proposed Packet Security Protocol (PSP) and accesses the internet through a Linux router also running PSP. A network protocol analyzer also running PSP is capturing traffic for data analysis purposes. An Attacker is connected to the same virtual switch as the Analyzer, Host A and the Linux Router, and it is sniffing the traffic. The virtual switch is configured in promiscuous mode.

5.3.2. Test Categories

The following test categories were used in this test methodology:

1- Ethernet without PSP applied. TOE disabled

The purpose of this category is to establish a baseline and a reference for all tests. Network protocols and services tested in other categories were compared to this one and reflected the network and system performance variation of the proposed Packet Security Protocol.

In this category, all tests were conducted in a regular scenario where data is sent over the network with no encryption or manipulation of the payload, layer 3 or layer 2 headers.

The network adapter of the Client and the Server had the TCP Offloading Engine (TOE), a feature of some network adapters to offload the TCP segments to the network adapter and improve cpu usage [23], disabled to mimic a regular network card. This is the most common scenario in today's network environments.

Protocols and services tested under this category: ARP, ICMP, VoIP, Skype, Iperf-TCP, Iperf-UDP, SCP.

2- Ethernet without PSP applied. TOE enabled

In this category, all tests were conducted in a regular scenario where data is sent over the network with no encryption or manipulation of the payload, layer 3 or layer 2 headers. The network adapter had the TCP Offloading Engine (TOE) enabled.

TOE is an optional feature included in some Network adapters used to free up the CPU from processing the TCP segments and add significant performance improvements. The TCP Offload Engine has a complete TCP Stack to handle all TCP transport and IP addressing functions. [23]

The purpose of this category is to measure the impact of enabling vs disabling TOE on TCP related tests and how the advantages of enabling TOE are compromised by the proposed Packet Security Protocol. This includes CPU and bandwidth usage.

Protocols and services tested under this category: Iperf-TCP, SCP.

3- PSP-NoENC-NoHMAC

This category measures the impact on network and system performance due to using Packet Security Protocol without encryption and HMAC calculation. Even though, the proposed protocol is not meant to be used without encryption and HMAC calculation, this category shows the processing overhead caused by the program developed to prove the concept of Packet Security Protocol. The specific details of this program are explained in section 5.5

Protocols and services tested under this category: ARP, ICMP, VoIP, Skype, Iperf-TCP, Iperf-UDP, SCP.

4- PSP Full

This category measures the real performance of Packet Security Protocol, just as proposed in this thesis, with encryption and HMAC capabilities. The results of the tests run on this category show the real processing overhead added by the proposed protocol and the computer program developed to prove the concept of the protocol. The specific details of this program are explained in section 5.5

Protocols and services tested under this category: ARP, ICMP, VoIP, Skype, Iperf-TCP, Iperf-UDP, SCP.

Table 5.1. Protocols and Services Tested In Each Category

		CATEGORY			
		1-Ethernet without PSP applied. TOE disabled	2-Ethernet without PSP applied. TOE enabled	3-PSP-NoENC-NoHMAC	4-PSP Full
TEST	ARP	X		X	X
	ICMP	X		X	X
	Iperf-TCP	X	X	X	X
	Iperf-UDP	X		X	X
	SCP	X	X	X	X
	VoIP	X		X	X
	Skype	X		X	X

Table 5.1 shows which protocols and services were tested in each category.

5.4. Hardware and Software Specifications

All tests were conducted on a virtualized environment. The following hardware and software was used:

Physical Machines Hosting the Virtual Machines

Due to the nature of some tests, we had to use two physical machines to host the virtual environment.

- Cisco UCS-C220-M3 Server
  - 2 x Intel(R) Xeon(R) CPU E5-2650 @ 2.0 Ghz
  - 32 GB Ram
  - Raid 1 with two 10,000 RPM 300 GB Seagate ST9300605SS SAS HDD

  - Integrated dual-port Gigabit Ethernet
  - VMware ESXi
- Laptop HP Pavilion dv6-3236nr
  - 1 x Intel(R) Core(TM) i3 CPU M370 @ 2.40 Ghz
  - 4 GB Ram
  - 128 MB OCZ Vertex3 SSD
  - Integrated 1 port Gigabit Ethernet
  - Windows 7 Professional x64

Virtual Environment

- VMware ESXi 5.1.
  - Used to test the following protocols and services: ARP, ICMP, Iperf-TCP, Iperf-UDP and SCP.
  - Hosted on Cisco UCS-C220-M3 Server.
- VMware Workstation 9.0.
  - Used to test the following protocols and services: VoIP and Skype.
  - Hosted on Laptop HP Pavilion dv6-3236nr.

Virtual Machines

Each virtual environment had 4 virtual machines with the following specifications:

- Virtual Environment VMware ESXi 5.1.
  - 2 x vCPU Intel(R) Xeon(R) CPU E5-2650 @ 2.0 Ghz
  - 4 GB Ram
  - 1 x Gigabit Ethernet Interface
  - Ubuntu Linux Desktop 12.04.2 LTS x64 Kernel 3.5.0-23-generic
- Virtual Environment VMware Workstation 9.0
  - 1 x vCPU Intel(R) Core(TM) i3 CPU M370 @ 2.40 Ghz
  - 4 GB Ram
  - 1 x Gigabit Ethernet Interface
  - Ubuntu Linux Desktop 12.04.2 LTS x64 Kernel 3.5.0-23-generic

Network Protocol

- IPv4

Programs

- Wireshark v1.6.7 for Linux: Wireshark is a cross-platform network protocol analyzer used to capture and analyze network traffic [24].
- Iperf v2.0.5 for Linux: Iperf is a network performance testing tool used to measure TCP and UDP throughput [25].
- SSH-2.0-OpenSSH_5.9p1: OpenSSH is a suit of computer programs that use the SSH protocol to secure data communication [26].
- Asterisk v11.3.0: Asterisk is a voice over IP telephony platform [27].
- SIPp v3.2: SIPp is a tool used to test the SIP protocol [28].
- Skype v4.1 for Linux: Skype is a real-time voice, video and text communication application [29].
- Ettercap NG-0.7.4.2: Tool used to perform ARP spoofing and Man in the Middle Attacks [30].
- Kippo-0.8: Kippo is an SSH honeypot that logs the user activity in an SSH session. [31].

5.5. Developed Program

We developed a program in C++ to test the concept of the proposed Packet Security Protocol (PSP) on Ubuntu Linux using the following characteristics:

- Pre-shared Key
- Block Cipher Mode of Operation: Cipher-Block Chaining
- Encryption Algorithm: Blowfish
- Hash Algorithm: SHA-1

The test program doesn't execute the PSP's Session Establishment Protocol (PSP SEP). Instead, it assumes that the session has already been established and encrypts the data using a defined static pre-shared key. It uses Blowfish in Cipher-Block Chaining

(CBC) mode as the encryption algorithm and calculates the HMAC value of the frame using the SHA-1 function.

Figure 5.3 shows a regular data transmission process in a Linux computer system. Data is generated by an application in the user space. The application passes the data to the kernel, where it handles the transport layer segmentation process, the network layer routing process and the data link layer framing process. The frames are then passed by the kernel to a network card of the computer for their transmission in a medium [32].

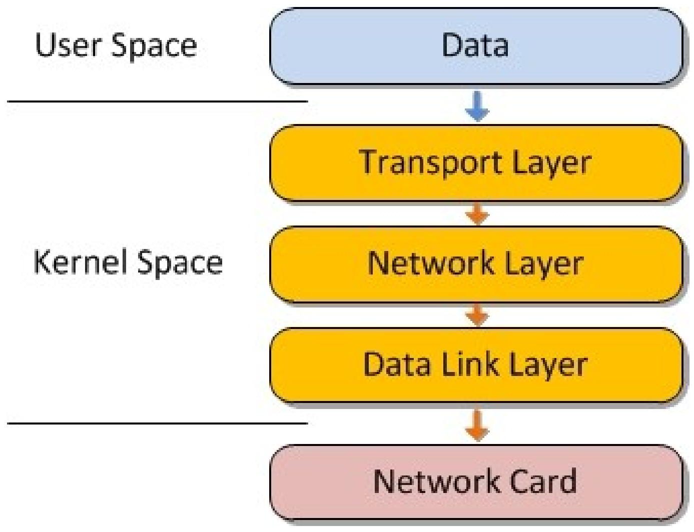


Figure 5.3. Regular Data Transmission Process

The developed program acts in line with the regular data transmission process. As depicted in Figure 5.4, once the kernel has processed all transport, network and data link layer requirements, instead of passing the data frames directly to the network card; these are passed to the developed program and protected by the Packet Security Protocol. The resulting frame is then passed by the developed program to the network card.

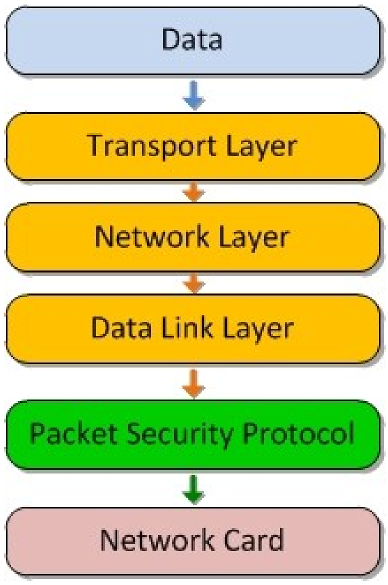


Figure 5.4. Data Transmission Process Using PSP

In a Linux environment, the virtual address space - a technique used to isolate a process in a unique address space - is divided in user space and kernel space. [33] User space is a reserved memory region used only for applications or user processes, and kernel space is the memory region reserved only for the kernel of the operating system [33]. An application designed to run in user space tend to have slower performance than if it were developed to run in kernel space [34].

The way we developed the program doesn't optimize the performance of the protocol. Data is created in the user space and then is passed to the kernel space. The program then adds a processing overhead by running in the user space and protecting the data with the Packet Security Protocol.

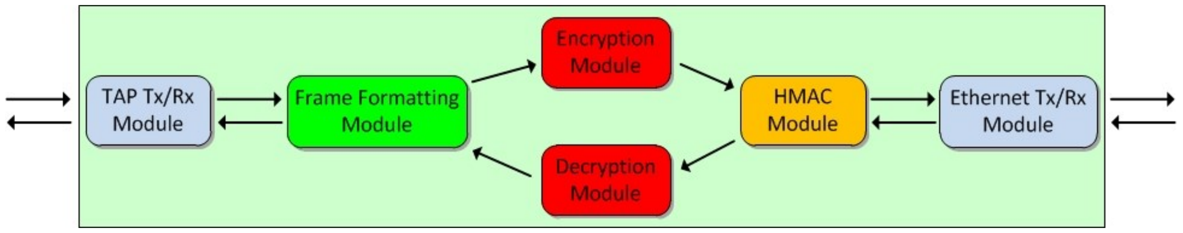


Figure 5.5. Block Diagram of the PSP's Developed Program

Figure 5.5 depicts a block diagram of the internal operation of the PSP's developed program.

Before going any further, it's necessary to explain what a TUN/TAP driver and a TUN/TAP interface is.

The universal TUN/TAP driver allows packets and frames to be received by and delivered to a TUN/TAP interface, which is a virtual interface that only exists in the kernel of the operating system. One of the advantages of this kind of interfaces is that they can be attached to a user space program, and as a result, packets and frames can be easily manipulated. TAP interfaces are used for Ethernet frames and TUN interfaces are used for IP packets. [35]

In the specific case of our developed program, from a data transmission perspective, the program uses the TUN/TAP driver to send frames to the TAP interface. As explained before, this interface only exists in the kernel and data can be easily received and manipulated by a program. Once a frame is received in the TAP interface by the TAP Tx/Rx Module, it is passed to the Frame Formatting Module to extract the payload, add the PSP header and create a new frame with an Ether_Type field equals to 0x1982. Once the new frame is created, it is passed to the Encryption Module where only the network packet (header and payload) is encrypted. The new frame is then analyzed and gets its keyed hash message authentication code (HMAC) calculated by the HMAC Module. The fully constructed data frame is then passed to the Ethernet Tx/Rx Module where it is sent to the network card.

From a Data reception perspective, the network card receives the PSP protected data frames that are sent over the network. The Ethernet Tx/Rx Module detects these frames incoming from the network card and passes them to the HMAC Module where they get their keyed hash message authentication code calculated and compared with the value attached to the HMAC field of the PSP protected frame. If both HMAC values are the same, the frame is passed to the Decryption Module where the network packet (header and payload) is decrypted. The frame with the decrypted packet is passed to the Frame Formatting Module where the PSP header information is removed and the Ether_Type field information is replaced by the original Ether_Type value of the frame. The fully recovered frame is then passed to the TAP Tx/Rx Module where it is sent through the TAP interface to the Kernel.

For the development of the program, we used the Crypto++ library for the encryption, decryption and HMAC functions used in the respective modules.

In section 5.3.2, we mentioned a test category of the Packet Security Protocol with no Encryption and no HMAC. For that purpose, we modified the developed program and disabled the Encryption, Decryption and HMAC modules, resulting in a block diagram as shown in figure 5.6.

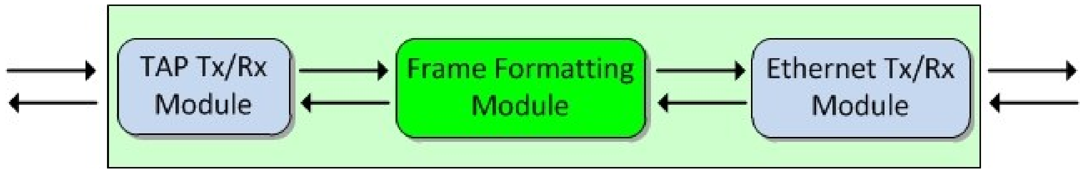


Figure 5.6. Block Diagram of the PSP's Developed Program without the Encryption, Decryption and HMAC Modules

6. TEST RESULTS

In this section we present a security and a performance analysis of the proposed Packet Security Protocol (PSP) based on the results obtained from the different tests we ran to our developed program. The security analysis demonstrates the efficiency of our solution at protecting the IP packets, while the performance analysis shows how the performance of the network and the host running the protocol is impacted.

6.1. Security Analysis

This security analysis explores if known network attacks can be avoided using our solution. It also examines the possible security implications derived from using our solution with a block cipher in Electronic Codebook mode (ECB).

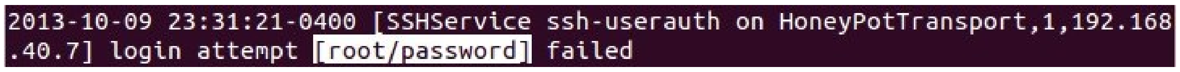
6.1.1. ARP Spoofing and Man in the Middle Attack

We performed an ARP spoofing and Man in the Middle attack to route the traffic of the affected hosts to an SSH honeypot installed in our attacking station and capture user credentials of the real server. The network topology is depicted in Figure 5.1, section 5.1. This attack was performed against an unprotected network to prove how vulnerable a wired Ethernet network can be; we later performed the same attack to the network protected with our solution to test if the attack can be avoided.

6.1.1.1. Anatomy and Results of the Attacks

First, we ran WireShark to sniff the network and discover the IP addressing scheme based on ARP broadcast messages. Next, we started Kippo, an SSH honeypot, in our attacking station. With the SSH honeypot up and running, we used Ettercap to sniff the network and perform an ARP poisoning and Man in the Middle attack against the discovered machines which, in this case, are a client and an SSH server.

After performing the attacks, we were able to manipulate at will all traffic between the affected machines. We redirected all SSH traffic from the client host to our SSH honeypot and started to log the user's login attempts on what he thought was the real SSH server.



```
2013-10-09 23:31:21-0400 [SSHService ssh-userauth on HoneyPotTransport,1,192.168
.40.7] login attempt [root/password] failed
```

Figure 6.1. User Credentials Captured in our SSH Honeypot

The user credentials were successfully captured, as shown in Figure 6.1.

6.1.1.2. Protecting the Network with Our Solution

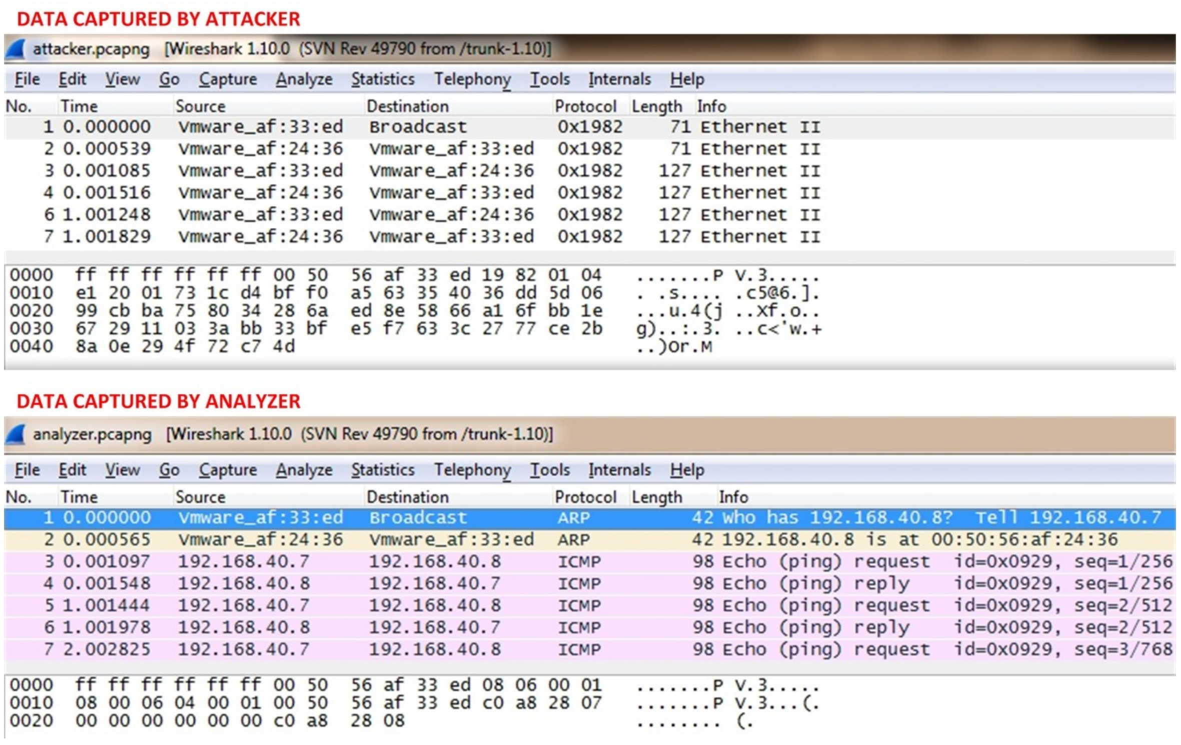
We protected the network with our solution and tried to replicate the same ARP poisoning and Man in the Middle attacks. The frames captured by WireShark from our attacking station didn’t disclose any useful information that would allow us to detect the encapsulated protocols or the IP addressing scheme of the network. The client and the server machine, when running our proposed protocol, only understood traffic that was protected with our solution and not regular traffic in clear text.

It was impossible for us to replicate the attacks on the hosts protected with our Packet Security Protocol.

6.1.2. Vulnerabilities Derived From a Block Cipher in Electronic Codebook Mode

For this study, we captured an ARP-ICMP communication between a client and a server protected with our proposed protocol. It is important to note that this analysis doesn’t include ways to decrypt or tamper the communication session between both hosts, but it examines the structure of the captured frames and presents how a block cipher in ECB mode can affect the effectiveness of our protocol and make the protected data vulnerable.

The network topology of our test scenario is depicted in Figure 5.1, section 5.1. A computer using the proposed Packet Security Protocol was capturing all communication between the client and the server for analysis purposes. We also installed another computer acting as an attacker capturing all communication. For this test, we setup the virtual switch in promiscuous mode, so it can act as a hub and replicate the frames on all ports.


DATA CAPTURED BY ATTACKER

attacker.pcapng [Wireshark 1.10.0 (SVN Rev 49790 from /trunk-1.10)]

File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help

```
No.  Time      Source              Destination         Protocol  Length  Info
   1 0.000000  Vmware_af:33:ed     Broadcast           0x1982        71  Ethernet II
   2 0.000539  Vmware_af:24:36     Vmware_af:33:ed     0x1982        71  Ethernet II
   3 0.001085  Vmware_af:33:ed     Vmware_af:24:36     0x1982       127  Ethernet II
   4 0.001516  Vmware_af:24:36     Vmware_af:33:ed     0x1982       127  Ethernet II
   6 1.001248  Vmware_af:33:ed     Vmware_af:24:36     0x1982       127  Ethernet II
   7 1.001829  Vmware_af:24:36     Vmware_af:33:ed     0x1982       127  Ethernet II

0000  ff ff ff ff ff ff 00 50  56 af 33 ed 19 82 01 04   .......P V.3.....
0010  e1 20 01 73 1c d4 bf f0  a5 63 35 40 36 dd 5d 06   . .s.... .c5@6.].
0020  99 cb ba 75 80 34 28 6a  ed 8e 58 66 a1 6f bb 1e   ...u.4(j ..Xf.o..
0030  67 29 11 03 3a bb 33 bf  e5 f7 63 3c 27 77 ce 2b   g)..:.3. ..c<'w.+
0040  8a 0e 29 4f 72 c7 4d                               ..)Or.M
```

DATA CAPTURED BY ANALYZER

analyzer.pcapng [Wireshark 1.10.0 (SVN Rev 49790 from /trunk-1.10)]

File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help

```
No.  Time      Source              Destination         Protocol  Length  Info
   1 0.000000  Vmware_af:33:ed     Broadcast           ARP           42  Who has 192.168.40.8?  Tell 192.168.40.7
   2 0.000565  Vmware_af:24:36     Vmware_af:33:ed     ARP           42  192.168.40.8 is at 00:50:56:af:24:36
   3 0.001097  192.168.40.7        192.168.40.8        ICMP          98  Echo (ping) request  id=0x0929, seq=1/256
   4 0.001548  192.168.40.8        192.168.40.7        ICMP          98  Echo (ping) reply    id=0x0929, seq=1/256
   5 1.001444  192.168.40.7        192.168.40.8        ICMP          98  Echo (ping) request  id=0x0929, seq=2/512
   6 1.001978  192.168.40.8        192.168.40.7        ICMP          98  Echo (ping) reply    id=0x0929, seq=2/512
   7 2.002825  192.168.40.7        192.168.40.8        ICMP          98  Echo (ping) request  id=0x0929, seq=3/768

0000  ff ff ff ff ff ff 00 50  56 af 33 ed 08 06 00 01   .......P V.3.....
0010  08 00 06 04 00 01 00 50  56 af 33 ed c0 a8 28 07   .......P V.3...(.
0020  00 00 00 00 00 00 c0 a8  28 08                     ........ (.
```

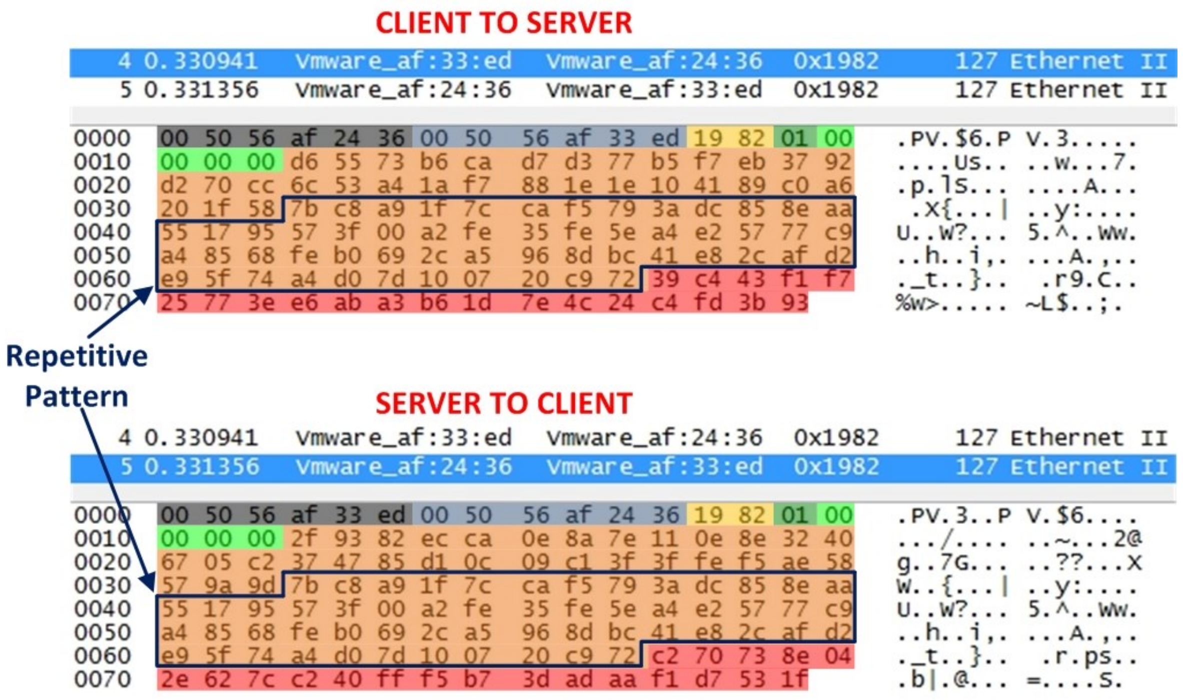
Figure 6.2. Data Captured by Attacker Vs. Data Captured by Analyzer

Part of the traffic captured by the attacker and the analyzer is shown in Figure 6.2. As we can be observe, traffic captured by the attacker was encrypted by our Proposed Packet Security Protocol; however, the same data was captured by the analyzer, and because it was able to decrypt the information, we see that there was an ARP and ICMP communication between two hosts.

The most important goal of the proposed Packet Security Protocol is to preserve the confidentiality and integrity of the data that is being sent, including all network layer information. Based on the length of the frame, if it is repetitive, the attacker can get an idea of the protocol that is being used; moreover, based on the kind of encryption algorithm used to protect the confidentiality of the information the attacker could exploit a vulnerability of this encryption algorithm and decrypt the data.

			Packet Security Protocol			
Destination MAC Address	Source MAC Address	Ethernet Type	Version + Security Options	Sequence Number	Encrypted Payload	Integrity Check Value

Figure 6.3. Structure of a Data Frame with Packet Security Protocol

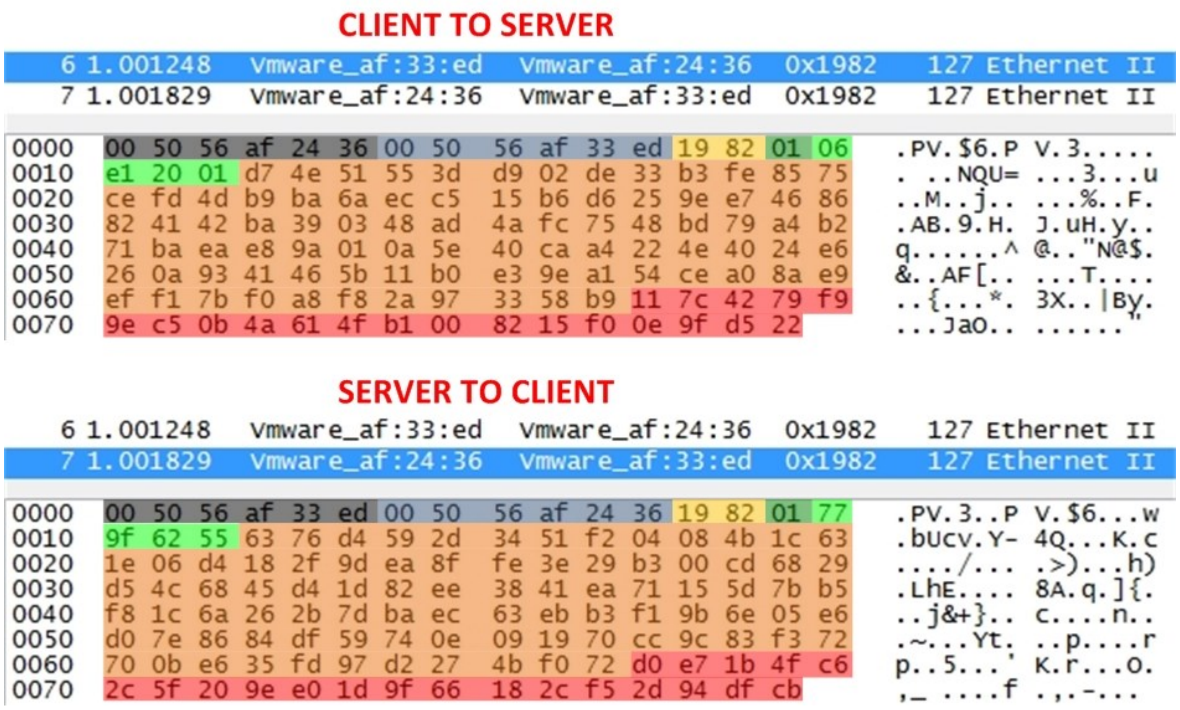


```
  4 0.330941   Vmware_af:33:ed   Vmware_af:24:36   0x1982      127 Ethernet II
  5 0.331356   Vmware_af:24:36   Vmware_af:33:ed   0x1982      127 Ethernet II

0000  00 50 56 af 24 36 00 50  56 af 33 ed 19 82 01 00   .PV.$6.P V.3.....
0010  00 00 00 d6 55 73 b6 ca  d7 d3 77 b5 f7 eb 37 92   ....Us.. ..w...7.
0020  d2 70 cc 6c 53 a4 1a f7  88 1e 1e 10 41 89 c0 a6   .p.ls... ....A...
0030  20 1f 58 7b c8 a9 1f 7c  ca f5 79 3a dc 85 8e aa    .X{...| ..y:....
0040  55 17 95 57 3f 00 a2 fe  35 fe 5e a4 e2 57 77 c9   U..W?... 5.^..Ww.
0050  a4 85 68 fe b0 69 2c a5  96 8d bc 41 e8 2c af d2   ..h..i,. ...A.,..
0060  e9 5f 74 a4 d0 7d 10 07  20 c9 72 39 c4 43 f1 f7   ._t..}..  .r9.C..
0070  25 77 3e e6 ab a3 b6 1d  7e 4c 24 c4 fd 3b 93      %w>..... ~L$..;.
```

```
  4 0.330941   Vmware_af:33:ed   Vmware_af:24:36   0x1982      127 Ethernet II
  5 0.331356   Vmware_af:24:36   Vmware_af:33:ed   0x1982      127 Ethernet II

0000  00 50 56 af 33 ed 00 50  56 af 24 36 19 82 01 00   .PV.3..P V.$6....
0010  00 00 00 2f 93 82 ec ca  0e 8a 7e 11 0e 8e 32 40   .../.... ..~...2@
0020  67 05 c2 37 47 85 d1 0c  09 c1 3f 3f fe f5 ae 58   g..7G... ..??...X
0030  57 9a 9d 7b c8 a9 1f 7c  ca f5 79 3a dc 85 8e aa   W..{...| ..y:....
0040  55 17 95 57 3f 00 a2 fe  35 fe 5e a4 e2 57 77 c9   U..W?... 5.^..Ww.
0050  a4 85 68 fe b0 69 2c a5  96 8d bc 41 e8 2c af d2   ..h..i,. ...A.,..
0060  e9 5f 74 a4 d0 7d 10 07  20 c9 72 c2 70 73 8e 04   ._t..}..  .r.ps..
0070  2e 62 7c c2 40 ff f5 b7  3d ad aa f1 d7 53 1f      .b|.@... =....S.
```

Figure 6.4. Data Encrypted Using a Block Cipher in Electronic Codebook Mode

When a block cipher in Electronic Codebook Mode (ECB) is used, patterns of encrypted data will be created and an attacker could extract more information from the captured frames. We used the PSP structure shown in Figure 6.3 to color code the rest of the figures and explain in better details this idea. The frames captured by the attacker and depicted in Figure 6.4 were protected with PSP, but data was encrypted using a block cipher in Electronic Codebook Mode. We can see that there is a repetitive pattern of 56 bytes in the encrypted section of the client to server and server to client frames. The attacker could take this pattern in his advantage and, employing brute force or other mechanisms, could decrypt the data. This is not a vulnerability of the proposed Packet Security Protocol, but a condition of the encryption algorithm that could result in an advantage for the attacker.



```
6 1.001248   Vmware_af:33:ed   Vmware_af:24:36   0x1982   127 Ethernet II
7 1.001829   Vmware_af:24:36   Vmware_af:33:ed   0x1982   127 Ethernet II

0000  00 50 56 af 24 36 00 50  56 af 33 ed 19 82 01 06   .PV.$6.P V.3.....
0010  e1 20 01 d7 4e 51 55 3d  d9 02 de 33 b3 fe 85 75   . ..NQU= ...3...u
0020  ce fd 4d b9 ba 6a ec c5  15 b6 d6 25 9e e7 46 86   ..M..j.. ...%..F.
0030  82 41 42 ba 39 03 48 ad  4a fc 75 48 bd 79 a4 b2   .AB.9.H. J.uH.y..
0040  71 ba ea e8 9a 01 0a 5e  40 ca a4 22 4e 40 24 e6   q......^ @.."N@$.
0050  26 0a 93 41 46 5b 11 b0  e3 9e a1 54 ce a0 8a e9   &..AF[.. ...T....
0060  ef f1 7b f0 a8 f8 2a 97  33 58 b9 11 7c 42 79 f9   ..{...*. 3X..|By.
0070  9e c5 0b 4a 61 4f b1 00  82 15 f0 0e 9f d5 22      ...JaO.. ......"
```

```
6 1.001248   Vmware_af:33:ed   Vmware_af:24:36   0x1982   127 Ethernet II
7 1.001829   Vmware_af:24:36   Vmware_af:33:ed   0x1982   127 Ethernet II

0000  00 50 56 af 33 ed 00 50  56 af 24 36 19 82 01 77   .PV.3..P V.$6...w
0010  9f 62 55 63 76 d4 59 2d  34 51 f2 04 08 4b 1c 63   .bUcv.Y- 4Q...K.c
0020  1e 06 d4 18 2f 9d ea 8f  fe 3e 29 b3 00 cd 68 29   ..../... .>)...h)
0030  d5 4c 68 45 d4 1d 82 ee  38 41 ea 71 15 5d 7b b5   .LhE.... 8A.q.]{.
0040  f8 1c 6a 26 2b 7d ba ec  63 eb b3 f1 9b 6e 05 e6   ..j&+}.. c....n..
0050  d0 7e 86 84 df 59 74 0e  09 19 70 cc 9c 83 f3 72   .~...Yt. ..p....r
0060  70 0b e6 35 fd 97 d2 27  4b f0 72 d0 e7 1b 4f c6   p..5...' K.r...O.
0070  2c 5f 20 9e e0 1d 9f 66  18 2c f5 2d 94 df cb      ,_ ....f .,.-...
```

Figure 6.5. Data Encrypted Using a Block Cipher in Cipher-Block Chaining Mode (CBC)

By default, Packet Security Protocol uses Blowfish in Cipher-Block Chaining mode (CBC) as the encryption algorithm. As depicted in Figure 6.5 and using the same PSP structure in Figure 6.3, this encryption mode doesn’t create any patterns of encrypted information because each block depends on the previous one to be encrypted [36]; however, it is vulnerable to multiple initialization vector attacks that could allow an attacker to modify the IV and get the decrypted information, as it has been demonstrated with the IPsec protocol [37]; PSP is not affected by these IV attacks because it authenticates the entire frame. It is also recommended that the IV must be unpredictable for any encrypted data [38]. In this particular case, as explained before in section 4.3, PSP uses a unique IV/cryptographic key pair every time it encrypts the information to avoid the formation of patterns of blocks that have the same clear text. Every time a new symmetric cryptographic key is defined, PSP calculates the HMAC-SHA1 of this new key and uses the resulting value as a Base IV Number to calculate the initialization vector (IV), which is defined by a mathematical function of the Base IV Number (BIVN) and the Sequence Number (SN).

If we were to analyze any of the frames shown in Figure 6.5, this is all the information we can get based on what PSP doesn't protect:

- Source Mac Address: 00-50-56-af-24-36
- Destination Mac Address: 00-50-56-af-33-ed
- Protocol: 0x1982 (PSP)
- Packet Security Protocol Version: 00
- Packet Security Protocol Options: Encryption + HMAC
    - Security Number: 0 (No)
    - Timestamp: 0 (No)
    - Reserved: 0 (No)
    - Digital Signature: 0 (No)
    - Encryption: 1 (Yes)
- Sequence Number: 0x77 0x9f 0x62 0x55
- Payload: Unknown
- HMAC: It is different with each frame.

6.1.3. Conclusions of the Security Analysis

Leaving the IP header unprotected can result in an easy way for an attacker to disrupt the communication of the network, impersonate a trusted host or steal information. Our ARP spoofing and Man in the Middle attacks and the redirection of network traffic to our SSH honeypot were successful because the encapsulated protocols and the IP header were exposed in clear text and didn't have integrity protection. After protecting the network with our proposed solution, it was impossible for us to replicate the attacks on the network. By protecting the network with our protocol, we can avoid network attacks based on packet injection, data tampering or information disclosed not only by IP packets, but by any protocol encapsulated into the layer 2 frames because the original ethertype field and the layer 3 data is encrypted and the entire Ethernet frame is integrity protected.

We also conclude that our solution relies on the strength of the encryption and hashing algorithms used to protect the information. We recommend avoiding the use of any encryption algorithm in Electronic Codebook mode because it generates data patterns that an attacker could use to extract more information or assume the kind of traffic being protected.

6.2. Performance Analysis

This analysis shows how network and host performance is impacted by the Packet Security Protocol. The performance efficiency of the protocol is based on the efficiency of the program we developed to prove the concept of our solution.

This analysis explains the results obtained from the different tests we ran, including transfer rate, round trip time, throughput, TCP time/sequence, TCP window scaling size, packet loss, CPU usage and memory usage.

In order to maximize the data capturing performance of the virtual machine running the Network Protocol Analyzer WireShark, we enabled the TCP Offloading Engine of the network card for the tests to the following protocols regardless of any test category: Iperf-TCP and SCP. As mentioned in section 5.3.2, some categories required enabling or disabling the TCP offloading engine of the network card. In our Linux environment, we used the following commands:

Enable TOE [39]:

```
sudo ethtool -K eth0 tso on
sudo ethtool -K eth0 gso on
sudo ethtool -K eth0 gro on
```

Disable TOE [39]:

```
sudo ethtool -K eth0 tso off
sudo ethtool -K eth0 gso off
sudo ethtool -K eth0 gro off
```

For system performance measurement, we used the Linux “top” command with a delay option of 1 second.

```
top –d 1
```

6.2.1. ARP-ICMP

The purpose of these tests was to measure the Round Trip Time of the Address Resolution Protocol (ARP) and the Internet Control Message Protocol (ICMP). To test these protocols, we issued the *ping* command from the client to the server virtual machine.

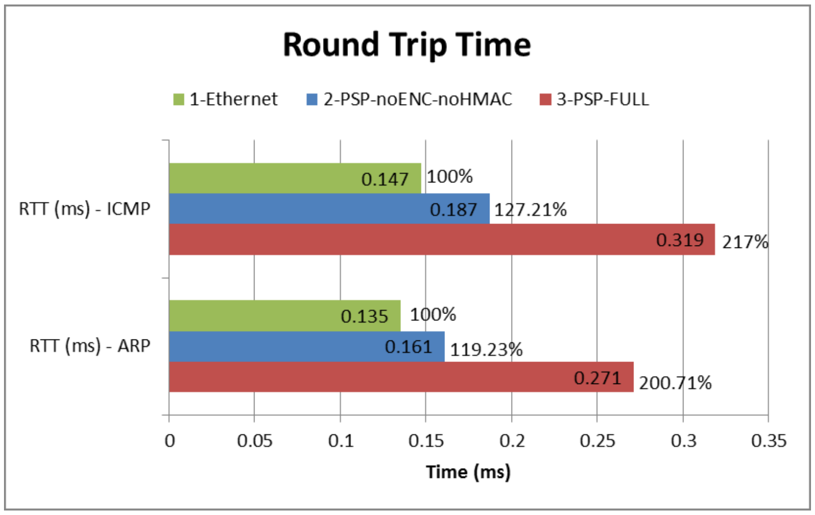


Figure 6.6. Round Trip Time of ARP and ICMP

Figure 6.6 depicts the Round Trip Time of the ARP and ICMP protocols. RTT was measured calculating the time difference between the client request and the server response as captured by WireShark.

We observed that the RTT of the protocols protected by our developed program (3-PSP-FULL) is significantly longer than the RTT of the protocols over a regular Ethernet communication (1-Ethernet). An intermediate point to analyze these results is the test of the protocols using PSP with no encryption and no HMAC (2-PSP-noENC-noHMAC). As explained in section 5.3.2 and shown in Figure 5.6, section 5.5, the encryption, decryption and HMAC modules have been disabled to analyze the impact these cause on performance. As seen in Figure 6.6, test results of ARP and ICMP for 2-PSP-noENC-noHMAC reflect an increase of 19.23% of the RTT for ARP and an increase of 27.21% for ICMP.

When we look at the test results of 3-PSP-FULL, we see that PSP and our developed program increased the RTT of ARP by 100.71% and 117% for ICMP. If we calculate the difference between 3-PSP-FULL and 2-PSP-noENC-noHMAC we can establish the overhead caused by the encryption/decryption and HMAC modules: 81.48% for ARP and 89.79% for ICMP.

These results are slightly correlated with the fact that ICMP frames are larger than ARP frames, resulting in more CPU processing due to frame construction and encryption.

6.2.2. VoIP

This test measured the performance of the proposed Packet Security Protocol on Voice over IP Applications. We used Asterisk v11.3.0 as our Voice over IP Server and the SIPp program to simulate SIP calls and RTP sessions.

The test consisted in generating, for 60 seconds, a limit of 100 simultaneous calls at a rate of 8 calls per second from SIPp to an Asterisk SIP extension number using the G.711 Codec. This call rate was the maximum supported by our test environment without dropping any calls in a regular Ethernet communication. For this test we did not increased the Linux's open file descriptors limit.

The command used to generate SIP calls from SIPp was:

```
./sipp -sf uac_pcap.xml -d 20000 -s 1001 192.168.40.8 -l 100 -r 8
```

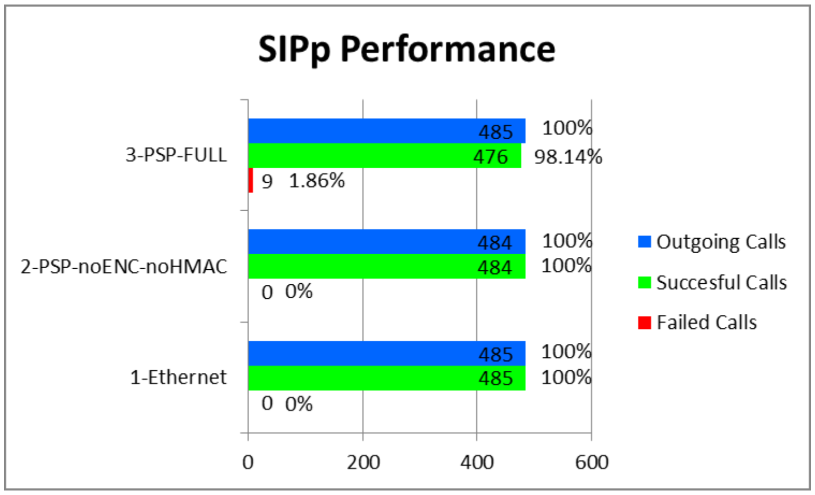


Figure 6.7. SIPp Performance

Figure 6.7 depicts the SIPp performance. Tests categories 1-Ethernet and 2-PSP-noENC-noHMAC had a 100% of successful calls. In the case of the proposed Packet Security Protocol and the developed program (3-PSP-FULL), only 9 calls out of 485 failed for a total of 98.14% of successful calls.

Average mean and max jitter per call, as calculated by Wireshark, are shown in Figure 6.8. For all test categories, jitter remained under the limits of 40 ms required for a good call quality [40].

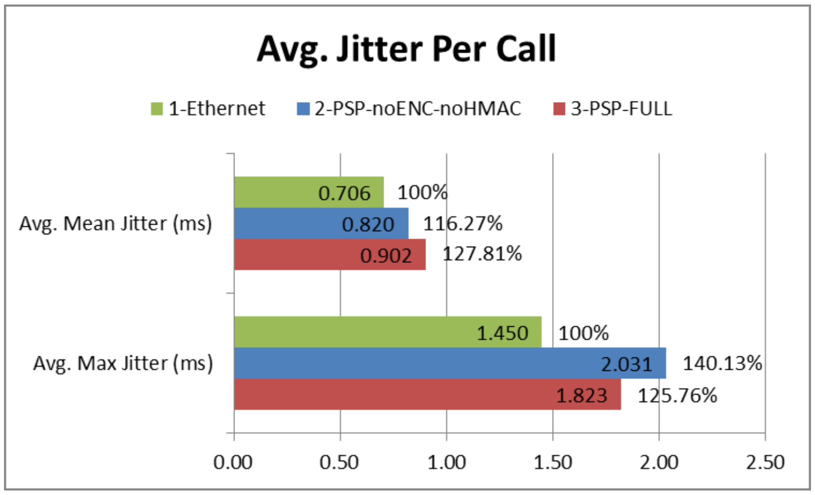


Figure 6.8. Average Calculated Jitter Per Call

System performance is not highly impacted. As shown in Figure 6.9, the developed program used a maximum of 27.4 % of CPU time (3-PSP-FULL). Due to the fact that the network transfer rate was relatively low (586 KB/Sec), PSP didn't have to encrypt/decrypt and calculate the HMAC values of too many packets. More details on transfer rate are explained in Figure 6.12.

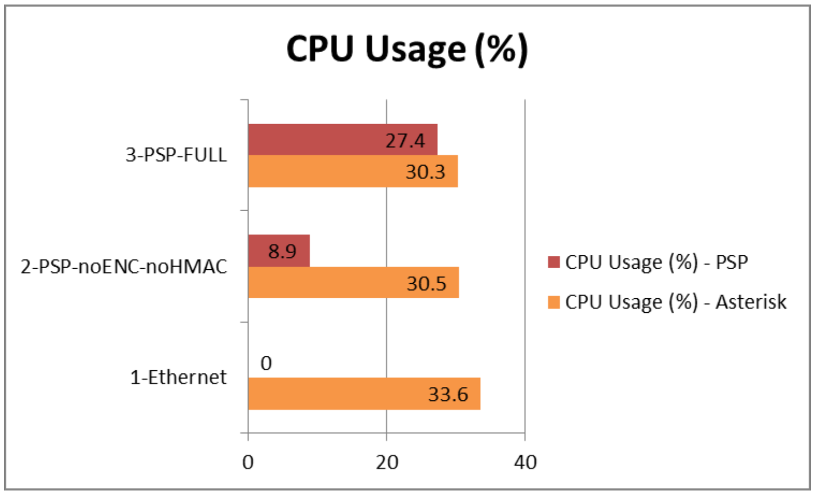


Figure 6.9. CPU Usage of Asterisk and PSP

Figures 6.10 and 6.11 show the memory used by Asterisk and the developed program. For categories 2-PSP-noENC-noHMAC and 3-PSP-FULL, Asterisk used 6,000 KB less than the memory used in category 1-Ethernet. The developed program used 3,204 KB of memory, out of which approximately 2,976 correspond to the encryption/decryption and HMAC modules. This value was calculated by subtracting the memory usage of the developed program in category 2-PSP-noENC-noHMAC from the memory used in category 3-PSP-FULL.

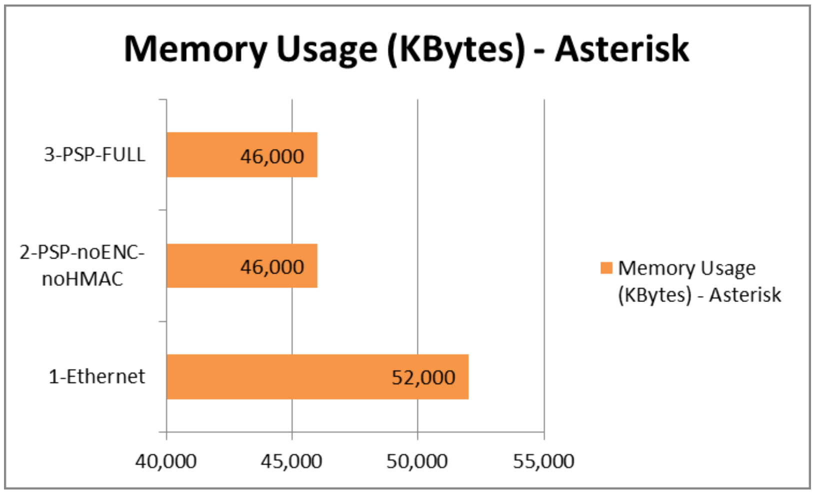


Figure 6.10. Asterisk Memory Usage

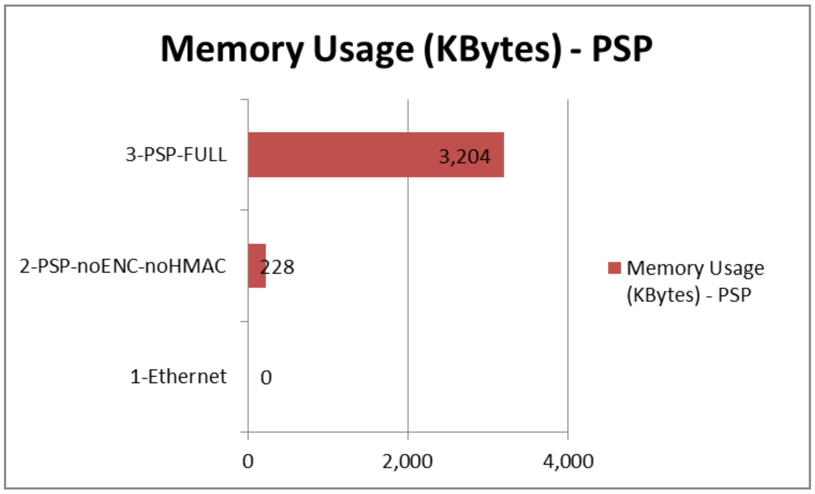


Figure 6.11. PSP Memory Usage During the VoIP Test

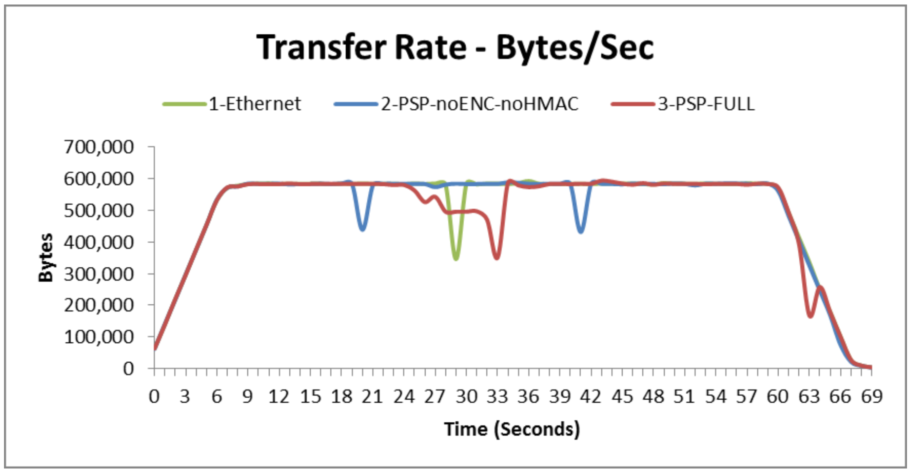


Figure 6.12. VoIP Transfer Rate in Bytes/Second

Transfer rate, as depicted in Figure 6.12, remained the same over time for all categories. As previously mentioned in this section, the transfer rate is approximately 586 KB/Sec. Such a low flow of packets didn't put too much effort on the CPU to process all the information.

6.2.3. Skype

On this test, we measured the performance of the proposed Packet Security Protocol on video and voice over IP applications. We decided to use Skype for this purpose.

Our test scenario was shown in Figure 5.2, section 5.3.1 and involved a host establishing a Skype video conference with another host over the Internet. Only one of the hosts was protected with the proposed Packet Security Protocol. Both hosts were connected to the Internet using a DSL connection with a downstream bandwidth of 3Mbps and upstream bandwidth of 768 Kbps.

Due to limitations of the hardware, operating system and virtual environment used for this test, the virtual machine protected by the proposed Packet Security Protocol was able to receive audio and video from the remote host, and it was able transmit audio but not video.

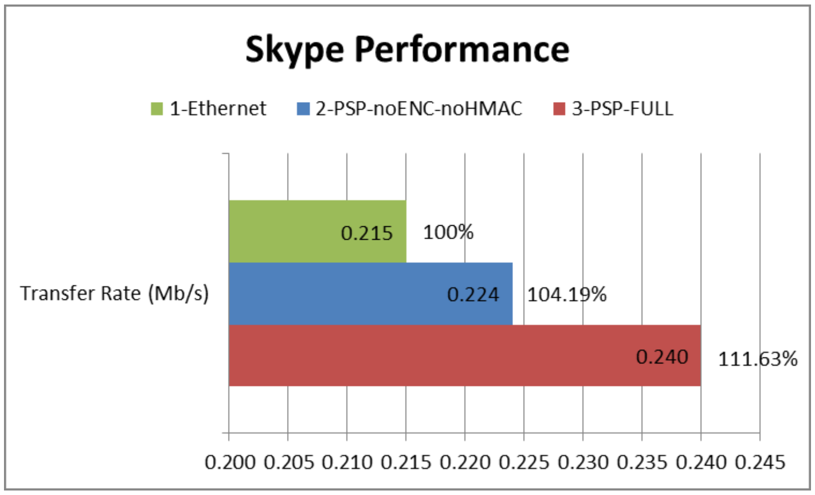


Figure 6.13. Skype Network Performance

The network performance of Skype is depicted in Figure 6.13. We observed that transfer rate was better in categories 2-PSP-noENC-noHMAC and 3-PSP-FULL, but this network performance increment is not related to any advantage of the proposed protocol.

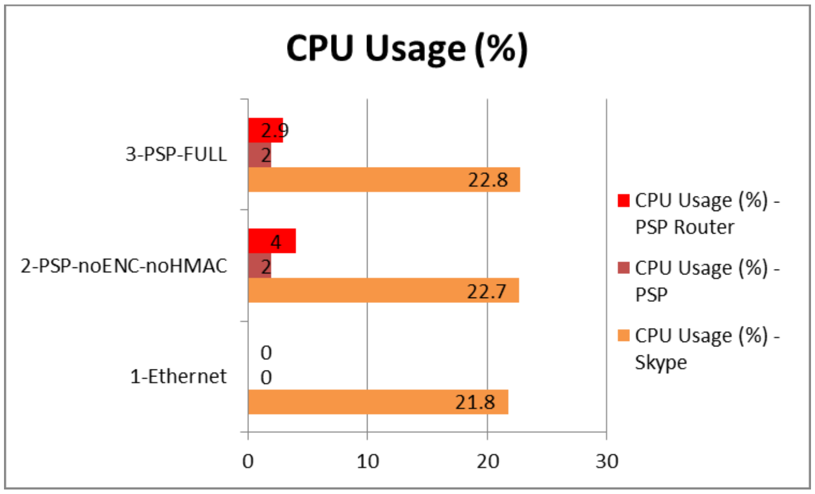


Figure 6.14. CPU Usage of Skype, PSP and PSP in the Router

The CPU usage remained very low along all test categories. Figure 6.14 shows a usage of 2.9% of CPU time on the Router and 2% on the host in category 3-PSP-FULL. Skype CPU usage remained almost the same in all test categories.

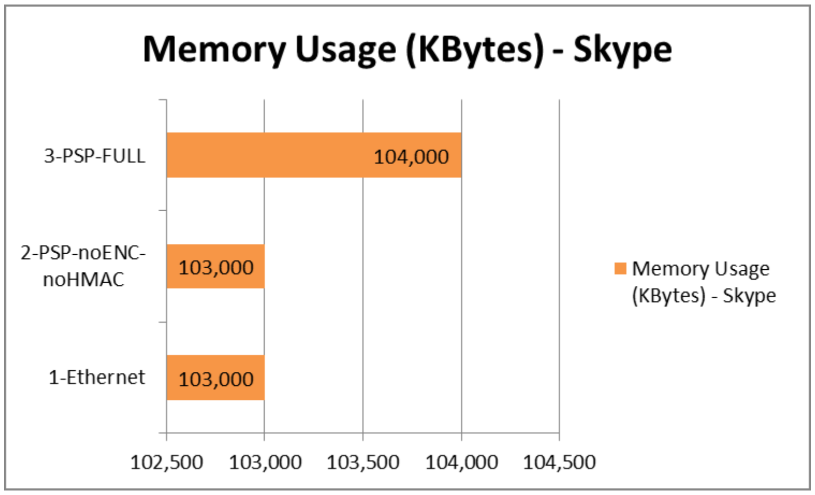


Figure 6.15. Skype Memory Usage

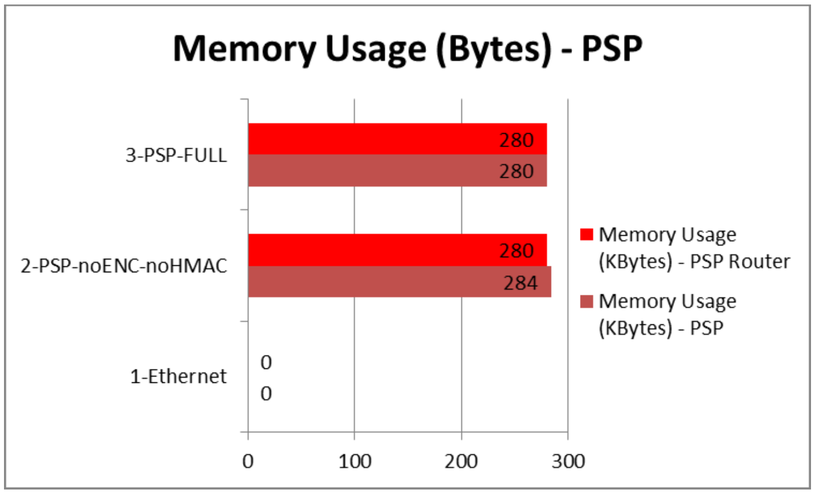


Figure 6.16. PSP Memory Usage in the Host and in the Router During the SkypeTest

The memory usage of Skype and PSP was not impacted by any of the test categories. As shown in Figure 6.15, Skype memory usage increased less than 1% in test category 3-PSP-FULL. This is the result of the increased transfer rate already shown in Figure 6.13.

As seen in Figure 6.16, memory usage of the developed program remained the same in all test categories.

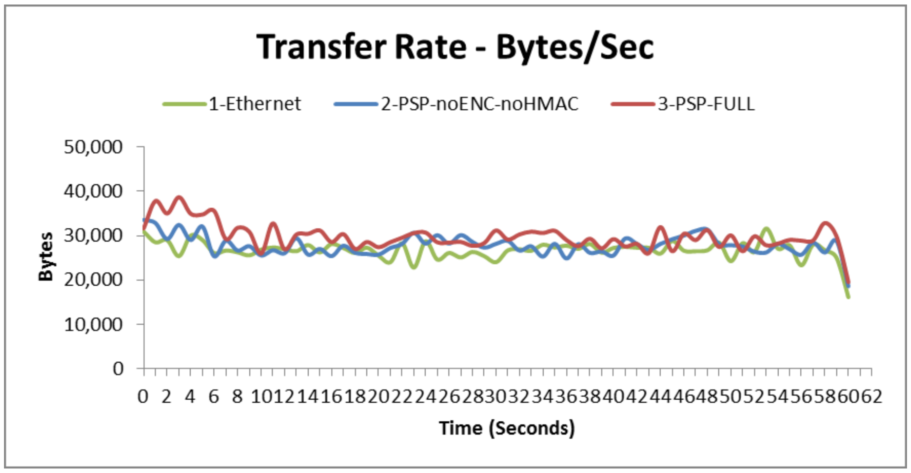


Figure 6.17. Skype Transfer Rate in Bytes/Second

Figure 6.17 depicts the Skype transfer rate for all test categories. We observed that it remained almost the same over time, showing very similar results to the ones attained in the VoIP test. Low transfer rates didn't put too much effort on the developed program and helped to keep the CPU usage at low percentages.

6.2.4. IPERF-TCP

This test measured the performance of the TCP protocol protected with the Packet Security Protocol.

To complete this test, we used Iperf with all default settings:

- TCP Window Size = 85.3 Kbytes.
- Test duration = 10 seconds.

We used the following commands to set the Iperf TCP server and generate TCP traffic from the client:

- Set the Iperf TCP server: `Iperf -s`
- Generate TCP traffic from the client to the server: `Iperf -c 192.168.40.8`

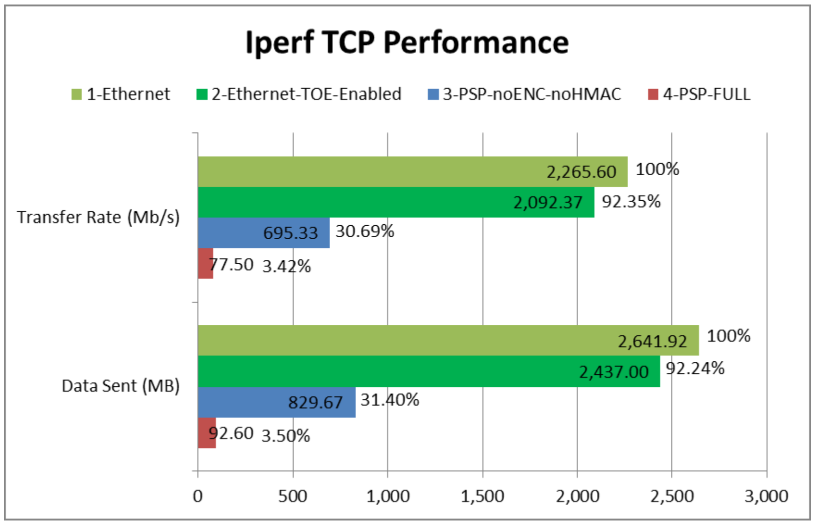


Figure 6.18. Iperf TCP Performance

Figure 6.18 depicts the TCP performance and total data transferred on each test category as measured by Iperf. We used test category 1-Ethernet as our baseline for all other results. In this scenario, test results of categories 3-PSP-noENC-noHMAC and 4-PSP-FULL showed a very low performance compared to test category 1-Ethernet. Our developed program (test category 4-PSP-FULL) had only 3.42% of the network performance of our baseline while test category 3-PSP-noENC-noHMAC only had 30.69%. These results are very different to our findings on the previous tests, but what was slightly suggested before about the developed program in the ARP-ICMP tests, can be confirmed now: the higher the network traffic, the lower the network performance is.

This test gave us much information about the performance of the Proposed Packet Security protocol and the developed program. Test results of category 3-PSP-noENC-noHMAC showed that running the application in user space drastically impacted, in almost 70%, the network performance. This condition worsened when encryption/decryption and HMAC algorithms were used, negatively impacting the network performance in almost 97%.

The developed program (test category 4-PSP-FULL) not only resulted in a low network performance, but in a high CPU usage as well. As seen in Figure 6.19, it used more than 90% of CPU time in test category 4-PSP-FULL. From this test result we were able to estimate that high network traffic protected by PSP will result in high CPU usage due to the fact that the developed application will have to encrypt/decrypt and calculate HMAC values at a very high speed.

Test category 2-Ethernet-TOE-Enabled resulted in a 92.35% of the network performance of our baseline as shown in Figure 6.18, but if we observe the CPU usage in Figure 6.19, Iperf only used 11% of CPU time compared to 99% of our baseline category 1-Ethernet.

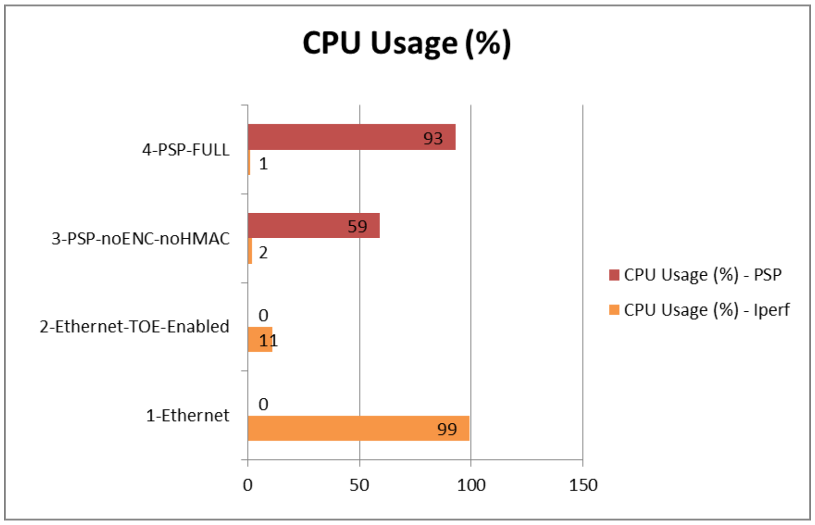


Figure 6.19. CPU Usage of Iperf-TCP and PSP

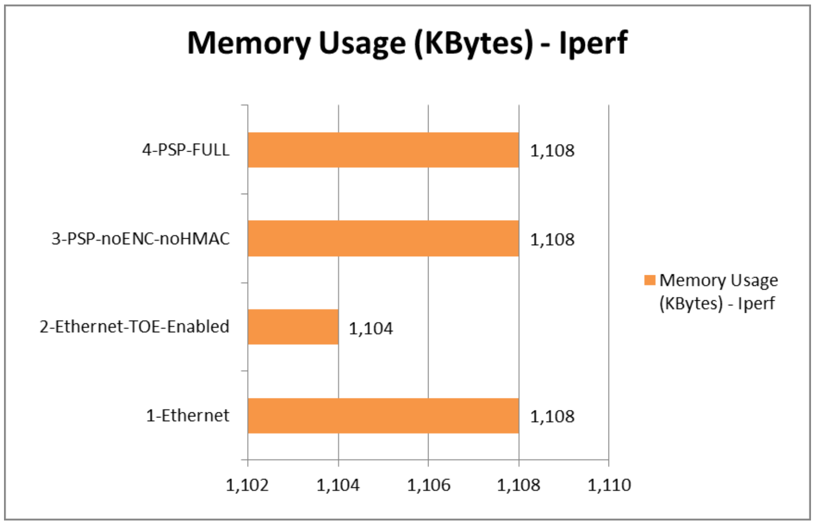


Figure 6.20. Iperf-TCP Memory Usage

Iperf memory usage didn't change on any of the test categories, except on category 2-Ethernet-TOE-Enabled, which presents a slightly variance of 4 Kbytes as shown in Figure 6.20.

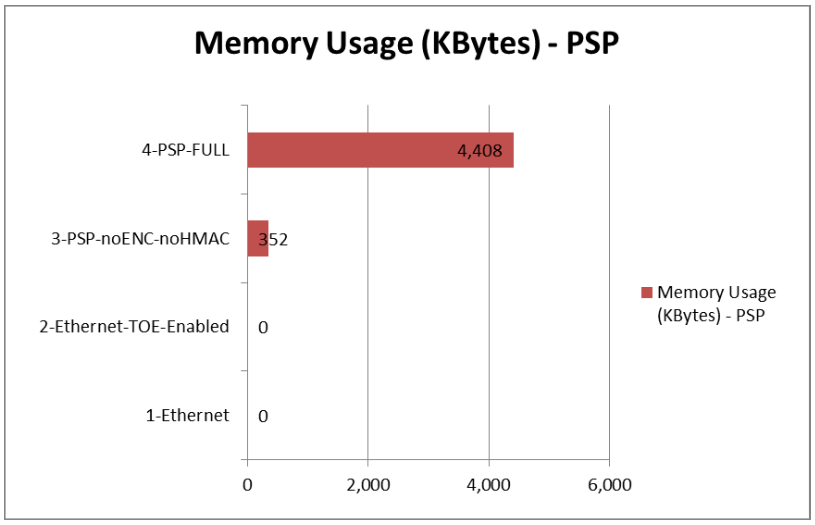


Figure 6.21. PSP Memory Usage During the Ipert-TCP Test

Figure 6.21 depicts the memory used by the developed program. Test categories 3-PSP-noENC-noHMAC and 4-PSP-FULL helped us to determine the impact on memory usage caused by the encryption, decryption and HMAC modules. As observed, there is a memory usage difference of 4,056 Kbytes between both test categories, and it is because of the effort of the developed program to encrypt, decrypt and calculate the HMAC values of all protected packets.

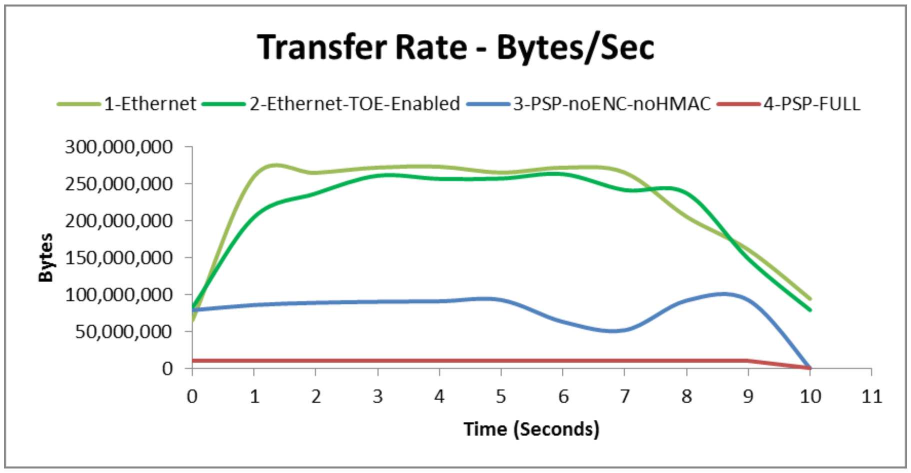


Figure 6.22. Iperf-TCP Transfer Rate in Bytes/Second

Iperf-TCP transfer rate is depicted in Figure 6.22. Test categories 1-Ethernet and 2-Ethernet-TOE-Enabled presented a variable transfer rate over time: It started slow, achieved and sustained its maximum for a prolonged period and then started to drop until the transmission was finished. This behavior is completely normal for an environment controlled by TCP and where data is delivered fast enough to the kernel. Test category 3-PSP-noENC-noHMAC presented a similar behavior but with a reduced transfer rate. What seemed to be different was test category 4-PSP-FULL. Transfer rate was almost the same since the beginning of the transmission until the end; however, the application could not deliver data fast enough to the kernel due to the high usage of CPU time.

6.2.4.1. Round Trip Time

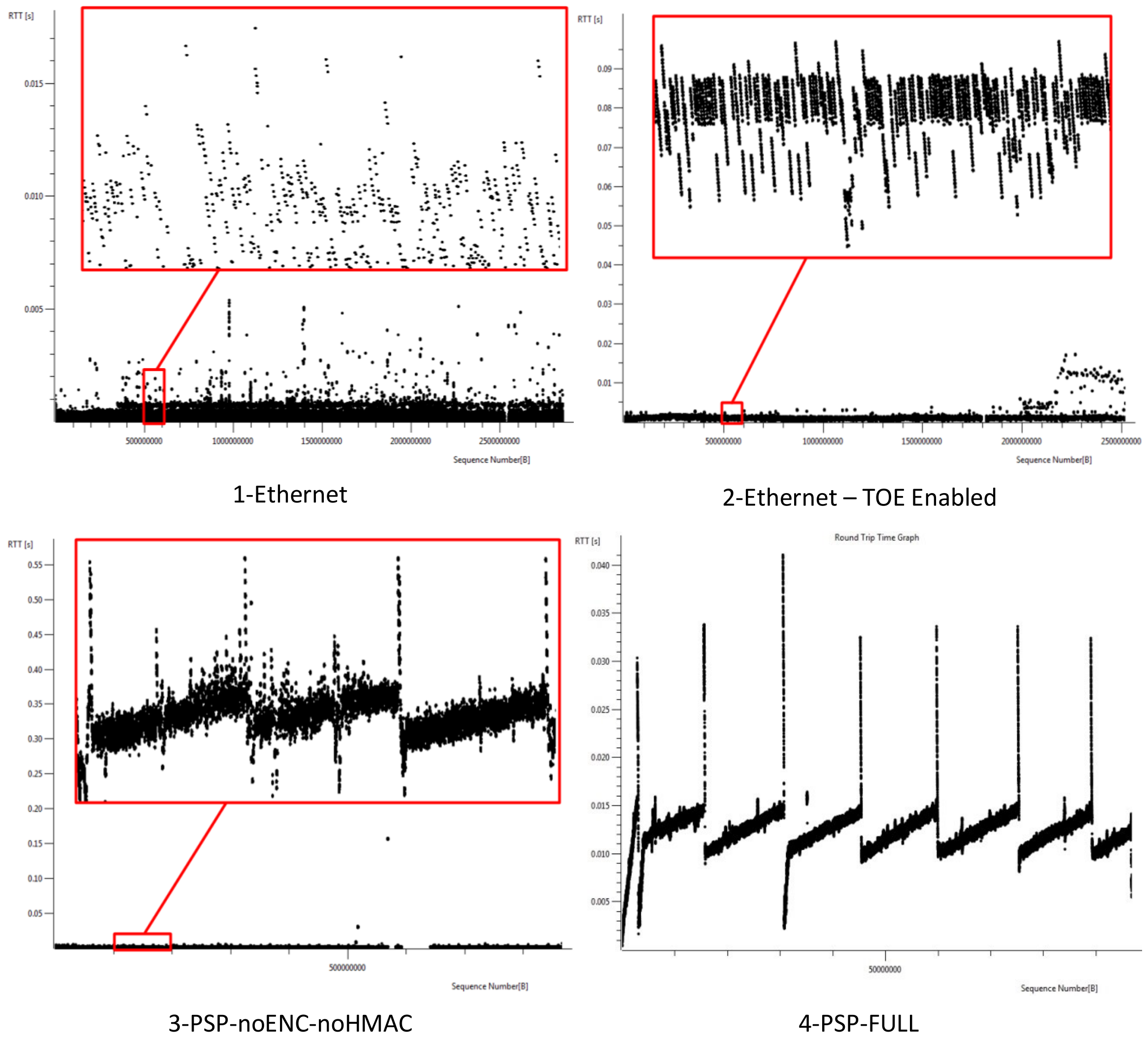


Figure 6.23. Iperf-TCP Round Trip Time

Round Trip Time, as measured by WireShark, is the time it takes for a packet to be transmitted by the client and acknowledged by the server [41]. Figure 6.23 depicts the RTT graph for each test category, and from this, we could observe a very interesting behavior in the TCP transmission.

Test category 4-PSP-FULL presented a repetitive pattern of long-vertical lines every certain amount of transmitted packets. The same pattern could be observed in the RTT graph of test category 3-PSP-noENC-noHMAC; however, test categories 1-Ethernet and 2-Ethernet-TOE-Enabled showed different patterns. The long-vertical lines represent sequential duplicate ACK packets sent by the Ipert-TCP server because some TCP packets were lost during the transmission.

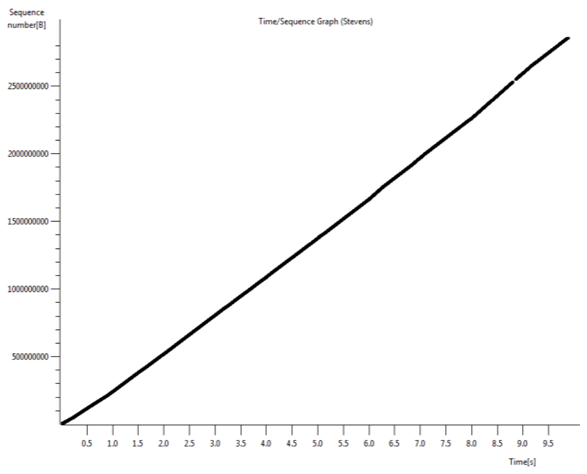
Table 6.1. Duplicate ACK Packets on Each Test Category for Iperf-TCP

	Captured Packets	Duplicate ACK Packets	Percentage of Duplicate ACK Packets
1-Ethernet	244,090	2	0.0008%
2-Ethernet-TOE-Enabled	137,137	31	0.023%
3-PSP-noENC-noHMAC	823,317	5,335	0.648%
4-PSP-FULL	103,417	875	0.846%

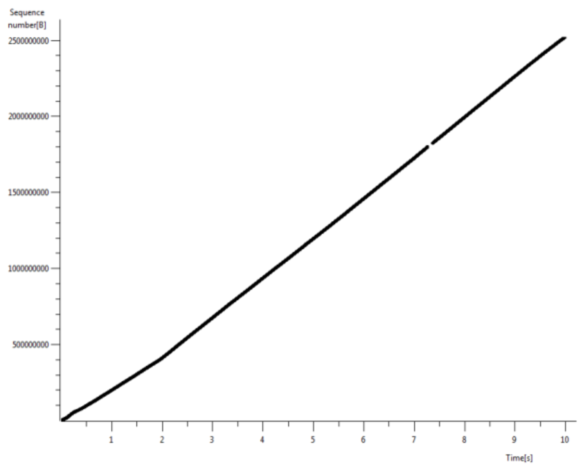
We observed, as shown in Table 6.1, that duplicate ACK packets increased in each test category. In our test scenario, packet loss could occur mainly for two reasons: There was either a transmission problem caused by a faulty medium (air, wire, fiber), or data from the TCP receiving buffer couldn’t be extracted fast enough to receive new data. We decided to discard the faulty medium because our test environment was 100% virtualized, which led us to conclude that the loss of packets was a result of the CPU not being able to process the affected bytes on time. There are other consequences derived from this situation that will be explained later, as for example, the TCP window scaling.

If we go back to Figure 6.19, we’ll see that test categories 3-PSP-noENC-noHMAC and 4-PSP-FULL were very CPU intensive. In the case of test categories 1-Ethernet and 2-Ethernet-TOE-Enabled we could get confused with the fact that test category 1-Ethernet was more CPU intensive than test category 2-Ethernet-TOE-Enabled; however, the CPU of the network card can’t process large loads of data as fast as the CPU of the computer system.

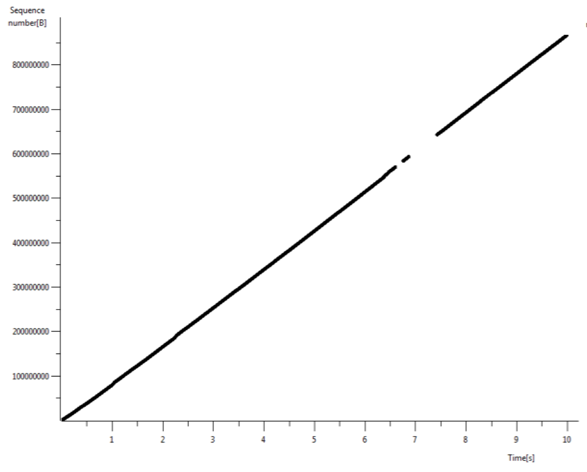
6.2.4.2. Time/Sequence



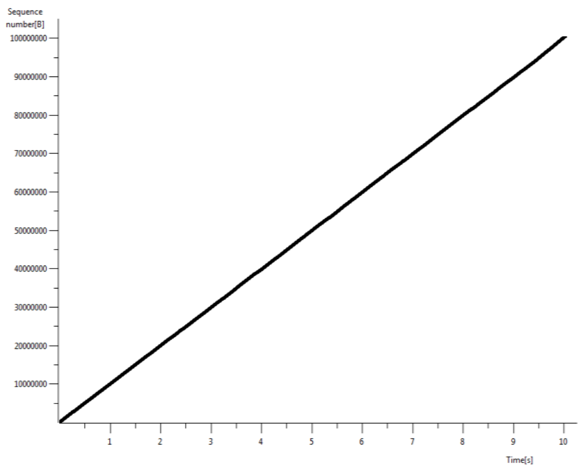
1-Ethernet



2-Ethernet – TOE Enabled



3-PSP-noENC-noHMAC



4-PSP-FULL

Figure 6.24. Iperf-TCP Time/Sequence

The Time/Sequence graph for all test categories showed a very stable transmission, as depicted in Figure 6.24.

The graph in test categories 1-Ethernet, 2-Ethernet-TOE-Enabled and 3-PSP-noENC-noHMAC presented a gap or interruption; however, this was the inability of WireShark to capture all packets in the data communication process, and not a pause in the data transmission.

All test categories presented an upward trend that is more stable in test category 4-PSP-FULL.

6.2.4.3. Throughput

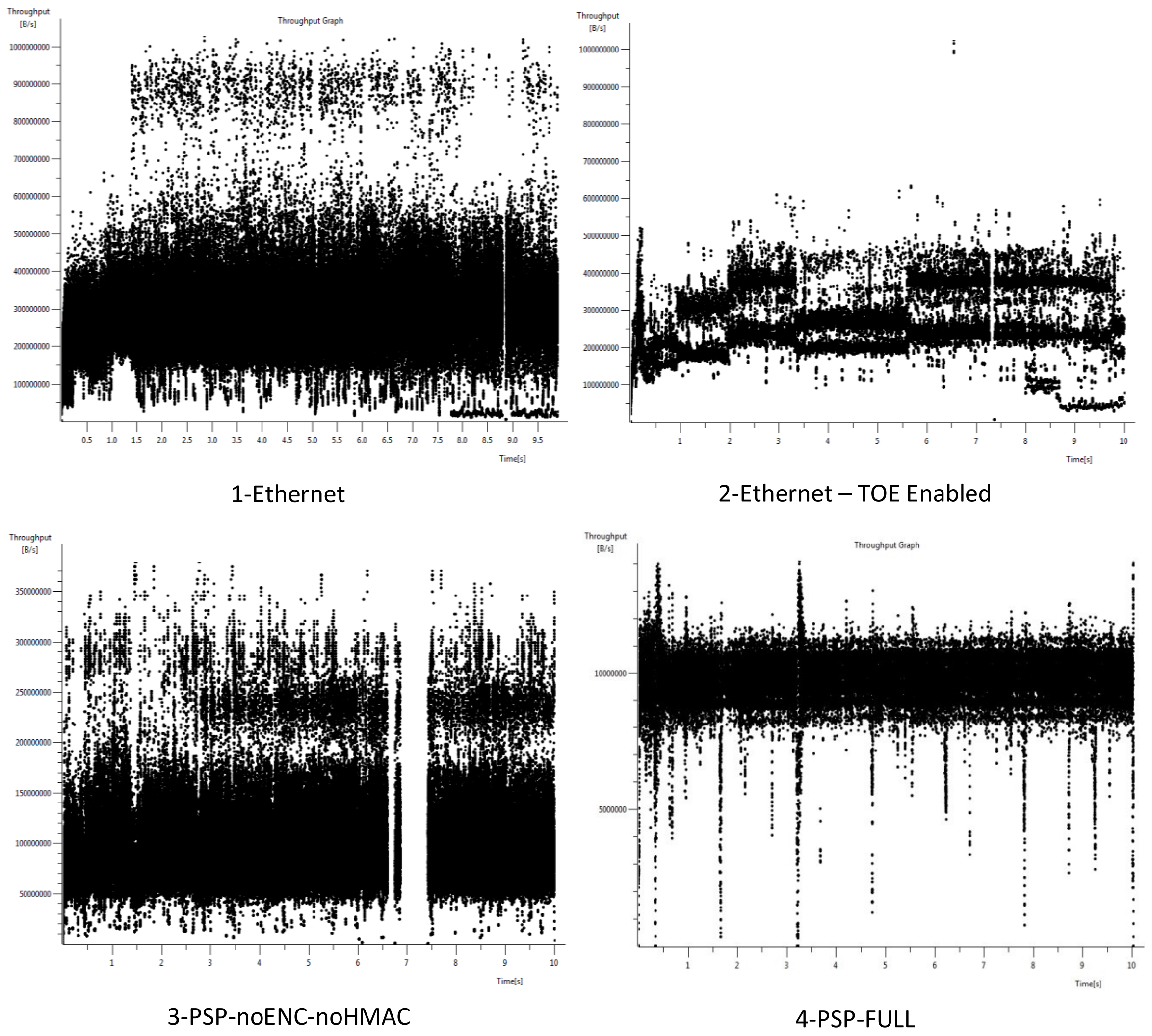
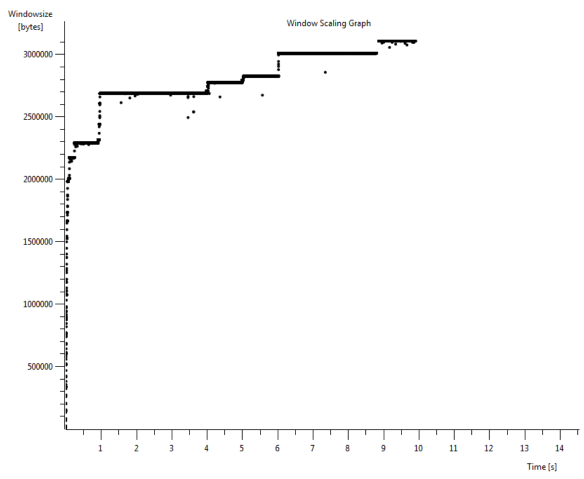


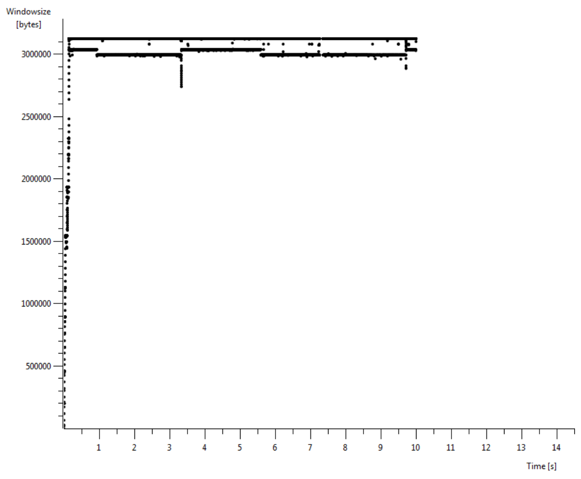
Figure 6.25. Iperf-TCP Throughput

All test categories had a very consistent throughput, as depicted in Figure 6.25. Test category 4-PSP-FULL had a transfer rate between 9 and 11 MB/second with some sequential packets with a reduced throughput and a drip like pattern. Throughput remained stable in all test categories and can be considered normal, with no anomalies observed.

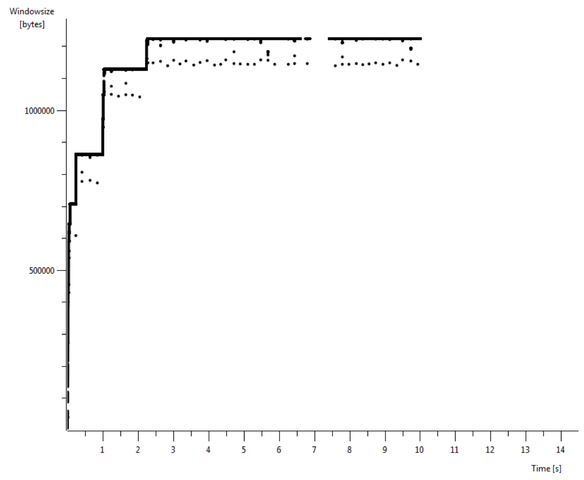
6.2.4.4. TCP Window Scaling – Server Side



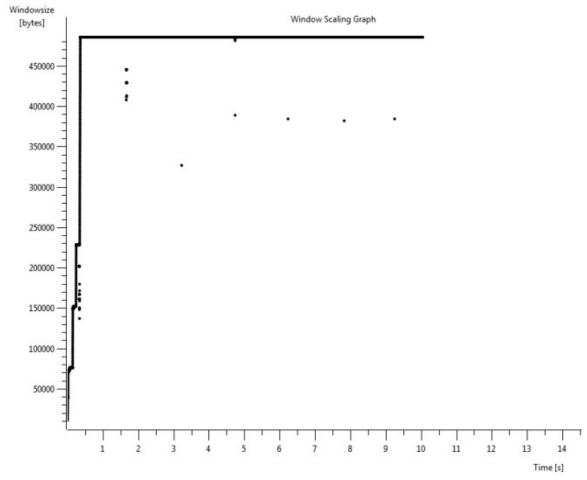
1-Ethernet



2-Ethernet – TOE Enabled



3-PSP-noENC-noHMAC



4-PSP-FULL

Figure 6.26. Iperf-TCP Window Scaling – Server Side

The server's TCP window scaling for all test categories is shown in Figure 6.26. The TCP receive window size is the amount of bytes TCP can receive before sending an acknowledge packet [42]. This value varies during the transmission of packets and depends greatly on the stability of the communication and the capacity of the hosts to process the bytes. Based on the previous definition, there are two aspects we have to analyze: maximum achieved window size and window size variation over time.

Test category 4-PSP-FULL reached a maximum TCP receiving window size of 480,000 bytes. This is a small window compared to the size of the TCP receiving window of test categories 1-Ethernet and 2-Ethernet-TOE-Enabled, which reached a maximum of 3,100,000 bytes. This reduction means that the developed program (test category 4-PSP-FULL) required more processing of packets, just as we saw in Figure 6.19, and couldn't open the receiving window much more. Test category 3-PSP-noENC-noHMAC presented a maximum TCP window size of 1,200,000 bytes; even though, this result was not as low as test category 4-PSP-FULL, it told us that the efficiency of the program resulted in a serious performance degradation, just as we saw with each one of the factors previously analyzed for the Iperf-TCP test.

All tests showed a constant increment of the window size over time with only some negative variations. It means that the communication process was affected by the capacity of the hosts to process all the received data, and by how fast they could empty their buffers to fill them up with new bytes. Test category 4-PSP-FULL had the most stable window size variation compared to the other test categories. The window size variations of this test category are consistent with the long vertical lines observed in the Round Trip Time graph of test category 4-PSP-FULL in Figure 6.23. Figure 6.27 shows a clear view of what has been said.

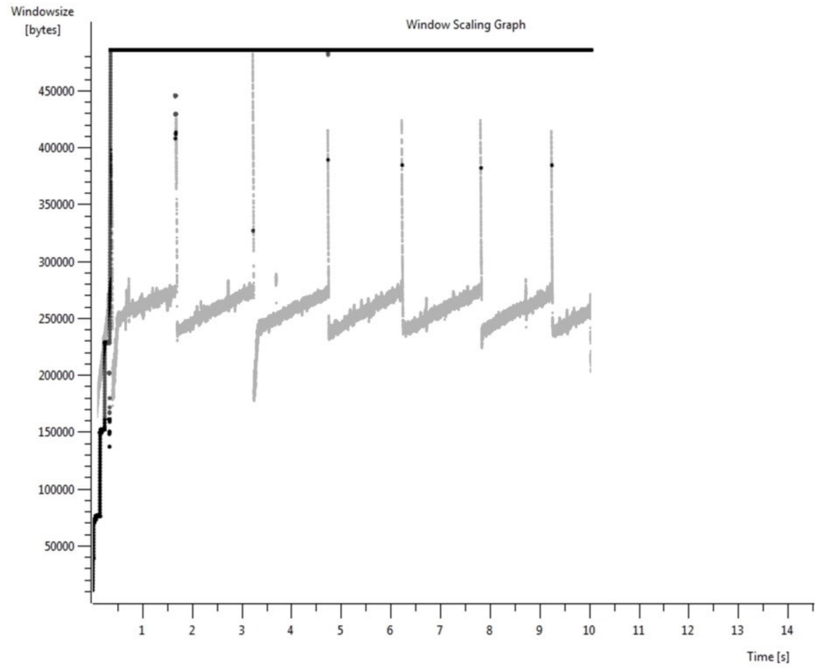


Figure 6.27. Iperf-TCP Window Scaling – Server Side vs. RTT of Test Category 4-PSP-FULL

6.2.5. IPERF-UDP

UDP is an unreliable transport protocol that doesn't guarantee the delivery of data. Unlike TCP, it doesn't use flow control, congestion control or any other process to guarantee an efficient transport of bytes; however, this lack of control translate into faster communications especially for real time protocols.

This test measured the performance of the UDP protocol protected by proposed Packet Security Protocol.

We used Iperf with all default settings to perform our tests:

- UDP Buffer Size = 208 Kbytes.
- Test duration = 10 seconds.

We used the following commands to set the Iperf UDP server and generate UDP traffic from the client:

- Set the Iperf UDP server:
```
Iperf -s -u
```
- Generate UDP traffic from the client to the server:
```
Iperf -c 192.168.40.8 -u -b 1000m
```

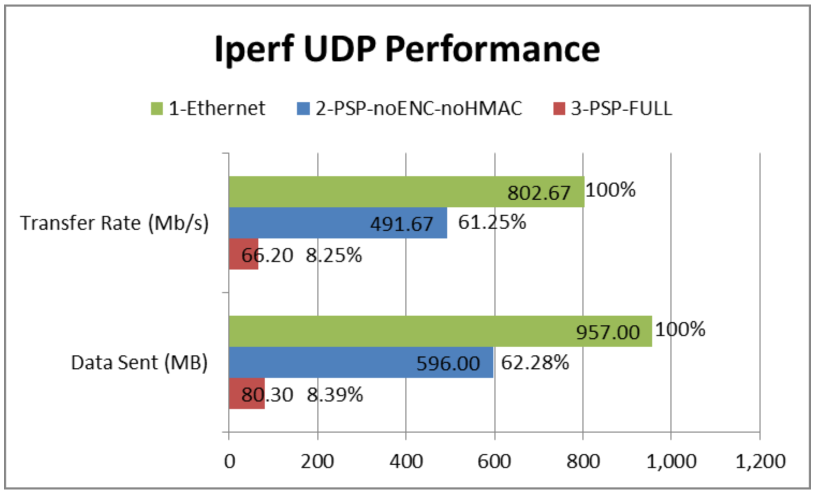


Figure 6.28. Iperf UDP Performance

UDP performance and total data transferred on each test category, as measured by Iperf, are shown in Figure 6.28. As in the Iperf-TCP test, we used test category 1-Ethernet as our baseline for all other results. The outcomes of this test are consistent with the results obtained in Iperf-TCP; test results of categories 2-PSP-noENC-noHMAC and 3-PSP-FULL showed a low performance compared to test category 1-Ethernet. Our developed program (test category 3-PSP-FULL) had 8.25% of the network performance of our baseline while test category 2-PSP-noENC-noHMAC, got much better results by having a 61.25% of the network performance of test category 1-Ethernet.

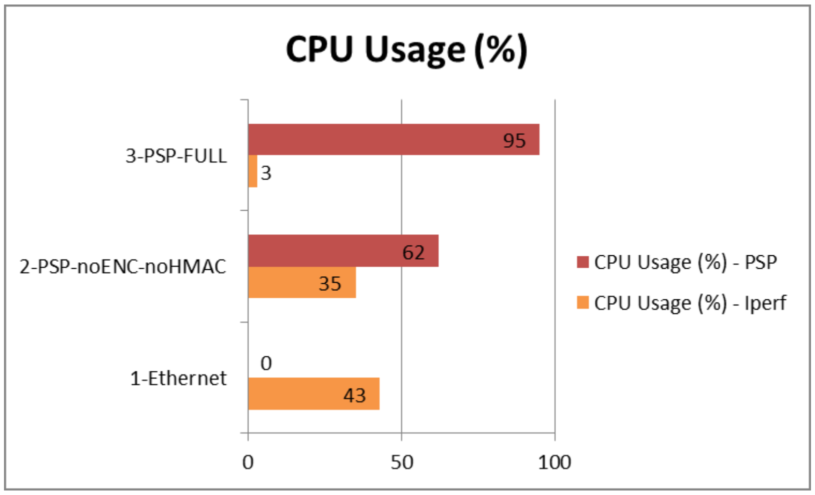


Figure 6.29. CPU Usage of Iperf-UDP and PSP

The CPU time used by our developed program while protecting the network traffic with Packet Security Protocol, remained high for this test as is depicted in Figure 6.29. This is the result of encrypting/decrypting and calculating HMAC values at a very high speed.

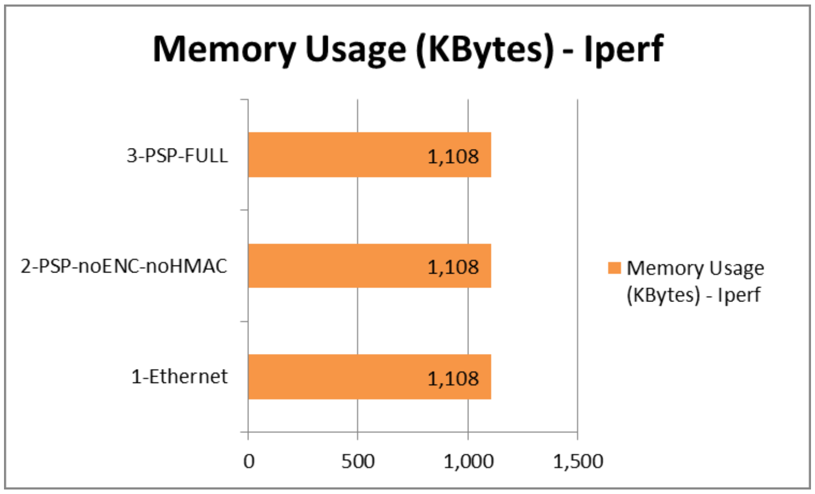


Figure 6.30. Ipetf-UDP Memory Usage

Iperf memory usage, as depicted in Figure 6.30, didn't change on any of the test categories; however, Figure 6.31 shows that the memory used by our developed program varied for each one of the test categories. This difference in memory utilization depends greatly on the amount of data that passes through the encryption/decryption and HMAC modules of the program.

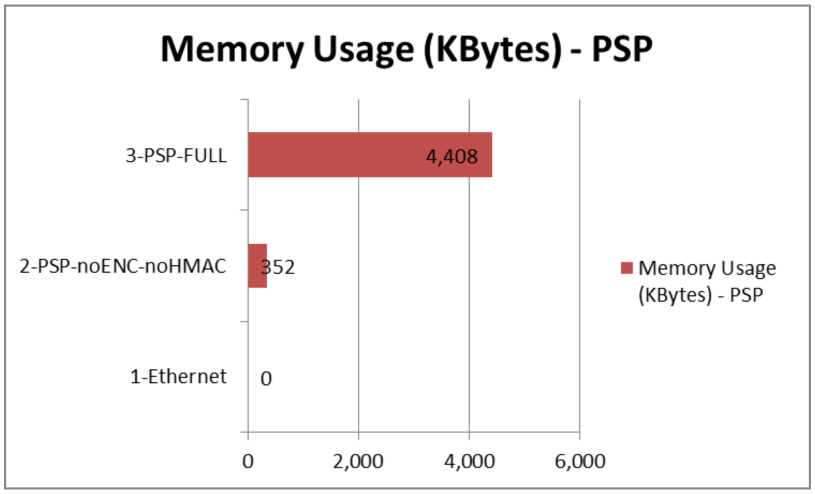


Figure 6.31. PSP Memory Usage During the Ipert-UDP Test

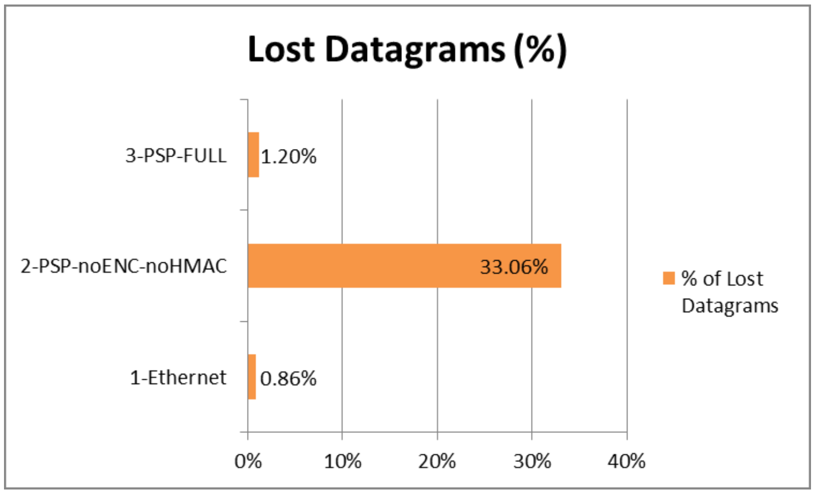


Figure 6.32. UDP Lost Datagrams

Due to the connectionless nature of the UDP protocol, it is expected to have certain loss of data during transmission. We observed that the percentage of lost Datagrams, as shown in Figure 6.32, was remarkably high (33.06%) in test category 2-PSP-noENC-noHMAC. This high percentage of lost datagrams didn't occur with the full implementation of the developed program in test category 3-PSP-FULL. This behavior in test category 2-PSP-noENC-noHMAC is more likely to be the result of inefficiency in some module of the developed program.

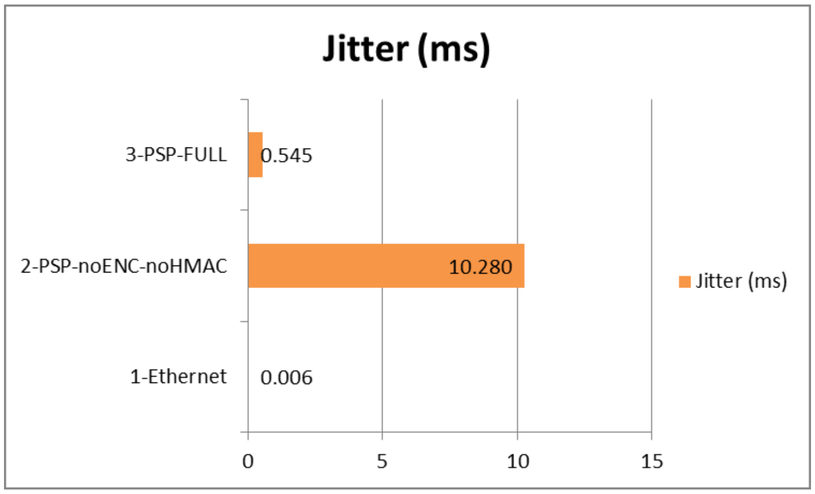


Figure 6.33. Jitter in the Iperf-UDP Test

Jitter for this UDP test, as measured by Iperf, is depicted in Figure 6.33. Test categories 2-PSP-noENC-noHMAC presented a very high Jitter compared to test categories 1-Ethernet and 3-PSP-FULL, and as explained above, it could be due to an inefficiency of some module of the developed program; however, the result obtained by the full implementation of PSP in test category 3-PSP-FULL indicates that doesn't seem to present any inconvenient for real time applications.

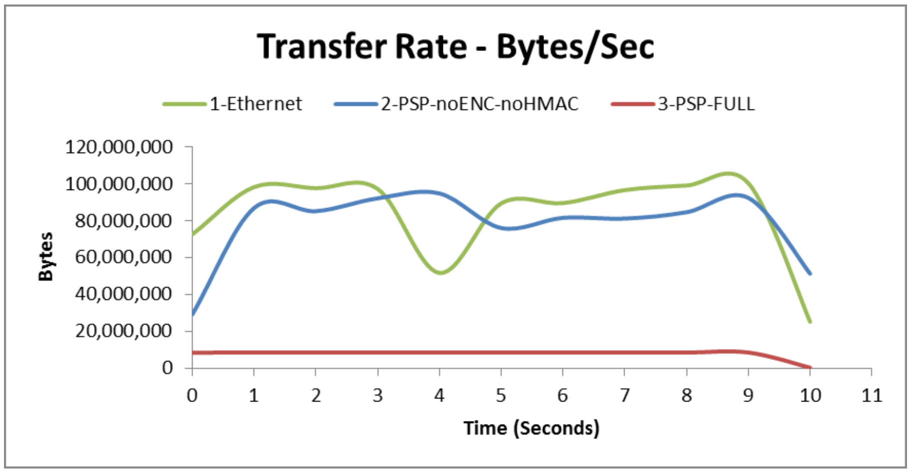


Figure 6.34. Iperf-UDP Transfer Rate in Bytes/Second

Iperf-UDP transfer rate is depicted in Figure 6.34. Test categories 1-Ethernet and 2-PSP-noENC-noHMAC presented a variable transfer rate over time; a behavior that, according to the depicted graphs, is normal for this kind of UDP communication. Test category 3-PSP-FULL presented a very stable transfer rate that is almost static from the beginning to the end of the communication.

6.2.6. SCP

In our previous tests we analyzed different protocols and services protected by our proposed Packet Security Protocol: We tested ARP, ICMP, VoIP and Skype as protocols and services that didn't require too much bandwidth. We also used Iperf as a tool to test the performance of the TCP and UDP protocols using as much bandwidth as possible.

This section presents an analysis of the network and system performance of the Packet Security Protocol while protecting a file transfer using SCP, a protocol that uses SSH to protect the transfer of files from a client to a server [43]. SCP by itself doesn't offer any protection mechanism; this is a function handled by SSH.

For this test, we created a dummy file of 200 MB using the following command:

```
dd if=/dev/zero of=dummyfile bs=209715200 count=1
```

The file was transferred from the client machine to the server using this command:

```
scp dummyfile root@192.168.40.8:/root/Desktop
```

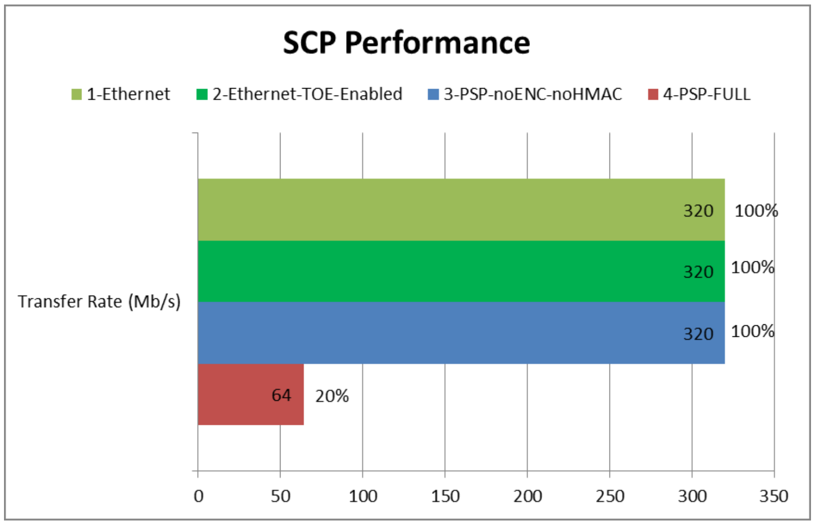


Figure 6.35. SCP Network Performance

The network performance of the SCP protocol for each test category is depicted in Figure 6.35. There were two things out of these results that brought our attention: a) The network performance was very low compared to the Iperf-TCP or Iperf-UDP results, and b) The network performance was the same in test categories 1-Ethernet, 2-Ethernet-TOE-Enabled and 3-PSP-noENC-noHMAC.

The SCP protocol adds a new processing layer to the data transfer process, and it is the SSH protocol. In order to secure data, SCP relies on SSH to encrypt each byte in user space. This translates into high CPU usage due to the encryption process used by SSH, as can be observed in Figure 6.36.

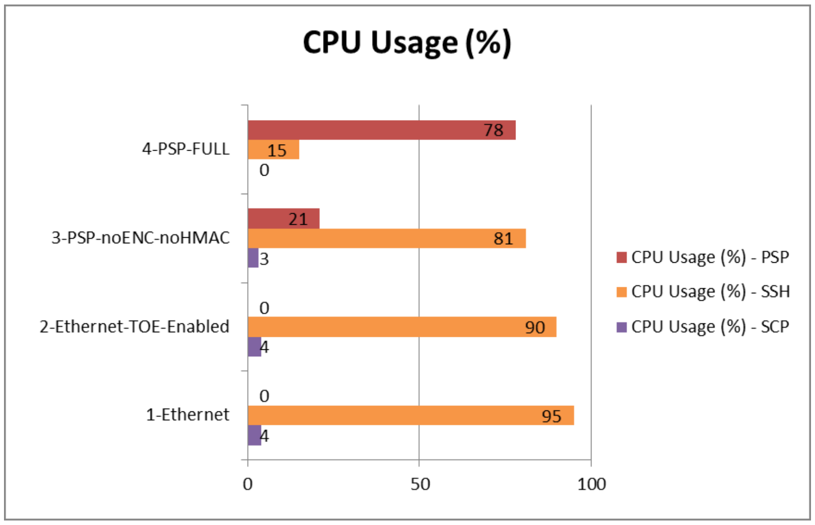


Figure 6.36. PSP, SSH and SCP CPU Usage

The SSH process used 95% and 90% of CPU time during test categories 1-Ethernet and 2-Ethernet-TOE-Enabled. If we compare the results of both test categories, we can see that the there is almost no difference between them and that the TCP offloading engine didn’t have any effect because the kernel of the operating system didn’t offload data to the network card.

The CPU usage of the SSH process is still high in test category 3-PSP-noENC-noHMAC. It used 81% of CPU time while our developed program without the encryption/decryption and HMAC modules only used 21%. Test category 4-PSP-FULL presented different results for all processes, where SSH used about 15% of CPU time and our developed program used 78%.

SCP used 4% of CPU time during test categories 1-Ethernet and 2-Ethernet-TOE-Enabled, 3% during test category 3-PSP-noENC-noHMAC and less than 1% during test category 4-PSP-FULL.

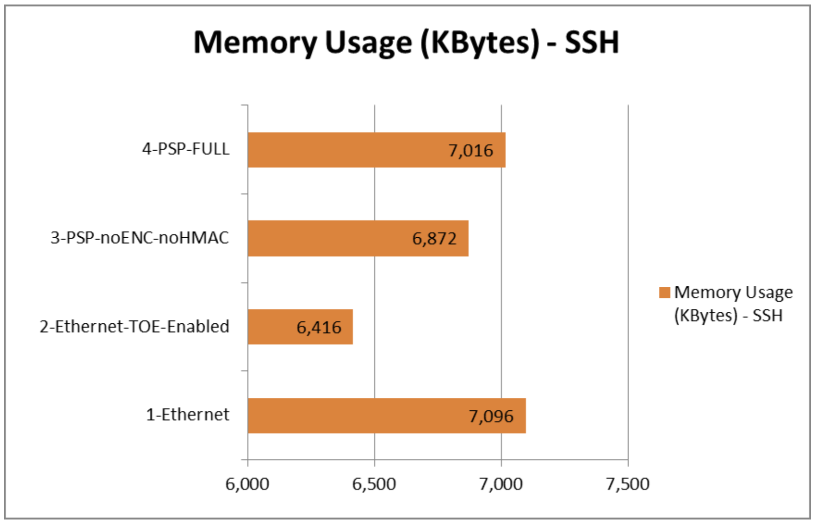


Figure 6.37. SSH Memory Usage

SSH memory usage didn’t vary much from one test category to the other, as shown in Figure 6.37. The only test category where SSH had a reduced memory usage was in 2-Ethernet-TOE-Enabled, consuming 6,416 KB compared to 7,096 in other test categories.

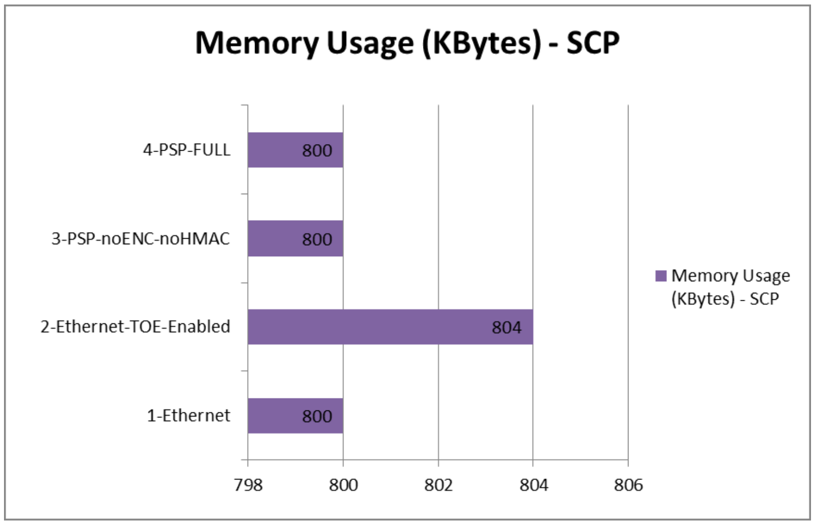


Figure 6.38. SCP Memory Usage

SCP memory usage was the same on each test category. As depicted in Figure 6.38, the only difference was in 2-Ethernet-TOE-Enabled, where it used 4 KB of memory more than the memory used in the other test categories.

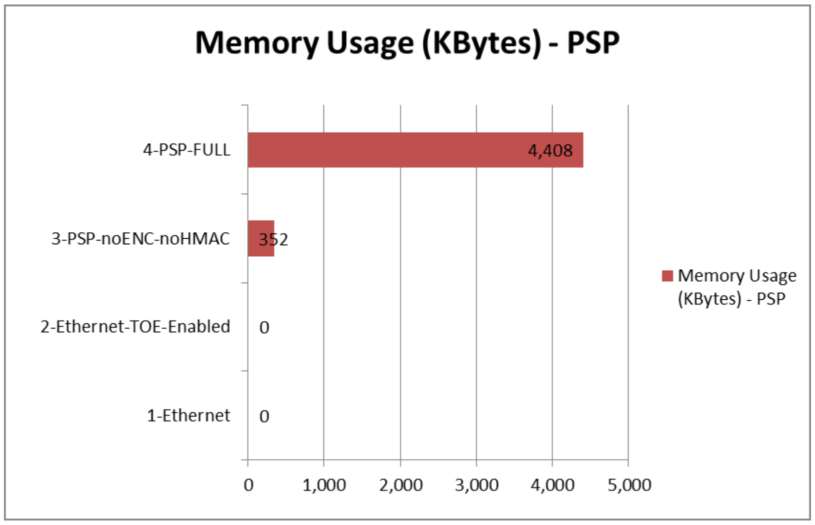


Figure 6.39. PSP Memory Usage During the SCP Test

As depicted in Figure 6.39, PSP memory usage on test category 3-PSP-noENC-noHMAC was only 352 KB; however, just as we observed on the Iperf TCP and UDP tests, memory used by PSP increased to more than 4,400 KB on test category 4-PSP-FULL, which is a direct result of the encryption/decryption and calculation of the HMAC values done by our developed program.

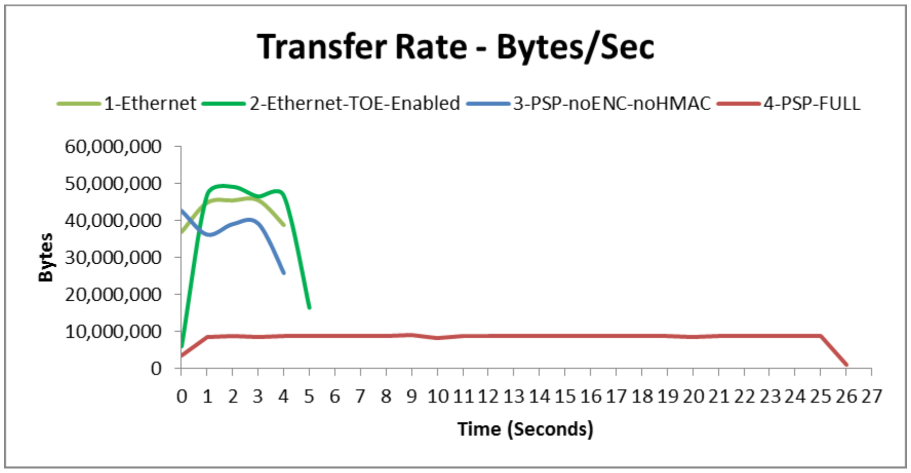


Figure 6.40. SCP Transfer Rate in Bytes/Sec

SCP transfer rate is depicted in Figure 6.40 and is very similar to the pattern observed in the Iperf TCP and UDP tests. While test categories 1-Ethernet, 2-Ethernet-TOE-Enabled and 3-PSP-noENC-noHMAC presented a variable transfer rate over time, SCP transfer rate on test category 4-PSP-FULL presented almost no variation at all.

6.2.6.1. Round Trip Time

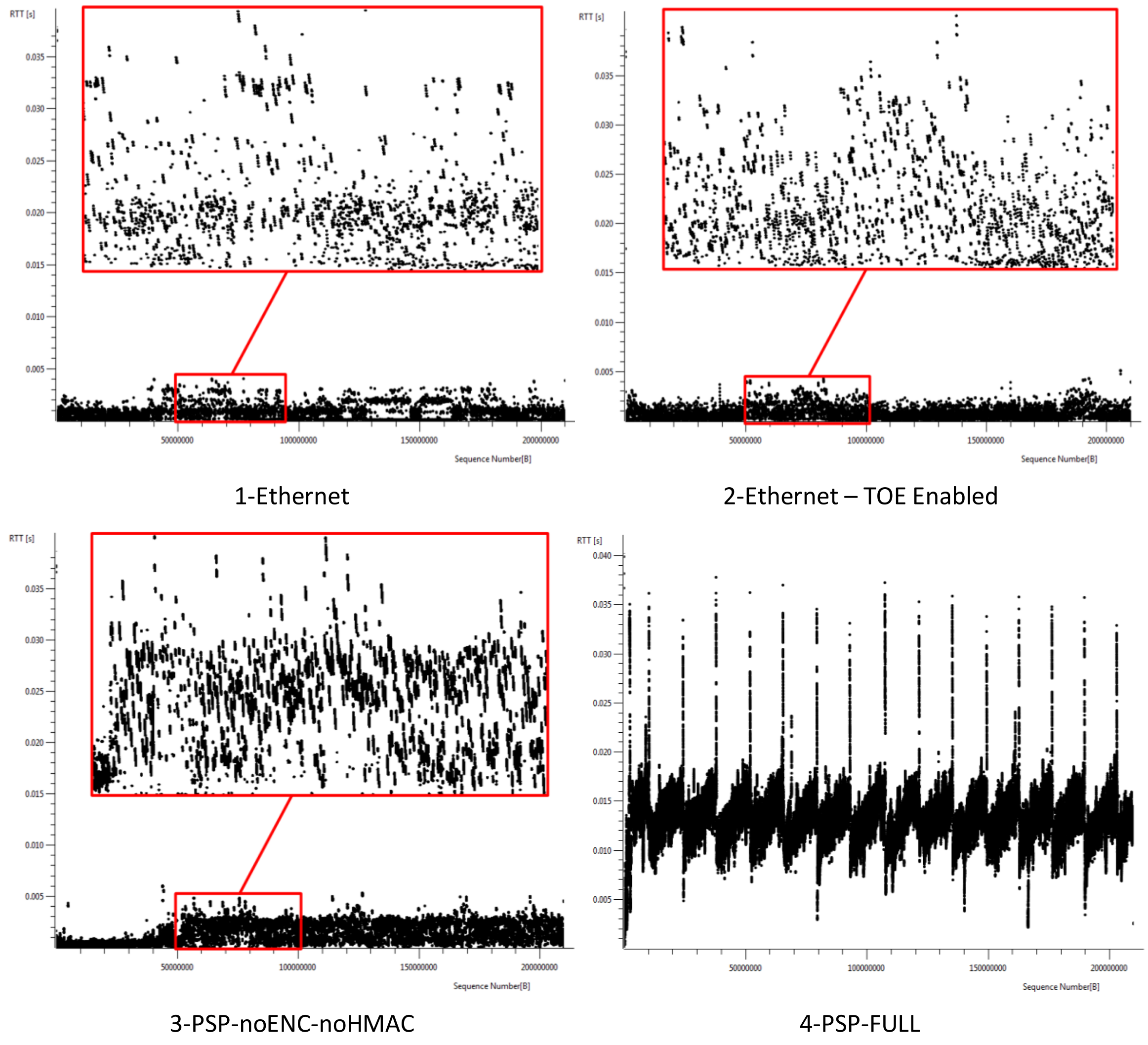


Figure 6.41. SCP Round Trip Time

The SCP Round Trip Time for each test category can be observed in Figure 6.41. Test category 4-PSP-FULL presented a repetitive pattern of long-vertical lines every certain amount of transmitted packets. The same pattern was observed on the same test category in the Ipert-TCP RTT graph (Figure 6.23).

Just as explained in section 6.2.4.1, the long-vertical lines represent sequential duplicate ACK packets sent by, in this case, the SCP server because some TCP packets were lost during the transmission.

Table 6.2. Duplicate ACK Packets on Each Test Category for SCP

	Captured Packets	**Duplicate ACK Packets**	**Percentage of Duplicate ACK Packets**
1-Ethernet	23,320	0	0%
2-Ethernet-TOE-Enabled	24,677	0	0%
3-PSP-noENC-noHMAC	132,133	228	0.173%
4-PSP-FULL	223,188	1695	0.759%

Interesting information about duplicate ACK packets is shown in Table 6.2. There were no duplicate ACK packets in test categories 1-Ethernet and 2-Ethernet-TOE-Enabled; however, we had some in test categories 3-PSP-noENC-noHMAC and 4-PSP-FULL. What is really important about the information disclosed in this table, is the percentage of duplicate ACK packets observed in test category 4-PSP-FULL (0.759%), which is close to the percentage of duplicate ACK packets (Table 6.1) under the same test category in the Iperf-TCP test (0.846%).

According to these statistics, we can conclude that under high network usage and intensive use of the CPU, our developed program will always present the same repetitive pattern of long-vertical lines every certain amount of transmitted packets, and approximately the same percentage of duplicate ACK packets. We can also conclude that this behavior of repetitive patterns is not related to the proposed Packet Security Protocol; but to a deficiency in the developed program to better balance the use of the CPU. We support this last conclusion with the fact that test category 3-PSP-noENC-noHMAC, as observed in Figure 6.41, didn't present the repetitive pattern reflected in test category 4-PSP-FULL, contrary to what occurred on both test categories in Figure 6.23 of the Iperf-TCP test.

6.2.6.2. Time/Sequence

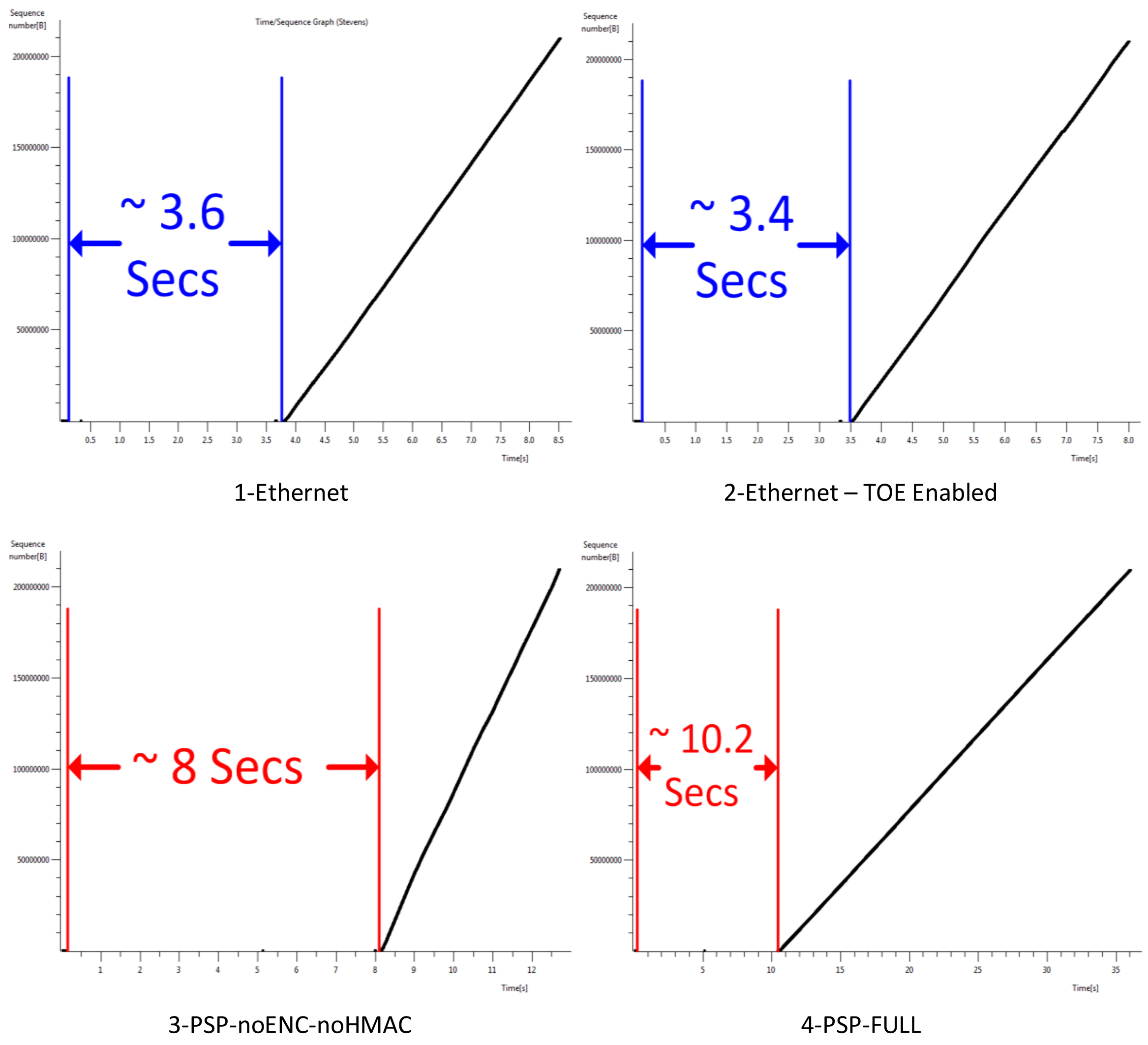


Figure 6.42. SCP Time/Sequence

The SCP Time/Sequence graph showed a very stable transmission, as depicted in Figure 6.42, but what we want to explain is the gap shown in the graphs of each test category.

First off, let’s give a general explanation of the transmission process before going into specific details.

The first step in the transmission of data using the SCP protocol is the establishment of a SSH session between a client and a server. In the process of establishing this SSH session, the client and the server inform each other about the SSH version that both can use, exchange session settings like encryption and compression algorithms, and agree on a session key to encrypt the rest of the communication. In the case of SSHv2, once the session key has been established the protocol can, as an optional feature, do a reverse DNS lookup to map the IP address of the client to a hostname and confirm that the hostname can resolve back to the IP address of the client [44].

In our test scenario, we used OpenSSH and by default, it comes with the reverse DNS lookup option enabled "UseDNS yes" [44]. The server sent a query to the multicast DNS address 224.0.0.251 asking for the FQDN of the ip address of the client; however, we didn't set up a DNS server to respond to reverse DNS queries. As a result, there is a long gap right after the session key agreement, as it is shown on the time/sequence graphs of each one of the test categories in Figure 6.42. Once the reverse DNS queries timed out, and the client-server authentication process was completed, the SCP protocol started up the transmission of data.

Now that we understand the reason of the gap in Figure 6.42, we can analyze its specific details in each test category.

Test categories 1-Ethernet and 2-Ethernet-TOE-Enabled had a gap of around 3.5 seconds. That is the time it took to SSH to try to resolve the client ip address to a hostname; however, it took longer on test categories 3-PSP-noENC-noHMAC and 4-PSP-FULL to time out after not receiving any reverse DNS resolution (approximately 8 and 10.2 seconds respectively). Even after repeating this test several times, we always got similar results.

We have seen that the more network traffic is protected using the proposed Packet Security Protocol, the higher the CPU usage is. As a result, data transmission takes longer than if it were not protected with PSP; however, in the case of a revere DNS lookup, the CPU usage is insignificant and there is no apparent reason for the SSH protocol to take too long before timing out, as in the case of test categories 3-PSP-noENC-noHMAC and 4-PSP-FULL. We couldn't find a reason for this behavior.

We wanted to see the behavior of the SCP transmission protected by our PSP, but without the delay added by the SSH option of doing a reverse DNS lookup. For this purpose we ran a new test modifying the configuration of OpenSSH "UseDNS no" to stop using DNS. The result of the test is shown in Figure 6.43, with no evidence of lost packets or significant time/sequence variation.

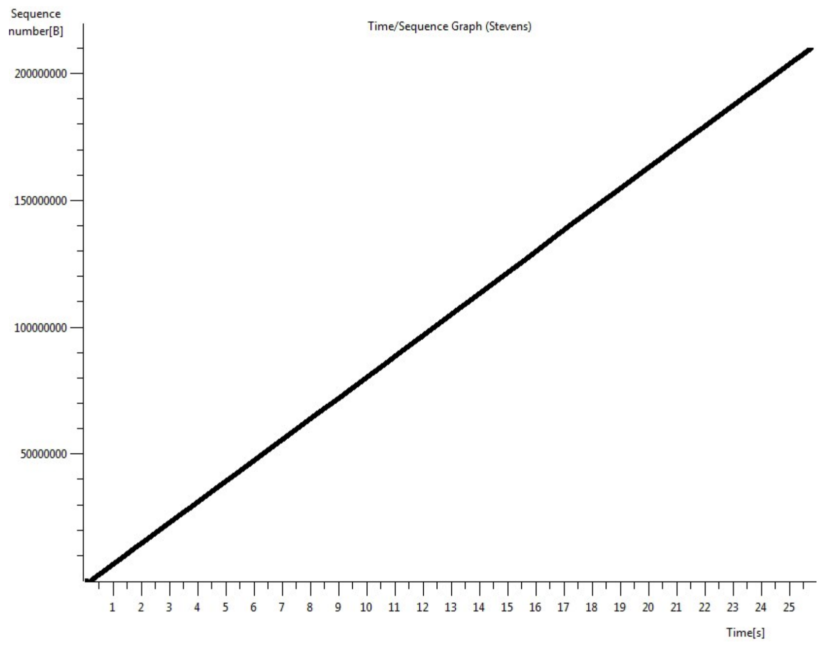


Figure 6.43. SCP Time/Sequence Graph in Test Category 4-PSP-FULL with no Reverse DNS Lookup

6.2.6.3. Throughput

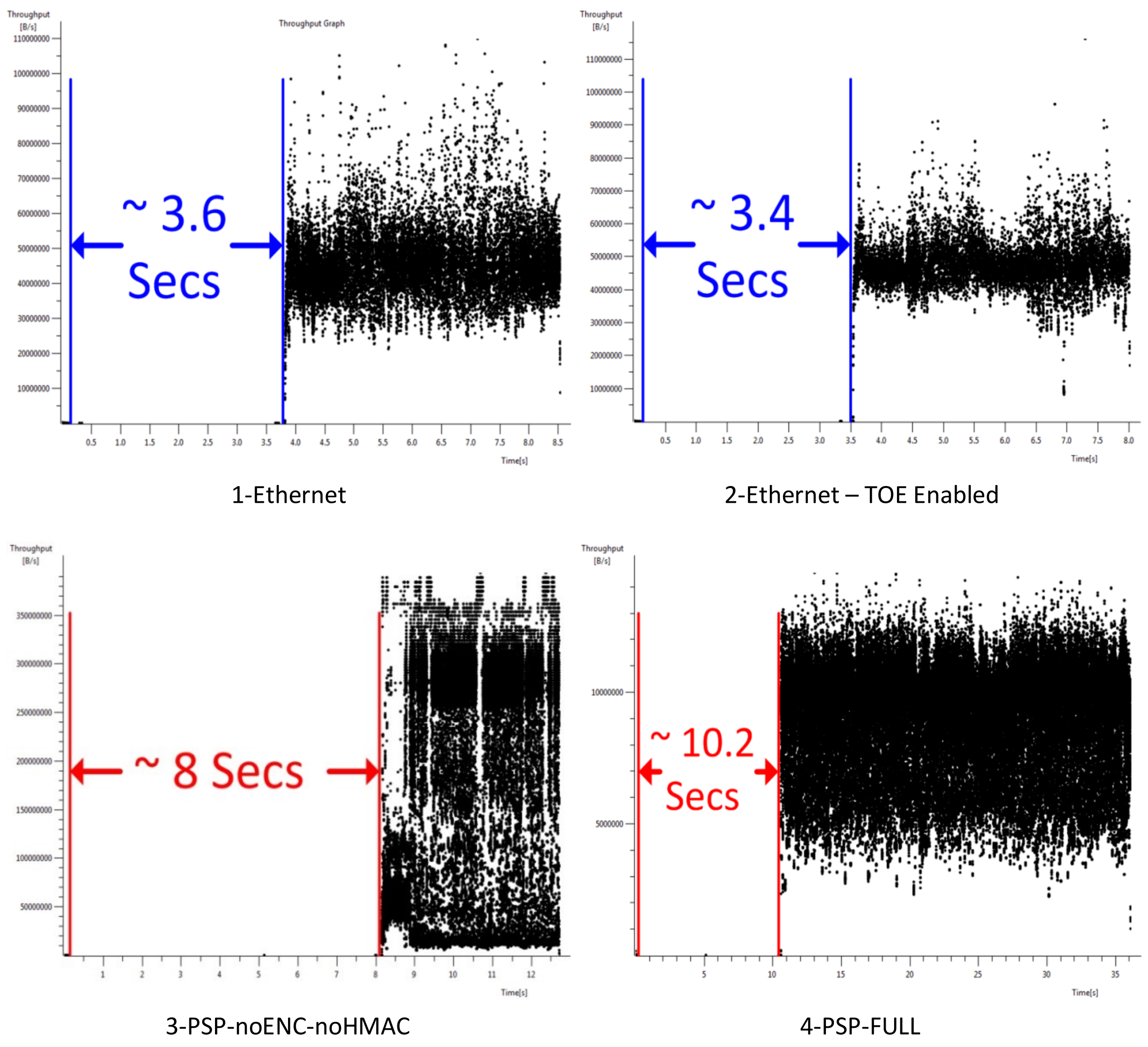


Figure 6.44. SCP Throughput

Figure 6.44 shows the SCP throughput. It remained stable in all test categories and can be considered normal, with no anomalies observed.

6.2.6.4. TCP Window Scaling – Server Side

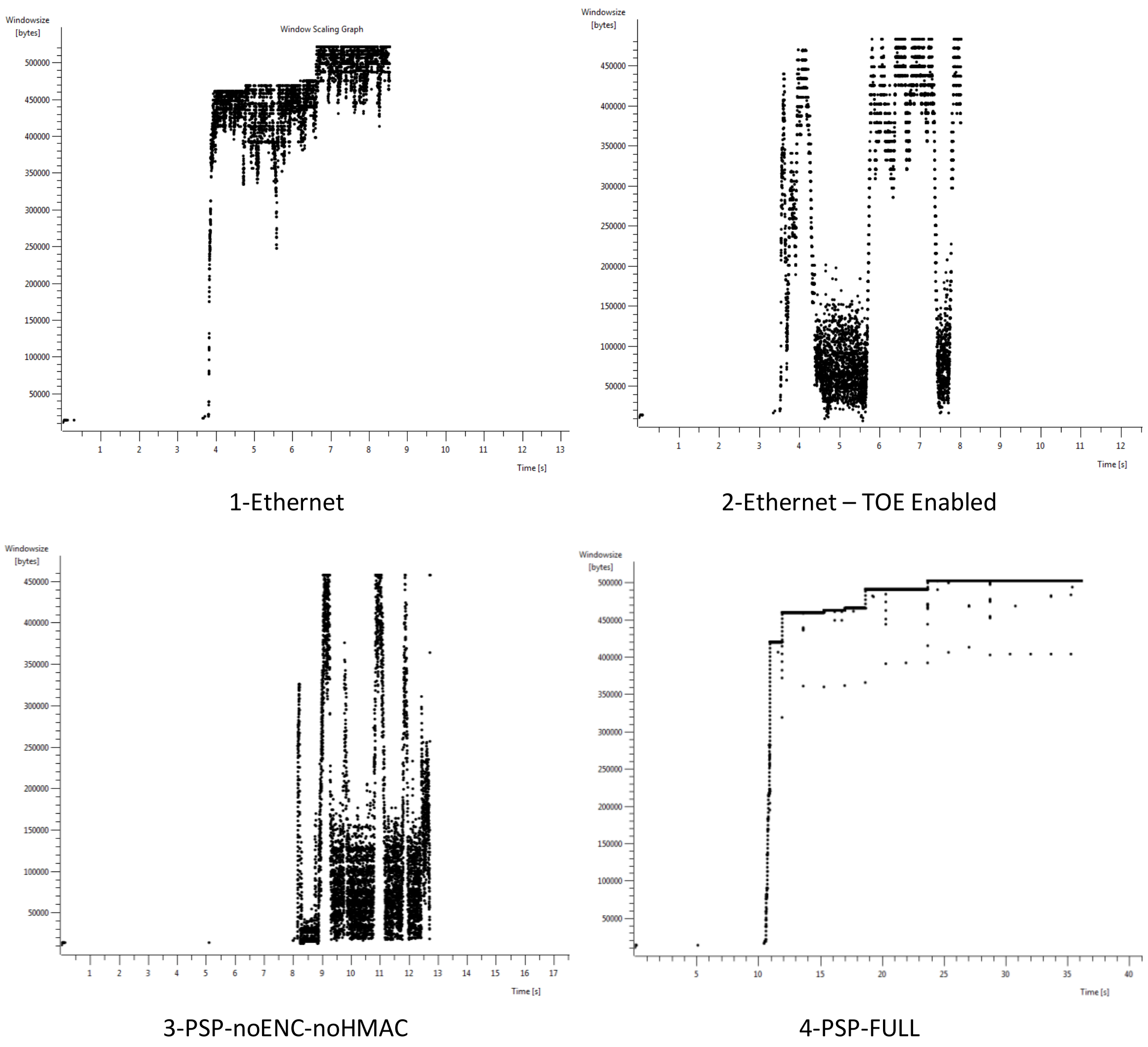


Figure 6.45. SCP TCP Window Scaling – Server Side

The server's TCP window scaling of all test categories is depicted in Figure 6.45. Test category 4-PSP-FULL reached a maximum TCP receiving window size of 500,000 bytes. This value is higher than size reached by test categories 2-Ethernet-TOE-Enabled and 3-PSP-noENC-noHMAC, with a maximum TCP receiving window size of 480,000 and 460,000 bytes respectively. Test category 1-Ethernet reached a maximum TCP receiving window size of 520,000 bytes.

Test categories 1-Ethernet, 2-Ethernet-TOE-Enabled and 3-PSP-noENC-noHMAC presented high variations in the size of the TCP receiving window, and were more inconsistent in test categories 2-Ethernet-TOE-Enabled and 3-PSP-noENC-noHMAC; however, test category 4-PSP-FULL presented a very stable window size with just a few variations.

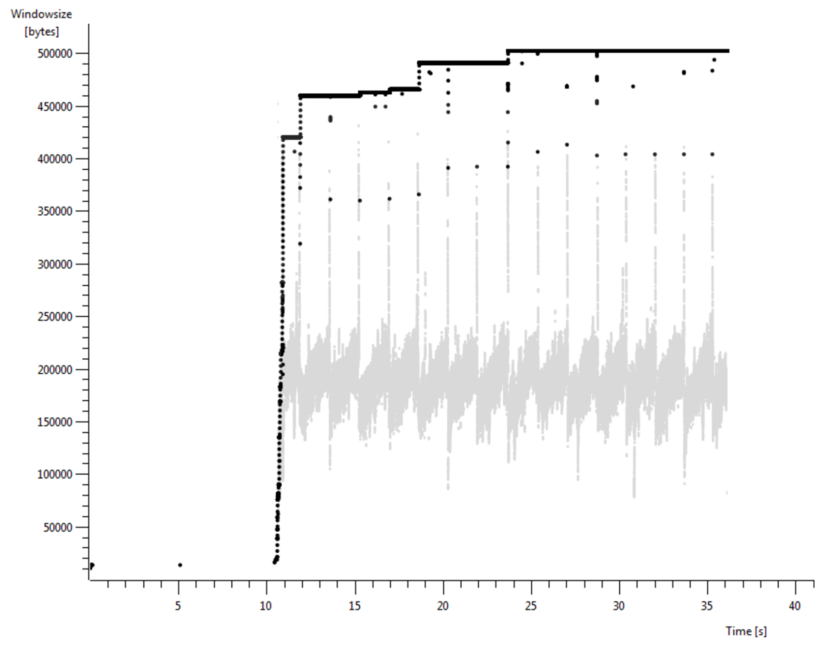


Figure 6.46. SCP Window Scaling – Server Side vs. RTT of Test Category 4-PSP-FULL

The sudden drop of the TCP receiving window size observed in test category 4-PSP-FULL as small dots, is consistent with the long vertical lines observed in the Round Trip Time graph of test category 4-PSP-FULL in Figure 6.41, as it is shown in Figure 6.46. This same behavior was observed in the Iperf-TCP test.

6.2.7. Conclusions of the Performance Analysis

The tests we ran to different network protocols showed interesting results based on their bandwidth utilization. The Packet Security Protocol presented a low processing overhead on the hosts running the protocol while protecting ARP, ICMP, VoIP and Skype communication. Network performance was not affected at all. The low transfer rates of these protocols didn't put too much effort on the developed program and helped to keep the CPU usage at low percentages. PSP can perfectly protect any low bandwidth communication without affecting network or CPU performance.

The results were completely different on protocols that demanded as much bandwidth as possible. Network transfer rate and CPU performance was highly impacted on the Iperf-TCP, Iperf-UDP and SCP tests because of the extra processing required by the cypher algorithms. High network traffic protected with PSP will result in high CPU usage due to the fact that the developed application will have to encrypt/decrypt data and calculate HMAC values at a very high speed.

Test results also showed that TCP Offloading Engines used in today's network cards will be rendered useless in a TCP communication protected with PSP, unless the firmware of these engines can be updated to use our proposed protocol.

7. CONCLUSION

IP communication in a wired Ethernet network can be compromised if the EtherType field of an Ethernet frame, the IP header or the IP payload is transmitted unencrypted or without integrity protection. In order to protect the entire IP communication in these networks, it is necessary to increase security at the OSI's layer 2 level by offering confidentiality protection to the EtherType field of the Ethernet frame and the encapsulated IP packet, and integrity protection to the entire Ethernet frame.

In this thesis, we proposed a new layer 2 protocol called Packet Security Protocol (PSP) to protect the confidentiality, integrity and authenticity of the IP communication in a wired Ethernet network. Our proposed solution was designed for flexibility, allowing the use of multiple encryption and hashing algorithms as well as multiple digital certificate standards.

We ran several tests to analyze the security characteristics and performance impact of our proposed solution; the test results demonstrated that the Packet Security Protocol can effectively protect the network communication between two hosts and avoid network attacks based on packet injection, data tampering or information disclosed not only by IP packets, but by any protocol encapsulated into the layer 2 frames because the original EtherType field and the layer 3 data is encrypted, and the entire Ethernet frame is integrity protected. It was also demonstrated that the choice of a vulnerable cipher increases the chances of breaking the confidentiality protection offered by our protocol, as was the case of our experiments with a Block cipher in ECB mode. The test results also showed that high network traffic protected with our solution can result in reduced network transfer rates and high CPU utilization due to the processing overhead imposed by the encryption/decryption of data and the integrity check calculation. Our findings also indicated that TCP Offloading Engines used in today's network cards will be rendered useless in a TCP communication protected with PSP, unless the firmware of these engines can be updated to use our proposed protocol.

Applying security to data communication in a Network environment can result in a trade-off that is mostly reflected in increased network administrative tasks, money investment and reduced network or host performance. Security must be thoroughly analyzed based on the real needs of the business and the different information security risks it can be exposed to.

7.1. Future Work

In this thesis, we developed a computer program in C++ to demonstrate the proof of concept of our proposed Packet Security Protocol; however, the program code is not optimized for performance and depends 100% on third party crypto libraries to perform cryptographic functions. It was also designed to be executed in user space, which also limits the performance of the program. Future work on this topic includes the following:

- Review the program source code and optimization for network and system performance. This optimization includes the customization of third party crypto libraries or the design of a new one that can be fully integrated and adjusted to cover the crypto purpose of the program.
- More work must be done in the development of the PSP Session Establishment Protocol and support for Digital Certificates.
- The computer program should be designed to be run also in Kernel space.
- We proved the concept of Packet Security Protocol in a Linux Operating System. A new computer program must be developed to be used as a fully functional implementation of our solution in all operating systems.

The Packet Security Protocol needs to be tested in existing network hardware, such as Network Routers and Layer 3 Switches. An interesting approach to achieve this goal could be the use of Software Defined Networking and OpenFlow.

Future work also includes doing more research on the behavior of certain protocols protected with our solution. During our tests, SSH presented a longer Reverse DNS lookup delay when protected with PSP compared to the regular Reverse DNS lookup delay when SSH was not protected with our solution. The results of this test were explained in section 6.2.6.2, but we couldn't find the reason for this behavior.

Even though, we designed the Packet Security Protocol to protect wired Ethernet networks, we believe that the concept is perfectly exportable to protect data communication over other layer 2 protocols.

Much more work related to our proposed solution will result in the future, but it will depend on the success of its implementation and the interest presented by other researchers.

REFERENCES

[1] Kozierok, Charles M. "The TCP/IP Guide." 236. San Francisco, CA: No Starch Press, 2005.

[2] Kozierok, Charles M. "The TCP/IP Guide." 330. San Francisco, CA: No Starch Press, 2005.

[3] Loshin, Pete. "TCP/IP Clearly Explained." 551. San Francisco, CA: Morgan Kaufmann Publishers, 2003.

[4] Loshin, Pete. "TCP/IP Clearly Explained." 335-338, 551-580. San Francisco, CA: Morgan Kaufmann Publishers, 2003.

[5] Hardjono, Thomas, and Lakshminath R. Dondeti. "Security in Wireless LANs and MANs." 131-141, 143-154. Norwood, MA: Artech House, 2005.

[6] Meyer, G. "The PPP Encryption Control Protocol (ECP)." RFC 1968. June 1996.

[7] Tu, Kwei. "Communications Link Layer Security." ICCT 2006. International Conference on Communication Technology. November 2006. 1-4, 27-30.

[8] IEEE Std 802.1AE. "IEEE Standard for Local and Metropolitan Area Networks: Media Access Control Security." IEEE, August 2006.

[9] Jerschow, Yves Igor, Christian Lochert, Björn Scheuermann, and Martin Mauve. "CLL: A Cryptographic Link Layer for Local Area Networks." In Security and Cryptography for Networks, 21-38. Springer Berlin Heidelberg, 2008.

[10] Kent, S. "Security Architecture for the Internet Protocol." RFC 2401. November 1998.

[11] Sosinsky, Barrie. "Networking Bible." 23-34. Indianapolis, IN: Wiley Publishing, Inc, 2009.

[12] Paterson, Kenneth G., and Arnold K. L. Yau. "Cryptography in Theory and Practice: The Case of Encryption in IPSec." In Advances in Cryptology - EUROCRYPT 2006, 12-29. Springer Berlin Heidelberg, 2006.

[13] Nikov, Ventzislav. "A DoS Attack Against the Integrity-Less ESP (IPSec)."

[14] Kummert, H. "The PPP Triple-DES Encryption Protocol (ECP)." RFC 2420. September 1998.

[15] Pall, G., and G. Zorn. "Microsoft Point-to-Point Encryption (MPPE) Protocol." RFC 3078. March 2001.

[16] Fluhrer, Scott R., Itsik Mantin, and Adi Shamir. "Weaknesses in the Key Scheduling Algorithm of RC4." Revised Papers from the 8th Annual International Workshop on Selected Areas in Cryptography (SAC '01). London, UK: Springer-Verlag, 2001. 1-24.

[17] IEEE Std 802.1X-2010. "IEEE Standard for Local and Metropolitan Area Networks: Port-Based Network Access Control." IEEE, February 2010.

[18] Ballew, Scott M. "Managing IP Networks with Cisco Routers." 53-57. O'Reilly Media, 1997.

[19] Taylor, Ian J., and Andrew B. Harrison. "From P2P and grids to services on the web: evolving distributed communities." 29. Springer-Verlag London Limited, 2009.

[20] Ciccarelli, Patrick, and Christina Faulkner. "Networking Foundations: Technology Fundamentals for IT Success." 16-29. Alameda, CA: Sybex, 2004.

[21] Miller, Philip M. "TCP/IP - The Ultimate Protocol Guide: Volume 1 - Data Delivery and Routing." 67-71. Boca Raton, FL: BrownWalker Press, 2009.

[22] Plummer, David C. "An Ethernet Address Resolution Protocol." RFC 826. November 1982.

[23] Sosinsky, Barrie. "Networking Bible." 437-440. Indianapolis, IN: Wiley Publishing, Inc, 2009.

[24] Wireshark Foundation. About Wireshark. http://www.wireshark.org/about.html (accessed May 2013).

[25] Iperf. http://iperf.fr/ (accessed May 2013).

[26] OpenBSD. OpenSSH. http://www.openssh.org/ (accessed May 2013).

[27] Digium, Inc. Get Started. http://www.asterisk.org/get-started (accessed May 2013).

[28] SIPp. http://sipp.sourceforge.net/ (accessed May 2013).

[29] Skype. About Skype. http://www.skype.com/en/about/ (accessed May 2013).

[30] Ettercap Project. About. http://ettercap.github.io/ettercap/about.html (accessed May 2013).

[31] Kippo. Project Home: Summary. https://code.google.com/p/kippo/ (accessed September 2013).

[32] Mauerer, Wolfgang. "Professional Linux Kernel Architecture." 745. Indianapolis, IN: Wiley Publishing, Inc, 2008.

[33] Mauerer, Wolfgang. "Professional Linux Kernel Architecture." 7-10. Indianapolis, IN: Wiley Publishing, Inc., 2008.

[34] Minghao, Koh, Khong Yun Chyang, and Ettikan Kandasamy Karuppiah. "Performance Analysis and Optimization of User Space versus Kernel Space Network Application." The 5th Student Conference on Research and Development – SCOReD 2007. Selangor, Malaysia: IEEE, 2007. 1-6.

[35] Universal TUN/TAP Device Driver. https://www.kernel.org/doc/Documentation/networking/tuntap.txt (accessed March 2012).

[36] Gregory, Peter. "CISSP Guide to Security Essentials." 163-164. Boston, MA: Course Technology, 2010.

[37] McCubbin, Christopher B., Ali Aydin Selcuk, and Deepinder Sidhu. "Initialization Vector Attacks on the IPsec Protocol Suite." IEEE 9th international workshops on enabling technologies: infrastructure for collaborative enterprises (WET ICE 2000). Gaithersburg, MD: IEEE, 2000. 171-175.

[38] Dworkin, Morris. "Recommendation for Block Cipher Modes of Operation." 8. NIST Special Publication 800-38A 2001 Edition, 2001.

[39] Ethtool. http://linux.die.net/man/8/ethtool (accessed May 2013).

[40] Walker, John Q., and Jeffrey T. Hicks. "Taking Charge of Your VoIP Project." 90. Indianapolis, IN: Cisco Press, 2004.

[41] Sanders, Chris. "PRACTICAL PACKET ANALYSIS: Using Wireshark to Solve Real-World Network Problems." 81. San Francisco, CA: No Starch Press, 2011.

[42] Kozierok, Charles M. "The TCP/IP Guide." 782. San Francisco, CA: No Starch Press, 2005.

[43] Barrett, Daniel J., Richard E. Silverman, and Robert G. Byrnes. "SSH, The Secure Shell: The Definitive Guide." 82-84. Sebastopol, CA: O'Reilly Media, 2005.

[44] Barrett, Daniel J., Richard E. Silverman, and Robert G. Byrnes. "SSH, The Secure Shell: The Definitive Guide." 158-159. Sebastopol, CA: O'Reilly Media, 2005.