Secure Military Networks: Black or Red?

Joana Afonso
Eletrical and Computer Engineering
Instituto Superior Técnico

***Abstract*—All sensitive information sent over the Internet and other unsafe networks should be safeguarded to prevent compromising the data and the entities involved in its transmission. This is especially true when discussing sensitive military data sent outside the limits of a military network. Military networks employ the Red/Black architecture concept, which divides the network into two distinct zones. The Red zone is deemed safe, while the Black zone is considered risky.**

This paper introduces an adapted version of the ESP protocol, from the IPsec suite, prepared to be used in military network devices separating the Red and Black zones. These devices consist of three parts: two packet Processing Units lacking direct communication, responsible for either sending or receiving packets, and an HSM. This HSM separates the two Processing units, and is responsible for the ESP cryptography tasks for packets passing through the device. The modified ESP protocol establishes a new method of distributing the original processing across the device's three components. This distribution can establish a processing pipeline, maintaining the security features provided by the standard version. The proposed solution operates with the AES-GCM encryption and authentication algorithm. It has been integrated into a C library, utilized to test the protocol with network packets.

***Index Terms*—IPsec, ESP, Security Protocols, Network Security, Encryption, Authentication**

I. Introduction

Military Networks are communication networks used by military forces to assure a secure exchange of information between military units. These networks are designed to be highly resilient in order to withstand both physical and logical attacks. They are also built using specialized devices and protocols that help secure sensitive information, usually by using cryptography in order to hide their true content.

Military networks make use of the Red/Black architecture concept, which refers to the division of the overall network into two distinct zones. A Red Zone, considered the internal and protected side of the network, where the classified information can be sent in a secure manner in plain text form, and a Black Zone, usually associated with the public network, where the classified information needs to be encrypted in order to being securely sent. To divide the network in these two areas there needs to be a separation device, hardware or software based, that is able to analyze and manipulate the traversing traffic in order to assure the security of the classified information.

The DISCRETION project, in which this thesis is part of, was first commissioned by the EU, with the goal of creating a device that can provide a separation point between the Red and Black sides of the network, for an Optical SDN solution that enables secure, robust and resilient communications for Europe Defense over the Internet, taking advantage of the high-security level and quantum resistance provided by QKD systems [1]. These devices will work as a separation node that stands in the intersection of a red and black zone in the network, and will be responsible for doing similar services to a router and a firewall, like segregating and forwarding packets, but above all provide the needed cryptographic operations in order to securely separate the red and black sides of the network.

The Devices in development can be divided into 3 separate parts: an IP processing unit located in the Red Side of the Network; an HSM device which will be responsible for encrypting and decrypting the packets that cross from one side of the network to the other; and another IP processing unit, this time located in the Black side of the network. The following Figure 1 illustrates this division.

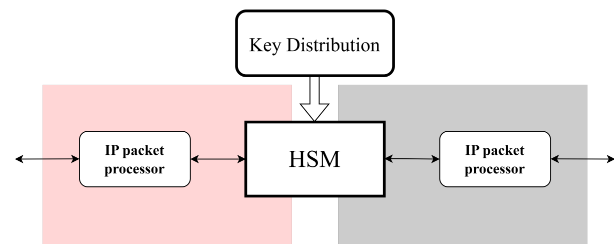

Fig. 1. Secure Device Architecture

In order to assure that the cryptographic services used inside the HSM are correctly protecting an Outbound packet, the device will also need to employ a security protocol that assures the secure traversing of the packet once it leaves the device, as well as guaranteeing that once an Inbound packet arrives, the device is able to correctly retrieve the original contents of the packet. The main goal with this dissertation is to adapt the Encapsulating Security Payload (ESP) protocol, part of the IPsec suite of protocols, in order to deploy in the device in question. This adaptation will focus of dividing and, if needed, altering the processing done to the both Outbound and Inbound packets, so the processing can be done correctly in all parts of the device.

A. Project Requirements

For the device's necessary security level, there are a set specific requirements for the adaptation and implementation done of the protocol that need to be respected and taken into account during this thesis project. These include:

- The data that is transported as plaintext in the Red Side of the Network must never be sent as plaintext into the Black Side. The only way that data can go into the Black Side is by being converted into cyphertext inside the HSM;
- There cannot exist any direct path for data inside the HSM from the Red Side to the Black Side, in order to prevent any leak of sensitive information;
- The keys can only be accessed inside the HSM and cannot be leaked onto either Red or Black side of the Network.

II. Background

A. Cryptography Concepts

In order to protect data being sent over public networks from possible attacks, security services have been developed and employed so that packets can safely traverse the internet: These security services consist of [2]:

- **Confidentiality** - This service ensures that secret information going through a network cannot be disclosed to unauthorized entities. This is commonly done by encrypting the private data with an encryption algorithm so that only the intended receiver is able to recover the data.
- **Integrity** - This service ensures that the data being transmitted in a network is not tampered with during its transmission and, therefore, can be trusted by the receiver.
- **Authentication**: This service ensures the entity validation of an endpoint sending data across a network. There are two types of authentication currently defined: peer entity authentication and data origin authentication.
- **Replay Protection** - This service guarantees the freshness of messages, thus ensuring that they cannot be replayed in the network. This is commonly done by the use of a timestamp or a counter.

B. Internet Protocol Security

The Internet Protocol Security (IPsec) is a suite of protocols that functions in the Network layer of the TCP/IP stack and that provides end-to-end authentication, integrity and confidentiality of data for secure sessions between two points in the network. IPsec is primarily composed of the following protocols [3]:

- **AH**, which provides replay protection, data integrity and data origin authentication;
- **ESP**, which provides confidentiality through encryption, data integrity, data origin authentication and replay protection;
- **IKE**, which negotiates the parameters and keys between two peers when establishing a connection.

IPsec is commonly used to create a VPN, either between two locations (gateway-to-gateway) or between a remote user and an enterprise network (host-to-gateway), though it can also be used to establish connections between hosts (host-to-host).

IPsec includes protocols for establishing mutual authentication between peers at the beginning of a session and negotiation of cryptographic keys to use during the session. Depending on the implemented protocols in a session, IPsec is able to provide network-level peer authentication, data origin authentication, data integrity, data confidentiality and replay protection.

One of the most important elements of the IPsec suite is the concept of Secure Association. A Secure Association is a logical ”connection” that applies security services to the traffic carried by it. The services each SA offers are defined based on the one protocol, and only one, in use (AH or ESP). If there is a need to use both protocols for a traffic stream, then two different SA's must be created, one for each protocol. These SA are also unidirectional so, in order to ensure the security a bi-directional traffic stream between two points in the network a pair of SA's is required, one for each direction [4].

Each SA that is used to carry uni-cast traffic can be identified by an unique value called SPI. The SPI is a 32-bit value that is attributed to each SA created by the destination system in order to identify the requirement and stipulations of each connection. This value is always carried inside the headers of both AH and ESP [4].

C. Encapsulation Security Payload

The Encapsulation Security Payload (ESP) is a protocol used to offer confidentiality, authentication, integrity and anti-replay protection to IP packets being transmitted over a network. This Protocol is commonly used when implementing VPNs and other types of secure channels as this services usually require most of the cryptographic services ensured by this protocol. An encryption algorithm is used to provide confidentiality and authenticity, and an authentication algorithm ensures the integrity of the packet. Unless otherwise indicated, the details provided in this section are sourced from RFC 4303 [5].

Every ESP packet created requires an Header and an Trailer, which will be equal in terms of fields and size for every packet, even considering packets transported in different modes. The Next Figure 2 details the format of an ESP packet, including the Header, Trailer and the fields that compose each one.

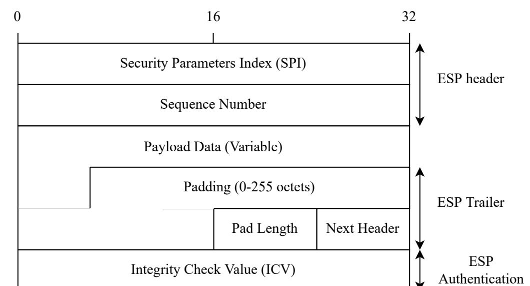

Fig. 2. ESP Standard Packet Structure

The ESP packet can contain up to a total of 8 fields (not all represented in the Figure above) that can be described as follows:

- **Security Parameter Index (SPI)**: Unique identifier that enables the receiver of a packet to process the packet correctly by indentifying the SA in use;
- **Sequence Number**: 32-bit counter that increases for each packet sent in the SA in use. This value is used in order to provide replay protection. Can also be used a an extended sequence number, where the counter has a total of 64 bits, with only the 4 least significant Bytes being carried in the packet.
- **Initialization Vector**: Field that is only present in the packet if the encryption algorithm in use requires a value of that kind.
- **Payload**: The Payload is a variable length field that contains the data being secured with the ESP protocol.
- **Padding**: The Padding field is included in the ESP packet in order to ensure that, when required, the data to be encrypted is a multiple of the block size used by the encryption algorithm.
- **Padding for Confidentiality**: Bytes that are added after the Payload in order to avert traffic flow analysis. The number of Bytes added as TFC Padding are not included in the Pad Length value.
- **Pad Length**: Field indicates how many Bytes of normal Padding (not TFC) were added in total to the original payload.
- **Next Header**: Field that indicates what type of data is being transported in the Payload of current the ESP packet.
- **Integrity Check Value**: The Integrity Check Value is a variable-length field that is computed over the ESP packet with an integrity and/or authentication algorithm (if this option is enabled) in order to protect the contents of the packet while traversing the network.

The ESP protocol can be employed in 2 different modes: Transport Mode or Tunnel Mode. As we will see both modes protect the original IP packet in different ways, which will then result in a different final IP packet structure for each one of the Modes.

In Transport Mode the ESP protocol is only applied to the contents of the original IP Packets, thus separating the Header from the Payload. The original IP header is maintained and re-used as the IP Header of the new packet created, while the original Payload is encrypted and encapsulated between the ESP header and trailer.

In Tunnel Mode the ESP protocol is applied to the whole IP packet, so that both the original header and payload are encrypted and encapsulated in a new ESP packet. Since the original IP header is encrypted, the ESP packet is then also encapsulated in a new IP header, which is most times different from the original IP header in order to hide its original contents.

The structure of ESP packets created in both Tunnel and Transport mode can be seen in Figure 3.

The existing RFC that establish the fundamental aspects of the ESP protocol also describes how the Packet Processing is

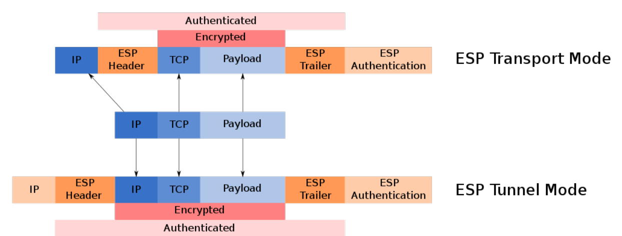

Fig. 3. ESP Transport Modes

supposed to be done for different ways that the protocol might be in use.

Once an IPsec implementation receives an outbound packet the first thing it needs to do use the relevant information inside the IP packet, like destination address, to search for an existing SA inside their databases in order to see if the packets requires ESP processing.

If an SA is found and the packet does require ESP processing the next thing to be done is to encapsulate the original IP packet into the ESP payload field. This action differs from transport mode to tunnel mode, as viewed previously. After this the ESP fields that are encrypted together with the payload are added, which include Padding (and optional TFC Padding), the Pad Length and Next Header fields.

Once the Plaintext is prepared we follow to the Cryptographic algorithms. If the defined SA uses separate Algorithms for Confidentiality and Integrity, the Encryption is done first and is then followed by the Integrity Algorithm. It's also important to point out that if an Integrity Algorithm is defined the Sequence Number is generated prior to its use, as it is one of the fields that takes part in the ICV calculation. If the defined SA uses a Combined Confidentiality and Integrity Algorithm then the specifics and needed information about the use of said protocol can be found in the RFC that defines the use of the combined algorithm with ESP.

After going through all the algorithms defined by the SA the ESP packet is then constructed in it's totality, with the correct header and ICV trailer, and is then encapsulated with the correct IP header, depending on the defined Mode.

When dealing with Inbound Packets, like what is done for Outbound Packets, the first thing done is to look-up, using the SPI, the appropriate SA stored in its database. If no valid SA is found then the received packet must be discarded.

If an SA is found based on the SPI, then, the first thing to do is verify if the Sequence Number carried in the ESP header is a valid one. Following this the ICV is then verified based on the algorithm in use. If the SA defines separate Confidentiality and Integrity Algorithms then the ICV is verified first, and, in case this value is correct, the Ciphertext is then decrypted with the defined Encryption Algorithm. If the SA defines a Combined Confidentiality and Integrity Algorithms then the ICV is verified and the Ciphertext is decrypted based on what is specified in the RFC that defines the use of the combined algorithm with ESP. If the ICV verification fails then the

packet is discarded.

After verifying the ICV and decrypting the Ciphertext the ESP implementation should inspect the Padding field before removing it and then proceed to check the Next Header field. After removing the fields added to the ESP payload the receiver is able to reconstruct the original IP datagram.

III. State of the Art

We will now introduce some of the already known and used software implementations of the ESP protocol, and try to understand if we would be be to directly use any of them in the device being developed.

A. StrongSwan

One of the first implementations we researched and studied when starting working in this project was the StrongSwan software. StrongSwan is a open source implementation of the IPsec protocol, based on the standards defined by the IETF, that was launched in 2005 as a fork of the now discontinued FreeSwan project [6].This implementation is mostly written in C and can be employed in various operating systems, including Linux, Windows and Android.

B. DPDK

DPDK, or Data Plane Development Kit, is an open-source project, started in 2010, that consists of libraries that are used to provide high-performance packet processing, upon which its users are able to build and run data plane applications [7]. Like StrongSwan, DPDK is mainly developed in the C language and also allows for multi threading processing.

C. Scapy

Scapy is an open source packet manipulation library developed in python that is able to craft and capture from the network packets of various protocols, as well as analyse network traffic [8].This software allows us to create analyse various types of packets, including ESP and AH packets. Although this open source software is not able to set up IPsec connections in a real network infrastructure, as it does not implement the IKE protocol, with Scapy we are able to easily analyse the needed processing done in order to create and send IPsec packets, as well as the processing done when receiving the same packets.

All of these three implementations mentioned are implemented according to what is established in all the RFCs that compose the IPsec protocol, and, because of this, it is not possible for us to directly use them as a way to process the packets in the device in development. Despite this, we are still able to use sections of these implementations in order to better understand certain parts of the processing done to the packets, and even use some of the logic presented in our own implementation.

IV. Proposed Solution and Implementation

A. Proposed Solution

As it was previously mentioned, the goal of this project is to adapt the ESP protocol in tunnel mode in order to protect the packets traversing the device in development.

The ESP implementation to be made will also be developed according to the use of the AES-GCM encryption and authentication algorithm with this protocol. As it was explained in the previous sections the details of ESP processing can change depending on the algorithms in use with the protocol. These changes are usually defined in a separate RFC for each algorithm and protocol in use. As the cryptographic algorithm chosen by the colleague responsible for the HSM cryptographic core implementation was the AES-GCM, this implementation will be done according to this choice.

Our goal with this project is to create an adapted implementation of the known IPsec ESP protocol, in use with the GCM algorithm, in order to protect packets that traverse the device previously described in the Introduction chapter. In order to have the implementation work with the device in development we will have to change and adapt some of the protocol characteristics, most importantly: Outbound and Inbound processing flow, databases, data flow and packet structure. As this implementation is done in order to perfectly obey the requirements set, most of the changes made will go against the standard defined, which means this implementation cannot be used with others available.

It is also important to point out that, since this project focuses solely on the adaptation and implementation of the ESP protocol, throughout the development of the proposed architecture and implementation we always used pre-established SAs in order to evaluate our progress and test each version of the implementation.

B. ESP Protocol with AES-GCM

The Galois/Counte Mode (GCM) is a cryptographic mode of operation for symmetric block ciphers that provides both confidentiality and data origin authenticity. This algorithm operates in counter mode with blocks of 128 bits and has a total of four inputs: a secret key, an Initialization Vector (IV), a plaintext, and an input for AAD, which is data that is meant to only be authenticated and not encrypted by the gcm algorithm. With these four inputs the algorithm then produces two outputs, a ciphertext with length equal to the plaintext, and an authentication tag [9]. RFC 4106 defines the use of this algorithm with the ESP protocol and the next information provided in this section can be referenced to this RFC [10].

The **secret key** used by the algorithm to encrypt the plain text is the one defined when both entities are creating an SA. When using the GCM algorithm this key is composed of a total of 256 Bytes.

The **plaintext** that is sent to be encrypted is formatted in the same way that it was defined for the ESP protocol, which is the ESP payload together with the Padding, Pad Length and Next Header fields. The one thing that might change is

the total amount of Padding added, since the value depends on the data alignment desired. Since GCM encrypts in blocks of 128 bits, the Padding will be added in order to achieve a natural number of 128 bits blocks.

The **Initialization Vector** defined for the GCM algorithm is an 12 Byte field that is created by combining an 8 Byte IV and an 4 Byte Salt value. This 4 Byte Salt value is different for every SA and is created and stored at the same time as the secret key. In the context of GCM-ESP this combination is often referred to as the nonce, and the IV name is used only when referring to the 8 Byte value. The IV is a value that is stored in the side of the sender and is also sent as part of the payload of the ESP packet. This value is incremented by one for every packet sent and cannot be repeated when using the same key. The salt value is stored in both the sender and received sides, and is setup at the same time as the secret key. This value is not sent inside the ESP packet, as it is stored in both sides of the SA, but, unlike the key, it is not considered a secret and can be shared via a public channel.

The **Additional Authenticated Data** (AAD) in GCM corresponds to data that is only supposed to be authenticated by the algorithm, not encrypted. In the case of the ESP protocol the AAD will be a combination of the SPI followed by the Sequence Number (or Extended Sequence Number), which are the fields that make up the header of the protocol.

With this four inputs the algorithm is then able to produce the final two outputs needed to form the ESP packet. The **Cipher Text** corresponds to the encrypted plain text described before, and the **Authentication Tag** (or Integrity Check Value,) which has a total of 16 Bytes and is transported as the trailer of the ESP packet.

The format of an ESP packet created when using the AES-GCM algorithm can be seen in the following Figure 4.

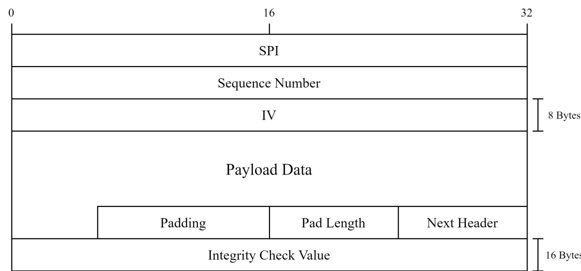

Fig. 4. ESP Header Using GCM Algorithm

C. Standard Alterations and Final Packet

During the development of the ESP protocol's adapted architecture, the structure of the standard ESP packet was modified for optimal efficiency and to allow a better adaptation for the desired architecture.

The main changes involved the Sequence Number and Initialization Vector (IV). As both values can be represented by 8 Byte fields (if using an Extended Sequence Number), and both are processed equally by the ESP protocol (are incremented by 1 for every packet sent), it was decided to merge these fields into a unique 8 Byte field.

The only established way to combine these two values is described in RFC 8750, and it defines a way to use an implicit IV based on the Extended Sequence Number in use. The Extended Sequence Number is used in the way it is defined, with the packet transporting the least significant 4 Bytes of the value, while the two endpoints keep track of the most significant Bytes [11].

Although this methods would reduce the total amount of data being added to the packet by the ESP header, it could also lead to some difficulties when establishing the protocol architecture as, depending on where these values are stored in the final version of our implementation, we may have difficulties in correctly keeping the track of these values in both ends of a connection.

To guarantee that there would be no need to keep track of the 4 most significant Bytes of the Sequence Number/IV value, it was decided that we would still employ the method described with one alteration. This alteration consist on, instead of only sending part of the Sequence Number/IV inside the packet, sending also the 4 most significant Bytes of this values as part of the payload, like what is when using separate values for the Sequence Number and IV.

With these changes we are also still able to keep the security services offered by the Sequence Number, as this value is still computed and verified according to what is defined in RFC 4303, while reducing the overall quantity of extra data being added to each final ESP packet by 4 Bytes. The final ESP packet created is depicted in Figure 5, where we can see how all the final fields are organized in the packet. The fields greyed out correspond to the fields being encrypted by the GCM algorithm.

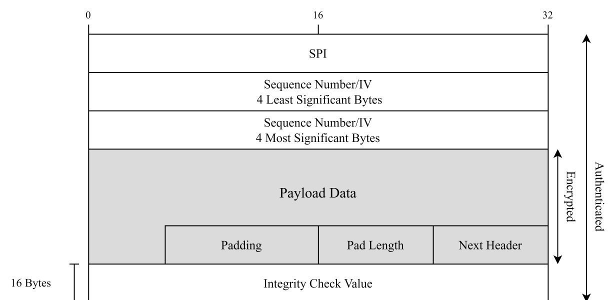

Fig. 5. Final Non-Standard ESP Packet

D. Adapted ESP Processing

After analysing the processing logic that is defined by the standard RFCs, we started adapting the processing flow of a normal ESP implementation, in order to be used in the developing device.

Almost all of the IPsec processing done to the packets will be done in the two Packet Processing Units, located in each side of the HSM. As the HSM is responsible for the encryption algorithm, our goal is to successfully separate and organize the

steps required to create an ESP into two different parts: what needs to be done before encryption and what needs to be done after.

Most of the processing done can be easily divided into the three parts. For outbound packets, the Packet Processing Unit, positioned before the HSM, identifies the SA using the IP destination address and constructs the plaintext, including the original IP packet, Padding, Pad Length, and Next Header fields, which is then encrypted within the HSM. Post-encryption, the cyphertext obtained is then used to create the ESP packet, together with the other fields that make the final packet like the SPI, IV, and ICV.

For inbound packets, when receiving the packet by the Processing Unit on the Black Side, the SA is identified via the SPI. The ESP header and trailer are then removed, and the cyphertext is sent for decryption within the HSM. After obtaining the original plaintext, we then validate the Sequence Number inside the HSM, since we needed to assure the legitimacy of the data being carried before doing this validation. After validating the Sequence Number, the plaintext leaves the HSM and the original IP packet is obtained by stripping the fields that were added to the packet before encryption, which concludes the Inbound Processing

E. Data Storage and HSM Interfaces

After establishing the division of the ESP processing flow between the two Processing Units in the device, our next step involved examining the specific data requirements for each unit, including the HSM, and define the mechanisms used to share data between multiple parts of the device when necessary. As it was explained before, one of the requirements given was that no value that enters the HSM can also leave directly the HSM, for data leakage purposes.

Starting with the ESP processing done to Outbound Packets, the Processing Unit located on the Red Side will receive a standard IP packet, and, based on the destination IP address carried inside the header, we are able to identify the SA to use. The SA is identified by the SPI value, which will need to be used in both HSM and Processing Unit on the Black side, since the GCM algorithm authenticates the fields that make up the ESP header, and because the value will need to be used in order to construct the header once the cyphertext leaves the HSM.

Since we are not able to pass the SPI value from the Red side to the HSM and then to the Black side directly, as this goes against the set requirements, we decided to store this, as well as other values that need to be used also on the Black, like the IV, inside the HSM. These values will then be sent them together to the Black side with the cyphertext, once the encryption is done. These values will be stored in an Hardware lookup table inside the HSM, along with other values that are only required inside the HSM, like the secret key and the Salt value to form the nonce.

Since the HSM is not able to lookup the destination IP address in order to identify the values of the SA to use, we still need to send a value, in this case, a table index, from the Processing Unit on the Red side in order access the correct values in the table. Instead of using the IP addresses, we decided to use 1 Byte index values to access the correct SA details. The 1 Byte value is obtained in the Processing Unit on the Red side, based on the destination IP address carried inside the packet, and is sent alongside the plaintext to be encrypted, in order to identify the SA to be used. The next Figure 6 depicts the structure of the lookup table stored in the Red Processing Unit.

16 Bytes	1 Byte	1 Byte
IP	IP Version	SA ID
...	...	...

Fig. 6. Lookup Table in Red Processing Unit - Outbound Processing

Once we get the SA ID value based on the destination address, this value is sent into the HSM, along with the plaintext to be encrypted and a 2 Byte variable that indicates the size of the plaintext in Bytes. The following Figure 7 depicts a scheme of the HSM interfaces with the two Processing Units, as well as the lookup table stored inside it, and how the values, both stored and received, are processed inside it.

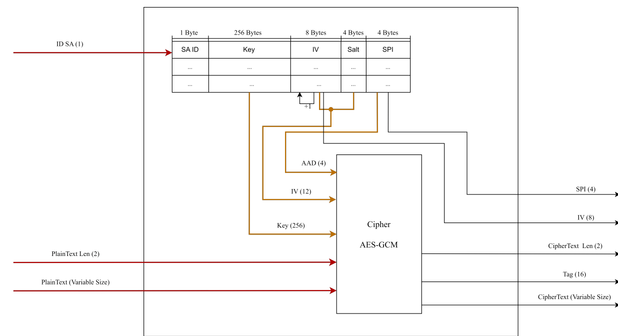

Fig. 7. HSM Interface and Lookup Table - Outbound Processing

With the SA ID inside the HSM we are then able to identify the row with the values needed for the encryption process. Once the encryption process is done, the cyphertext obtained leaves the HSM along with a 2 Byte value indicating the total length of the cyphertext, the resulting authentication Tag, and the SPI and IV values, which will then be used to form the ESP packet. The SPI value will also be used on the Black Processing Unit in order to identify the destination address used to transport this new IP packet. To get this IP address we will have a lookup table similar to the one used on the Red Processing Unit. The structure of this lookup table is represented in the following Figure 8.

The HSM and lookup table architecture defined for Inbound Packets follows the same logic as the one described for Outbound Packets. The information needed to perform the decryption, that is not transmitted inside the ESP packet, is all located inside the HSM and, in order to identify these values based on the SA in use, an index value, acquired based on the SPI value inside the packet, is sent from the processing unit

4 Bytes	16 Bytes	1 Byte
SPI	IP	IP Version
...	...	...

Fig. 8. Lookup Table in Black Processing Unit - Outbound Processing

on the black side into the HSM. The next Figure 9 shows the structure of the lookup table used do get the index value.

4 Bytes	1 Byte
SPI	SA ID
...	...

Fig. 9. Lookup Table in Black Processing Unit - Inbound Processing

With this index value we are then able to get the correct values inside the HSM. The following Figure 10 depicts a scheme of the HSM interfaces with the two Processing Units, as well as the lookup table stored inside it, and how the values stored and received are processed inside it. The index value is sent into the HSM along with the SPI, IV, authentication tag, cyphertext and the indicator of the cyphertext length. Since most of the values needed to preform the decryption are being transported in the ESP packet, we need only to store the secret key and the salt value inside the HSM lookup table. All the needed values are then sent into the cryptographic core, including the nonce which is created inside the HSM by combining the IV with the Salt value.

Once the cyphertext is successfully decrypted we are then able to validate the Sequence Number sent in the packet. To do this we also have stored inside the HSM an array of 64 bits, or 8 Bytes, which represents the sliding window, and another 8 Byte field that indicates the sequence number value correspondent to the last position on the sliding window. If the Sequence Number is correctly verified and the packet is valid, then the obtained plaintext is sent to the Processing Unit on the Red Side, along with a 2 Byte indicator of the plaintext's size. As we don't need to make any alterations to the original packet, once it leaves the HSM, the ESP Inbound processing is finished.

F. Implementation Details

We will now give some details about the choices made when developing the C library that implements the ESP architecture previously described.

The most current version of the library developed implements a total of six functions: three used for processing Outbound Packets, and another three used for processing Inbound ones. Each of these functions is responsible for the ESP processing done in each of the parts of the device, so for both cases, we have a function that does the Pre-HSM processing, a function that simulates all the processing done inside the HSM, and finally, a function that is responsible for the Post-HSM computations.

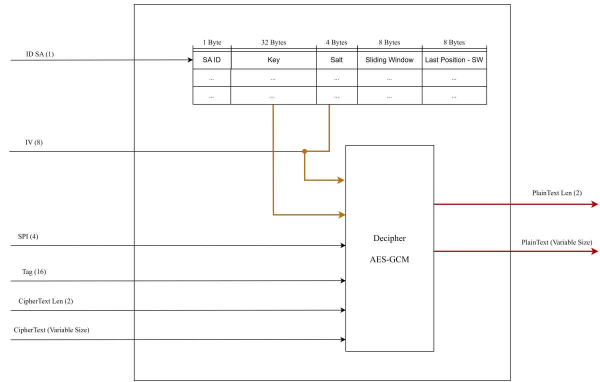

Fig. 10. HSM Interface and Lookup Table - Inbound Processing

To share the required data between the functions we used dynamically allocated structures, that were defined for each of the "interfaces". In order to share these values, for example, from the Pre-HSM function to the HSM one, the function Pre-HSM function returns a pointer to the allocated structure when it has finished, and that same pointer is then sent has an argument when calling the HSM function.

The look-up tables that were defined are also implemented using dynamically allocated structures organized as a single linked list.

V. Results

This section will present a detailed overview of the results obtained with the final implementation done for the presented architecture. We will start by presenting the software set-up used to run most of the tests done, proceeded by an analysis of overhead and throughput values obtained.

In order to run test solely with our implementation we needed to simulate the main code running on the Processing Units, which is responsible for receiving and sending the packets into the network.

To simulate the arrive and sending of packets into the network we used a sample of packets stores into pcap files, and loaded them into the code with functions available on the pcap library. With these functions we were also able to store the resulting packets on other files of this type.

The following Figure 11 visually represents the software setup used. In these Figure both Outbound and Inbound Processing blocks represent one of the device in development and the pcap files are used to simulate the packets traversing the network and arriving and leaving both these devices.

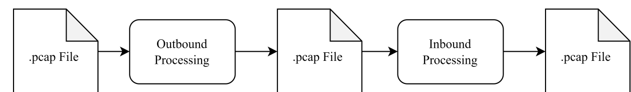

Fig. 11. Test Set-Up

A. ESP Data Overhead

We started by analysing the values of Header Overhead required by our adaptation of the ESP Protocol. In the next

Table I we present the values of needed of Overhead needed for both our adapted ESP protocol and the standard. The values calculated include the minimum and maximum number of Bytes required by the ESP protocol, as well as the percentage of Bytes required by the ESP Protocol, relative to the whole ESP packet.

The number of Bytes added by the ESP processing does not change depending on the size of the original IP packet. The only factor that influences the number of Bytes added by the ESP Protocol (in specific, working with the GCM algorithm) is the alignment of the data with the block size of the encryption algorithm, which will in turn influence the quantity of Padding Bytes added. As the block size of the GCM algorithm is of 16 Bytes, the Padding can go from 0 to 15 Bytes, which is the difference seen between the minimum and maximum Overhead data added. Besides the Padding, our adaptation of the ESP protocol requires an additional 30 Bytes of ESP data in order to send the packet securely. The difference of 4 Bytes between ours and the standard implementation comes from the combination done of the IV and the Sequence Number, as in the standard protocol, the Sequence Number occupies a total of 4 Bytes in the final packet.

From the values presented we can see that the percentage of the data occupied by ESP specific fields in the final ESP packet tends to decrease as the size of the packet increases, with this values occupying from up to 30% of the final packet, to a minimum of 2%. When comparing these values to the ones obtained based on the standard ESP protocol we see that there is always a difference on the percentage between the two versions. When dealing with smaller packets this difference is a little bit more noticeable than when working with larger one, such that when we compare both values obtained based on the 1500 Byte packet, the difference is almost non-noticeable.

B. ESP Processing Overhead

We will now go into the analysis of the Processing Time results obtained with the current software implementation. While the figures shared represent the time taken to process a single packet, these values were obtained by first obtaining 10 separate measurements of for the time it took to process 1000 packets and then calculate their average. The obtained value was then divided by 1000 in order to provide an average of the processing time for a single packet. The formula used is also represented in Equation 1. We decided to use this approach as the values measured for a single packet did not provide the necessary accuracy.

$$\text{Processing Time} = \frac{\sum_{i=1}^{10} T_i}{10 \times 1000} \quad (1)$$

In the following Figure 12 and Figure 13 we have two graphs that show us the Processing Times obtained for each one of the three phases of both the Outbound and Inbound ESP Processing. These results were obtained with the board that is being currently used to develop the Processing Units, which is the Marvell Armada A388 SOM [12].

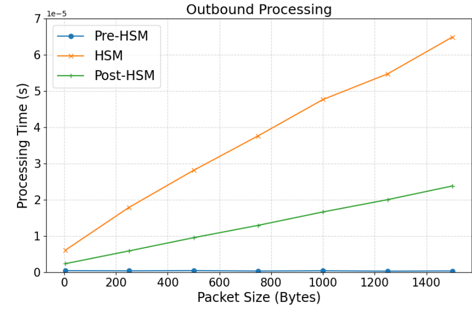

Fig. 12. Processing Time - Outbound Processing

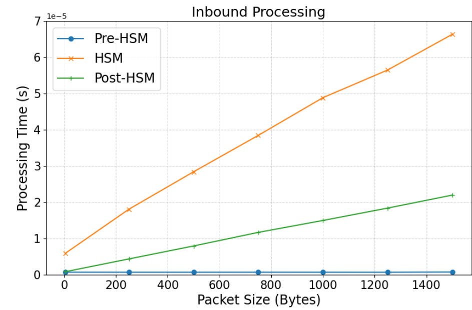

Fig. 13. Processing Time - Inbound Processing

As we can see, the Processing Times obtained are similar for each phase on both the Outbound and Inbound Processing. The processing times obtained for both Pre-HSM functions stay visibly constant for every packet size, since we are only adding or removing fields from a packet that stored in a buffer. As this fields are of a static size and do not vary with the size of the packet, it is expectable that the Processing Time for these functions stays constant even when increasing the packet size.

We can also see that, unlike the Pre-HSM functions, the results obtained from the Post-HSM function show a linear progression that increases with the size of the packet being processed. This happens because once the data leaves the HSM and gets to the Post-HSM processing we need to copy the received cyphertext or plaintext, depending on what processing is being done, into its position in the buffer, which is a task that gets progressively longer as the data to be copied increases. Besides this, the Post-HSM function of the Outbound process is also responsible for adding the ESP header and trailed to the final ESP data, while the Post-HSM function is not required any other major tasks beside copying the plaintext, which explains the small difference of both processing times obtained for the smallest packet.

The graphs also included the processing times obtained for the HSM functions developed, even though these values are not representative of the final processing times that will be obtained from the Hardware version. As we can see, the processing times taken from both HSM functions also show a linear progression that increases with the size of the packet, which is what was also expected from the GCM algorithm.

TABLE I
ESP PROTOCOLS OVERHEAD

	Final ESP Packet Size (Bytes)	Min. ESP Overhead (Bytes)	Max. ESP Overhead (Bytes)	Data Overhead Percentage ESP (%)
Standard IPsec	150	34	49	[22,7 ; 32,6]
	500			[6,8 ; 9,8]
	1000			[3,4 ; 4,9]
	1500			[2,3 ; 3,3]
Adapted IPsec	150	30	45	[20 ; 30]
	500			[6 ; 9]
	1000			[3 ; 4,5]
	1500			[2 ; 3]

C. Throughput

Besides the Processing Time values, we were also able to calculate the throughput values for both Pre-HSM and Post-HSM functions of the Outbound and Inbound Processing done to the packets. These values were also calculated from the the values obtained for the processing of 1000 packets. To calculate the throughput we used the following formula 2.

$$\text{Function Throughput} = \frac{1000 \times 8 \times IPPacketSize}{\sum_{i=1}^{10} T_i} \quad (2)$$

The throughput values were calculated based on the size of the original IP packet being processed and the values are represented in Gigabits per second. As the three parts that form the device are expected to work in a Pipeline format, we have calculated the individual throughput for each function that works in the Processing Units individually, instead of calculating it for the whole process.

The following Figure 14 and Figure 15 depicts the values obtained for the Pre-HSM and Post-HSM functions, for both Outbound and Inbound Processing.

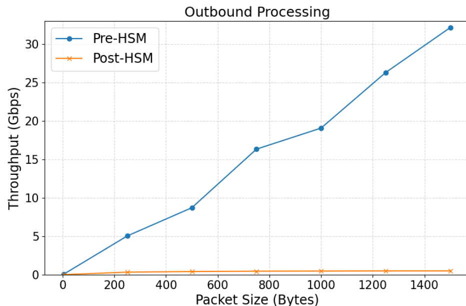

Fig. 14. Throughput Obtained - Outbound Processing

As expected, based on the previous Processing Time values showed, the throughput values obtained for both the Pre-HSM functions tend to increase based on the size of the original IP packet, since the processing time for each individual packet stays constant for every packet size. For the Post-HSM functions, we see that, although the throughput increases

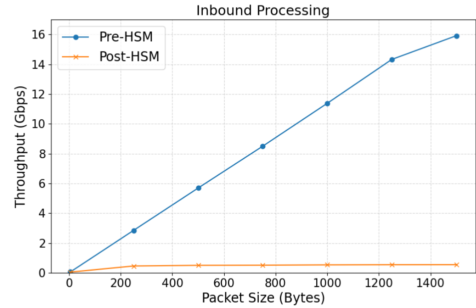

Fig. 15. Throughput Obtained - Inbound Processing

slightly with the original packet size, the values stay relatively constant when compared to the values observed for the Pre-HSM function. This happens because the Processing time for each individual packet tends to increase with the size of the original packet. Because of this, while the Pre-HSM function of the Outbound process can reach up to 30Gbps with a 1500 Byte packet, the Post-HSM function only achieves a throughput of around 0.5Gbps. Similar values can be observed for the Inbound Processing. With this results, we can conclude that both Post-HSM functions constitute the biggest bottleneck present in the current processing done to the packets.

It is important to point out that, since these results were taken based on the Processing Time of each packet, these values correspond only to the throughput possible with optimal conditions. Since we were not able to test the processing with packets come directly from the network, we are also not able to account with latency values that come with those conditions. Because of this, we are assuming that the true throughput results that will be observed once the code is tested in normal conditions will show a decrease in the final values when compared to the results obtained in the tested environment.

VI. CONCLUSION

This dissertation presented a possible adaptation of the ESP protocol, part of the IPsec suite of protocols, to be used in the devices being developed as part of the DISCRETION project, in order to protect packets that are being shared

through the Internet in between military controlled networks. The main goal with this adaptation was to successfully divide the protocol into the three parts of the device in question and define an architecture that kept providing the original security services provided by ESP, while respecting the requirements and limitations set by the device's design.

We began our work by first studying existing security protocols that are commonly employed in the TCP/IP stack, and how they guaranteed different security services to the the traffic traversing the Internet.

We then proceeded to, based on the chosen protocol, to develop an adapted version of this protocol in order to successfully protect the packets that are originated in a safe network environment and are then sent to traverse the Internet. The adaptation done was mostly focused on the division of the steps that are made during the regular ESP processing into the three main parts of the device in development. Once that division was done, we proceeded to define which information and data needed to be shared in between the parts of the device, and also what data needed to be stored for the correct processing, and where. Besides defining the processing to be done in the Processing Units of the device, the majority of the processing to be done inside the HSM has also been defined, in order to simplify the process of implementing the needed steps in the Hardware.

Based on the architecture defined, a C library was developed that implements the three parts of the protocol processing, for both Outbound and Inbound packets. Based on the code developed, we were then able to run the needed tests in order to evaluate our implementation in terms of overhead and throughput values. Based on the analysis done to the protocol header overhead values we were able to conclude that our current adapted implementation does improve slightly on the usage of data for the header construction. Furthermore, with the throughput and overhead processing time values obtained we were also able to identify the bottlenecks of our system, which include the Post-HMS Processing function, for both Outbound and Inbound Processing.

A. Future Work

Finally, in this section we will approach the tasks that take part of the work that still needs to be done to conclude our part in this project.

As our implementation shows to be working correctly when it comes to processing IPsec packets, the next part of our work will consist on taking our HSM software function, which is now working as a place holder for the FPGA to be used, and start implementing its processing steps into Hardware. As the cryptographic core to be used inside the FPGA is already being developed by a colleague, our goal is simply to implement the Look-up tables that are stored, as well as all the processing that needs to be outside the cryptographic core.

At this point we have yet to define how the Hardware implementation will be done, but at the moment we are researching ways in which we can use pre-established structures in VHDL to achieve the best implementation possible.

While going through the hardware implementation, we also plan on revisiting our Sequence Number processing in our ESP architecture. While its processing is currently defined in the architecture, we are currently exploring more efficient implementation methods, and also assessing its necessity, especially if other methods that prevent DoS attacks and guarantee the correct order of the packets will be employed outside the Device in development.

REFERENCES

[1] D. E. S. A. (2023) Discretion website. [Online]. Available: https://discretion-eu.com/

[2] W. Stallings, *Cryptography and Network Security: Principles and Practice*, eigth ed. Pearson.

[3] J. F. Kurose and K. W. Ross, *Computer Networking: A Top-Dow Approach*, eigth ed. Pearson.

[4] S. Kent and K. Seo, "Security architecture for the internet protocol," Internet Requests for Comments, RFC 4301, 12 2005. [Online]. Available: https://www.rfc-editor.org/info/rfc4301

[5] S. Kent, "Ip encapsulating security payload (esp)," Internet Requests for Comments, RFC 4303, 12 2005. [Online]. Available: https://www.rfc-editor.org/info/rfc4303

[6] T. strongSwan Team. (2023) strongswan website. [Online]. Available: https://www.strongswan.org/

[7] L. Projects. (2023) Dpdk project. [Online]. Available: https://www.dpdk.org/

[8] S. community. (2023) Scapy project. [Online]. Available: https://scapy.net/

[9] D. A. McGrew and V. John. (2005) The galois/counter mode of operation (gcm). [Online]. Available: https://csrc.nist.rip/groups/ST/toolkit/BCM/documents/proposedmodes/gcm/gcm-spec.pdf

[10] J. Viega and D. McGrew, "The use of galois/counter mode (gcm) in ipsec encapsulating security payload (esp)," Internet Requests for Comments, RFC Editor, RFC 4106, 06 2005. [Online]. Available: https://www.rfc-editor.org/info/rfc4106

[11] D. Migault, T. Guggemos, and Y. Nir, "Implicit initialization vector (iv) for counter-based ciphers in encapsulating security payload (esp)," Internet Requests for Comments, RFC Editor, RFC 8750, 03 2020. [Online]. Available: https://www.rfc-editor.org/info/rfc8750

[12] SolidRun. (2023) Marvell armada a388 system on module. [Online]. Available: https://www.solid-run.com/embedded-networking/marvell-armada-family/armada-som/carrier-boards