Vector Packet Encapsulation: The Case for a Scalable IPsec Encryption Protocol

Michael Pfeiffer*
Technische Universität Ilmenau

Michael Rossberg*
Technische Universität Ilmenau

Franz Girlich*
Technische Universität Ilmenau

Guenter Schaefer*
Technische Universität Ilmenau

ABSTRACT

The IPsec protocol family, although not always undisputed, has shown to be extremely reliable over the last two decades. However, given the fact that communication networks evolved tremendously since ESP was standardized, this paper proposes changes to the security protocol to accommodate for the needs of modern wide area and data center networks. In particular it addresses optimizations for high-speed software implementations as well as use cases in data center networks. The evaluation shows that rather small yet targeted changes are sufficient to allow for more flexible and scalable implementations.

CCS CONCEPTS

• **Security and privacy** → **Security protocols**; • **Networks** → *Network protocols*; • **Hardware** → Networking hardware.

KEYWORDS

Virtual Private Networks, IPsec, Performance, Multicast, QoS

ACM Reference Format:
Michael Pfeiffer, Michael Rossberg, Franz Girlich, and Guenter Schaefer. 2020. Vector Packet Encapsulation: The Case for a Scalable IPsec Encryption Protocol. In *The 15th International Conference on Availability, Reliability and Security (ARES 2020), August 25–28, 2020, Virtual Event, Ireland*. ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3407023.3407060

1 INTRODUCTION

The IPsec protocol suite [10], in particular the Internet Key Exchange Protocol Version 2 (IKEv2) [6] and the Encapsulating Security Payload (ESP) [9], forms one of the prevalent solutions for providing confidentiality, integrity, and authenticity for communication across untrusted networks. Despite repeated criticism, in the course of two decades IPsec has proven reliable and secure in theory and practice [**? ?**]. Nonetheless, one can isolate three developments that strain its current architecture.

Hardware Parallelism. ESP was initially standardized the same year Fast Ethernet appeared. Today, networking hardware providing 1000 times higher throughput is affordable. On a first glance, this should only affect protocol handling marginally, i.e. requiring a larger space of sequence numbers [7], but modern CPUs scale by increasing parallelism rather than frequency. Thus, software that handles packets at high speeds must be built in a fundamentally different way, which in turn affects protocols.

Modern Processor & NIC Features. Modern processors not only have significantly more cores, but also use vector instructions, e.g. Intel's Advanced Vector Extensions (AVX), to process much more data in parallel. However, the potential speedup requires a stricter memory alignment and emphasizes effective cache management. Furthermore, today's Network Interface Controllers (NICs) offer the possibility to spread packets over multiple small buffers to allow for large jumbo frames yet be cache efficient. Both developments must be supported at protocol level.

Evolving use cases. Due to their growing scale, many data centers shift from switched to routed networks. At the same time, there is a trend towards encrypting intra-data center traffic, often dubbed *zero trust*. This leads to layer 3 Virtual Private Networks (VPNs) being applied in scenarios with a controlled layer 2 network infrastructure. Often, these layer 3 networks are then in turn used to transport encapsulated layer 2 frames, e.g. in Virtual eXtensible Local Area Network (VXLAN) packets [**?**]. These developments lead to the necessity to secure broad- and multicast, jumbo frames as well as Quality of Service (QoS) tagged traffic. Furthermore, servers in data centers can be multihomed, and select uplinks depending on load. All three situations are generally not encountered when running VPNs over the Internet.

To address these challenges, we propose *Vector Packet Encapsulation (VPE)*, a revised security protocol especially suited for the scenarios sketched above. Following good practice in designing security systems, our work is not intended to lead to a revolutionary approach, but a careful evolution of ESP, much in the spirit of IKEv2. Nonetheless, it also provides a unique chance to remove some historical artifacts, e.g. cipher modes that do not provide Authenticated Encryption with Associated Data (AEAD), and to reconsider certain parts, e.g. the transport mode.

In detail, our contributions are the following:

(1) A detailed analysis of the shortcomings of ESP with regard to the points laid out above,

*firstname.lastname@tu-ilmenau.de

ARES 2020, August 25–28, 2020, Virtual Event, Ireland

ACM ISBN 978-1-4503-8833-7/20/08…$15.00
https://doi.org/10.1145/3407023.3407060

(2) a proposal for a modification of the ESP packet layout that allows faster software processing and facilitates multicast as well as QoS, and
(3) a comparison of resulting performance figures of both ESP and our packet header using a modern encryption data plane developed towards 100 Gbit/s networks.

The remainder of this paper is organized as follows: Section 2 proposes requirements for a modern packet security format. Section 3 discusses the deficiencies of current ESP in detail. Section 4 explains how VPE resolves those issues. Section 5 discusses the protocol and provides a quantitative evaluation based on a prototypical implementation. Afterwards, Section 6 discusses related research. Section 7 concludes the paper and points out directions for future research.

2 OBJECTIVES

The IPsec protocol suite defines two data plane security protocols, the aforementioned ESP and Authentication Header (AH) [8]. The latter provides authenticity, integrity, and replay protection, but no confidentiality. It is less commonly used and sometimes outright discouraged [**?**]. Although some of the findings presented in this paper might be transferable to AH, we refrain from a further discussion for reasons of space.

Both ESP and AH can process packets in tunnel and transport mode. The former encapsulates entire IP packets, whereas the latter adds a security header after the original IP header. Although tunnel mode implies an overhead due to the duplicated headers, it provides traffic flow confidentiality and is more flexible than transport mode by supporting Security Associations (SAs) that terminate in a gateway as well as Network Address Translation Traversal (NAT-T). Based on these considerations and in agreement with [**?**], we will focus on tunnel mode in the remainder of this publication.

Combining these considerations with the points discussed in Section 1, a modern packet security header should be based on ESP in tunnel mode, but

- enable parallel processing,
- allow implementations to utilize modern CPU and NIC technologies such as vector instructions and split buffers,
- back evolving use cases by supporting multicast and QoS, and should
- be built with modern AEAD ciphers in mind.

3 SHORTCOMINGS OF ESP

In the following, we will discuss the areas where ESP falls short of these objectives.

3.1 Hardware Parallelism

Sustaining data rates of 100 Gbit/s relies heavily on the ability to parallelize packet processing. A rough estimation is sufficient to illustrate this fact: A single-core 4 GHz CPU processing minimum-sized Ethernet frames at wire speed from a 100 Gbit/s link could spend less than 27 clock cycles per packet, hardly enough for parsing a simple IP header. Therefore, modern NICs expose multiple independent receive and send queues for each network port. Each CPU core can thereby process network packets by using a different pair of queues, waiving the need to synchronize their accesses. *Flow steering* or Receive-Side Scaling (RSS) is employed to direct packets of a single flow, e.g. with the same addresses and port numbers, to the same queue. This avoids packet reordering, which disturbs higher-layer congestion control, e.g. in Transmission Control Protocol (TCP).

In the case of ingress ESP traffic, TCP and UDP headers are encrypted. Therefore, the NICs can only steer entire SAs to specific queues, identified by their addresses and Security Parameter Indices (SPIs). This is not an issue if there are many SAs with light traffic, e.g. a gateway serving hundreds of road warrior clients. But to support high-traffic SAs, e.g. those that couple two data centers, operators are forced to choose the lesser of two evils: They can either disable flow steering or create multiple SAs between the two gateways.

Disabling flow steering is unsuitable, even if one would accept the resulting packet reordering. This is due to the fact that although multiple cores could process packets simultaneously, their replay windows would need to be synchronized. Locking the respective data structures completely causes prohibitive performance issues, but even atomic instructions provided by the hardware can perform the required operations only at rates that are by magnitudes lower than the required packet rates.

Creating multiple SAs pushes data plane properties such as the number of processor cores into the operational domain. This entails a higher risk of misconfiguration, a larger overhead due to the multiple IKE exchanges, and a more complicated monitoring, e.g. for dead peer detection. Furthermore, there are issues with the scalability as the required SAs would actually form a Cartesian product of CPU cores, senders, and QoS classes, as discussed in more detail below.

While hardware accelerators, i.e. Field-Programmable Gate Arrays (FPGAs) or Application-Specific Integrated Circuits (ASICs), may provide higher throughput, they cannot solve the problem entirely. First, they also rely on increased parallelism to keep up with growing network throughput [19], as chip frequencies do not advance as fast as data rates. Second, they are not generally applicable, e.g. in Network Function Virtualization (NFV) deployments, in setups that require independent review of cryptographic implementations, or simply due to development or deployment costs.

3.2 Modern Processor & NIC Features

Achieving high throughput and low latency in software packet processing depends on CPU cache efficiency and proper data alignment. The latter heavily depends on the actual CPU architecture and means that memory addresses should be a multiple of the byte-length of the CPU registers. Current ESP requires the payload, i.e. the encrypted part of the packet, to be aligned to 4 or 8 bytes for IPv4 and IPv6 respectively. In contrast, the cryptographic extensions provided by modern architectures operate on vector registers of 16 (Intel SSE, ARM Neon) or 32 bytes (Intel AVX). However, by adjusting the *headroom*, i.e. the offset of a received packet within its memory buffer, it is possible to align the start of the encrypted data to a 16- or 32-byte boundary. Still, ESP's trailer, which contains the ICV (see Fig. 1), may be unaligned regarding the needs of a specific architecture. Furthermore, depending on packet size, prepending

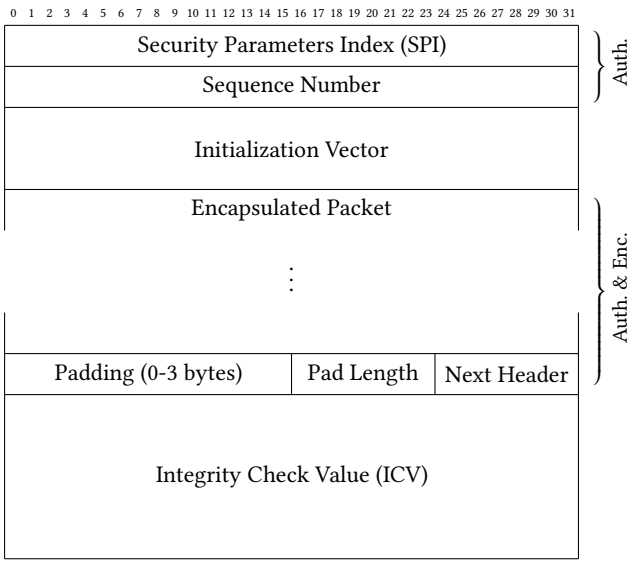

Figure 1: IP payload of an ESP tunnel mode packet with AES-GCM as specified by the corresponding RFCs [9, 18].

an ESP header and appending a trailer at the same time, may cause the use of two additional cache lines (instead of one), which may also result in a performance penalty.

Even worse, many network cards place large packets, i.e. jumbo frames, into multiple chained packet buffers. This can generally not be avoided by making buffers large enough to even hold jumbo frames, as then cache efficiency would decrease if large and small frames are intermixed. But if chained buffered are used, the protocol trailer may be split between two buffers. Its size depends on the length of the ICV (negotiated during key exchange) and the flexible padding in the encrypted part of the packet. Thus, this cannot be fixed at the NIC level. To avoid this special case, software implementations generally just merge the buffers, which again leads to decreased performance.

Chained packet buffers may also help when performing packet fragmentation during ESP encapsulation, as new headers could simply be placed in extra buffers that are chained to parts of the original one. However, also in this case the ESP trailer adds non-negligible overhead, as parts of it also need to be encrypted – leading to non-contiguous memory during encryption. These memory operations are especially aching, as fast cryptographic offloads in modern CPUs shift the focus of performance optimization from cryptographic operations to the protocol handling itself.

3.3 Evolving Use Cases

While QoS and multicast support are mentioned in the IPsec standards [10], there are several issues with both functions, and the deployment of IPsec in local networks makes the situation more pressing. Applying *QoS mechanisms* to prioritize some traffic within a single SA, e.g. by copying the QoS flags to the outer IP header, is in its essence a form of intentional traffic reordering. After bursts of higher-prioritized packets have advanced the replay window, lower-prioritized traffic still in flight will be discarded by the receiver. Avoiding this situation at high data rates may require extremely large replay windows, which in turn contradicts with high processing speeds due to cache inefficiency.

In effect, the problem is similar to the issue discussed in Section 3.1. Again, it would be possible to establish multiple SAs between two devices, entailing the same disadvantages. Worse, if both parallelism and QoS were to be supported, the number of required SAs would increase multiplicatively.

In the case of multicast, problems are even more substantial, as large multicast groups require shared keys among their members. While there have been efforts to standardize group key derivation with IKE (e.g. [20]), ESP is currently not a well-suited transport protocol if multiple senders are involved. The most central issues are:

- Replay protection does not work as it requires synchronized sequence numbers if multiple senders are involved,
- extended sequence numbers only transport the lower 32 bit, hence joining group receivers would need to guess the current higher part, and
- there are no mechanisms to avoid the reuse of Initialization Vectors (IVs) built directly into ESP (while there may be in related standards).

Note, that generating an SA per sender to allow for replay protection is not suited in general, as joining senders would cause the installation of a new SA in all devices. This is considered a *1 affects n* scalability issue.

Apart from these points, many data centers have evolved layer 2 networks, which are no longer based on plain trees, but for example fat-tree or Clos topologies. These topologies have the advantage of being more robust in case of link or switch failures and less susceptible to being blocked by few elephant flows. However, packets between fixed sources and destinations may be sent over different paths, and often servers are even multi-homed; balancing traffic over the uplinks. Thus, in these topologies packets may regularly overtake, leading to situations like described for QoS.

3.4 Summary

To summarize, ESP falls short in the areas of parallel processing, support of QoS and multicast. Its alignment and trailer are suboptimal for fast processing, especially in software. Some, but not all of these shortcomings can be concealed at the cost of higher control plane complexity or more expensive hardware. Nevertheless, we argue that certain changes to ESP protocol are the cleaner, more future-proof approach. They are presented in the following section.

4 PROPOSED CHANGES TO PROTOCOL HEADER AND HANDLING

Resulting from the discussion in Sections 2 and 3, we propose VPE, based on ESP in tunnel mode combined with a modern AEAD cipher. For sake of simplicity, we presume AES-GCM is used [18], although it can be easily substituted by others, e.g. ChaCha20/Poly1305 [15]. VPE shall be based on the established concepts of ESP and only deviate as little as it is required to address the shortcomings described previously. In particular, the security requirements outlined by the existing ESP protocol [10] must not be jeopardized. We further

assume a suitable key exchange that can signal VPE SAs is present, e.g. IKEv2 with a suitably extended security protocol list [6].

The following subsections will discuss changes in processing sequence numbers, deriving IVs, and further reductions in complexity. Section 4.4 concludes with the resulting packet layout.

4.1 Replay Windows and Sequence Numbers

The issues outlined in Section 3 regarding parallel processing, QoS, and multicast replay protection can be remedied by the same technique: Turning the bijective mapping between SA and replay window into a 1:M relationship. That is, one or more unsynchronized senders may assign sequence numbers to packets from multiple, independently running counters. An identification of the counter the sequence number refers to is transmitted with each packet. Thus, the sender does not only use the SPI to lookup the replay window data structure, but instead a triple of (`SPI`, `Sender ID`, `Window ID`). Their use differs slightly, depending on whether the SA is unicast or multicast.

4.1.1 Unicast. In unicast SAs, the `SPI` alone suffices to identify the device that generated a sequence number. The `Sender ID` is not used. However, by assigning a unique `Window ID` to each processing unit, e.g. the CPU core number, parallel processing can easily be supported on the sender side. Note that as long as a receiving processing unit can maintain multiple replay windows, this does neither require the sender and receiver to have the same number of processing units, nor do they need to coordinate tightly. Similarly, QoS or the combination of QoS and parallelism can be supported. In this case a mapping between `Window ID` and QoS semantics must be present, e.g. by configuration. But again, there is no direct need to coordinate this configuration with the receiver. Figure 2 provides an overview of the resulting cardinality relationships.

4.1.2 Multicast. In multicast setups that use a shared group key the SPI identifies a cryptographic context. Separate sequence counters and replay windows are instantiated for each pair of `Sender ID` and `Window ID`. Parallelism and QoS can hence be supported analogously. The addition of the `Sender ID` allows replay protection in presence of multiple uncoordinated senders. To do so, the `Sender ID` must uniquely identify a device sending with the SPI. As secure multicast groups should not become too large for reasons of scalability, we suggest to reserve 16 bit, which is also the maximum value mentioned for IV partitioning in [11].

In practice, there are multiple ways to achieve the uniqueness requirement, e.g. a central key server may assign them to devices dynamically. It is also possible to distribute the `Sender ID` utilizing certificates that are used during IKE. For a prototypic implementation, we used simply the serial number of the certificates. However, this may require a dedicated Public Key Infrastructure (PKI) for each VPN due to the small number space.

4.1.3 Receiver Side. Regarding parallel processing, one problem remains at receiver side: All ingress encrypted packets with the same combination of `Sender ID` and `Window ID` must be steered to the same processing unit by the NIC to avoid costly rescheduling or replay window synchronization (see Section 3.1). In particular this ensures:

	Identification of…				
	Key		**Device**		**Queue & QoS class**
Unicast	SPI	←1:1→	Sender ID	←1:n→	Window ID
Multicast	SPI	←1:n→	Sender ID	←1:n→	Window ID

Figure 2: Cardinality relationships of SPIs, Sender and Window IDs in VPE.

- All packets that use the same replay window are handled on the same receiver CPU, thus replay windows may be tracked using local memory.
- As encrypting gateways usually utilize RSS to spread ingress unencrypted traffic over CPUs, packets of the same flow are also handled by one CPU in the receiver. Thus, packet reordering may be ruled out.

Technically, traffic steering could be achieved by using NICs that implement VPE or those that offer so-called *raw matches* on arbitrary byte positions. However, we may also change the packet structure a little to achieve the same result for existing NICs: By placing `Sender ID` and `Window ID` at the same place in the packet header as the SPI in the original ESP, we can simply utilize RSS (or deterministic flow steering rules) on the SPI field. Note that there is no need to steer VPE's SPIs to specific cores, i.e. there is no functional degradation.

From a security point of view this traffic steering may or may not introduce a certain trust in the NIC depending on the implementation. If traffic steering was broken, replayed packets could be steered to different CPUs, which would in turn not recognize the packets being replayed. However, this attack vector may be foreclosed, if receiving threads simply check that they are responsible for a certain combination of `Sender ID` and `Window ID`.

4.1.4 Transmission Format. Finally, [9] specifies both standard and Extended Sequence Numbers (ESNs), with lengths of 32 and 64 bit, respectively. ESNs are required at today's network speeds to avoid immoderate rekeying and therefore already standardized to be used by default [9]. However, only the lower 32 bit are actually transmitted in each packet. The higher-order bits are kept in the state of sender and receiver. This not only requires reconstructing higher-order values for each packet at the receiver, but is also not feasible for group-key multicast. Members joining a group late may simply be unaware of the higher bits currently in use, and it is not always possible for a central key server to track them either. We hence suggest to transmit the entire 64 bit sequence number in every packet.

4.2 Initialization Vectors

The security of many cipher modes, including Galois/Counter Mode (GCM), depends on the uniqueness of the IVs. The current specification for IPsec with AES-GCM [18] requires a number unique to each packet, e.g. a counter, combined with a *salt* that is unique to each SA and originates from the key exchange.

We propose deriving the IV from the replay sequence number instead, removing the need to maintain a separate counter. This

has been suggested previously for IPsec [14] and a comparable technique is used for MACsec [4, p. 58]. Of course, the `Window ID` and `Sender ID` must also be included into the IV, as their very purpose was to allow sequence numbers without synchronization.

The inclusion of the `Sender ID` would not be necessary in unicast connections, as different senders will negotiate different keys. However, including it unconditionally removes a source for potential implementation errors. The salt provides no security value, as IVs must neither be secret nor chosen at random and can be dropped.

4.3 Reducing Complexity & Providing Alignment

The measures discussed in the previous section already reduce the complexity of header processing significantly. For example, it is possible to read the IV directly from the packet, there is no need to reconstruct it. Furthermore, there is no need for Additional Authenticated Data (AAD) as all fields of the header are either used to lookup key material, are part of the IV or the ICV. Hence, attackers may not modify them without causing the authentication to fail.

To further reduce complexity in packet handling and to address the issues discussed in Section 3.2, we suggest removing ESP's trailer altogether. First, the `Next Header` field can be omitted. It does not carry any useful information in tunnel mode, as the encapsulated data will always be an IP datagram. This is even true in the case of IPv6 packets tunneled over IPv4 networks, or vice versa, as in this case the protocol version can be safely inferred from the packet's first nibble.

According to its specification [9], the explicit padding fulfills two functions: Inflating the plaintext to the cipher's block size, and aligning the trailing ICV to a 4-byte boundary. The former is not necessary for modern AEAD modes of operation, and the second does not allow significant acceleration anymore, as modern hardware would require alignment to a significantly higher boundary (see Section 3.2). IPsec requires the padding to be checked at the receiver side, but this is actually just a precaution and not a real security mechanism. Hence, the padding and the padding length field can be removed.

Note that the transmission of an explicit padding length is not required in tunnel mode anyway, as both IPv4 and IPv6 headers contain a length field that allows discerning the actual payload from padding bytes. Interestingly, padding to provide Traffic Flow Confidentiality (TFC) is specified exactly this way in ESP [9], and therefore remains possible with the proposed format modification.

As the padding and `Next Header` fields have been removed, only the ICV would remain in the packets trailer. However, there is no technical reason for a trailing ICV. In contrast, as described by Section 3.2, moving the ICV to the header may prevent unaligned ICV fields (in terms of current and future vector instructions) as well as issues with multi-segment buffers and caching.

The earlier is due to the way NICs place received packets in memory: They do not allocate buffers of the same size as the received packets dynamically, but rather place packets into preallocated buffers at a configurable offset. By placing the ICV in the packet header, software may set this offset to a CPU-specific value that aligns the ICV on the required boundary.

In case a single preallocated buffer is too small, NICs may spread packets over multiple segment buffers. Placing the ICV in the header ensures that it is not placed on a segment boundary, avoiding checks and potential reassembly costs.

Furthermore, appending and prepending a packet at the same time complicates IP packet fragmentation. Without going into details (for reasons of space), this is due to the fact, that the operations may not be performed in-place, as appending a trailer would overwrite the data of the next fragments.

4.4 Resulting Packet Layout

The packet header resulting from the preceding discussion is shown in Fig. 3. `Sender ID`, `Window ID`, and `Sequence Number` are placed at its beginning. The position of the IDs equals that of the SPI on the ESP header (see Fig. 1), allowing the reuse of existing hardware RSS (see Section 4.1). Furthermore, a 12-byte read operation at the beginning of the packet efficiently returns a 96-bit IV (see Section 4.2). Unlike ESP's sequence number, the three fields are not used as AAD. Still, their manipulation would cause an ICV mismatch and a subsequent rejection of the packet.

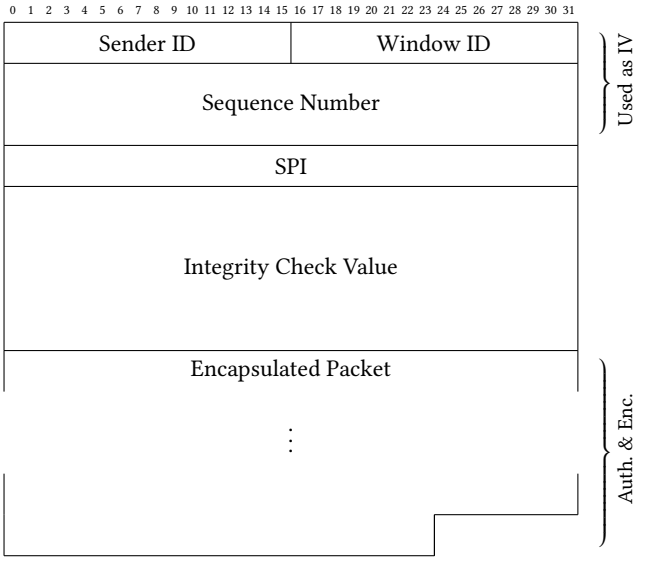

Figure 3: The proposed packet header.

The `SPI` is located in the next four header bytes. It does not need to be authenticated, as a manipulation would either route the packet to a cryptographic context with different key material or an invalid one. Both result in the rejection of the packet. The IDs, SPI and Sequence Number are transmitted in little-endian format, saving byte-order conversions on the x86 and AArch64 architectures dominating current hardware. The following ICV is aligned on a 16-byte boundary respective to the beginning of the header. The remainder of the datagram contains the ciphertext of the encapsulated packet, encrypted and authenticated by the AEAD algorithm.

5 EVALUATION

The security protocol described in Section 4 must persist against the requirements laid out in Section 2.

5.1 Security Discussion

The security guarantees are straightforward to discuss, as on a high level, the encapsulation and protection remains the same, i.e. in both classic ESP and VPE the tunneled packet is completely encrypted and authenticated. The particularly complex key exchange has not been touched. The choice of algorithms and cipher modes follows state-of-the-art recommendations [16, 21]. Although current AEAD modes do not require block-size padding, VPE is future-proof as it is still possible to insert arbitrary padding (see Section 4.3), in case a future cipher mode reintroduces this requirement. For the replay protection, the NICs must either be trusted to direct traffic with the same sender and window IDs to the same core, or an additional software check verifies the steering (see Section 4.1).

In certain cases, we argue that VPE improves the security:

- Replay protection for multicast becomes feasible in multi-sender setups,
- inadvertent reuse of IVs becomes less likely due to their explicit definition, and
- unauthenticated encryption as well as plain authentication are not specified at all.

Furthermore, software implementations are less complex and the need for AAD is removed.

Traffic Flow Confidentiality (TFC) requires a closer examination: With regard to packet format, identical guarantees can be made, because no fields of encapsulated packets are exposed. But, similar to IPsec, an observer could reason about sizes and timings of packets. However, as with our approach there are multiple parallel flows that can be discerned by their window IDs, more information is leaked. The extent can be reduced by changing the RSS hash function periodically. If required, the same countermeasures as with traditional ESP can be applied on top: Packet clocking combined with fill packets (on a per processing unit basis) conceals all traffic flow information.

QoS, multicast, and high-speed processing are supported by our design. However, especially the impact of the latter requires a thorough evaluation, which will be performed in the remainder of this section.

5.2 Implementation & Experimental Setup

We implemented a highly parallelized, performance-optimized software IPsec processor that supports both ESP and VPE. It is primarily programmed in C++, but intrinsics and inline assembler have been applied where necessary. Our implementation makes heavy use of templating and function inlining to provide a modular architecture without sacrificing performance to function calls or branching. For the same reason, care was taken to perform no memory allocations at runtime.

To access networking hardware, the Data Plane Development Kit (DPDK) [**?**] is used, which allows our software to run in userspace, bypassing the operating system's kernel entirely. DPDK does not use interrupts, but continuously polls for newly arrived packets, avoiding expensive context switches. This approach is especially suited for setups where network traffic is a relevant performance bottleneck, e.g. VPN gateways, cloud platforms. Please note that this is simply an implementation decision and orthogonal to the security protocol, i.e. VPE could be implemented in a classical operating system kernel using interrupts just as well.

The actual cryptographic operations, i.e. AES encryption and decryption, are delegated to the native instruction set found on current CPUs. We resorted to the *Intel Multi-Buffer Crypto for IPsec Library* [**?**] which provides a fast, vectorized implementation of AES-GCM using AVX2 instructions.

The threading model of our VPE implementation is extremely simple: Each thread is bound to a specific NIC queue pair, receives a *burst* of up to 64 packets from its receive queue, performs the entire processing for these packets, and pushes them into its transmit queue. This *push-through* model has proven for other packet processing applications as well, e.g. [1]. There is a configurable number of those workers. By default, one half of them is used for encryption and the other performs decryption.

For ESP, we implemented two different threading models. One follows the same principle as the one for VPE, but therefore limits the processing of a single SA to a single worker. The other one tries to lift these limitations but is significantly more complex. As shown in Fig. 4, it passes packets between threads via lockless ring buffers. During encryption, the allocation of sequence numbers is performed by a single thread, avoiding the need to synchronize. On the receiving side, RSS is performed on the sequence number field to ensure packets of a single SA are spread to all workers.

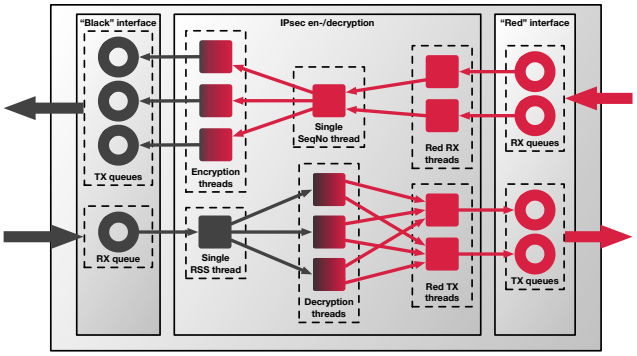

Figure 4: Threading Model for Parallel IPsec Processing

The benchmarks were run on four servers, each equipped with two 20-core Intel Broadwell CPUs (Xeon E5-2698 v4), 256 GiB RAM and 100 GbE NICs (Mellanox ConnectX-5). The worker threads were pinned to specific physical cores, i.e. without hyperthreading, while respecting the Non-Uniform Memory Access (NUMA) topology. The four servers were interconnected directly in a chain:

- The first server ran a traffic generator creating packets of configurable sizes in multiple distinct User Datagram Protocol (UDP) flows at high rates and record the received data rates.
- The second and third server were used as VPN gateways, coupled via a single SA.

- The fourth server executed a traffic reflector by simply swapping the packets' Ethernet and IP addresses and sending them back.

Both generator and reflector where implemented by us, again using DPDK. This setup allows testing higher data rates without additional hardware, as each packet is passed through the gateways twice.

5.3 Non-Parallel Throughput

First of all, we performed an experiment to compare the performance of our prototype against another state-of-the-art implementation. We chose the *IPsec Security Gateway application* [**?**] (*ipsec-secgw*) provided with DPDK, which also uses native AES instructions and is highly optimized. Both implementations were configured similarly with one encryption and one decryption thread per security gateway. We performed measurements with both 64-byte packets (Ethernet's minimum frame size) to achieve high packet rates and 1420-byte packets to obtain high data rates. The latter allow for high data rates while leaving room for cryptographic headers, i.e. no packet fragmentation is introduced.

We expected our ESP implementation to be at least as fast as the state-of-the-art DPDK implementation. Furthermore, we expected VPE to be slightly faster than ESP due to the optimized packet layout. However, VPE's major performance improvement is due to its scalability, which is not examined in this experiment, therefore the improvements were not anticipated to be major. Finally, packet processors do not retain state in the sense that, e.g., simulations do, thus we expected multiple runs to yield highly stable results with hardly any deviation.

Figure 5 shows the mean of 60 one-second measurements for each configuration. As anticipated, the confidence intervals were too small for a sensible display. With small packets, *ipsec-secgw* was able to sustain 1.91 Gbit/s. Our implementation surpassed this, achieving 2.29 Gbit/s (+19.9 %) for ESP and 2.32 Gbit/s (+21.5 %) for VPE. Large packets were processed significantly faster, with 18.09 Gbit/s by *ipsec-secgw* and 19.28 Gbit/s (+6.6 %) for ESP and 19.50 Gbit/s (+7.8 %) for VPE by our prototype.

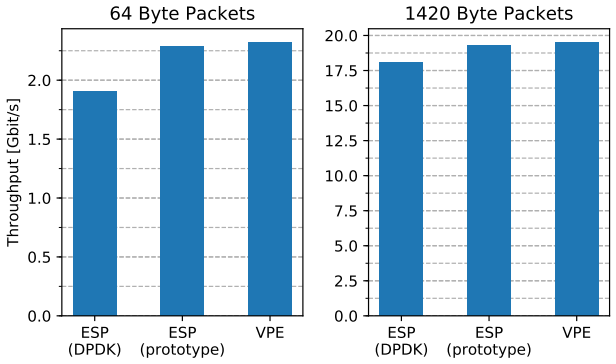

Figure 5: Single-Threaded Throughput

Note that numbers show the throughput received at the traffic generator after the round-trip, i.e. they represent *goodput* including neither the cryptographic nor the tunnel mode's outer IP headers. The data rates of the encrypted traffic, i.e. between both gateways, are therefore higher. Due to the design of our experiment, the numbers represent the unidirectional traffic processed by a single thread, i.e. the entire VPN gateway processes twice the presented rates. The PCI interconnect must handle four times the throughput, as each packet is transported from the NIC to the CPU and back again.

This experiment allows to draw several conclusions:

- Our prototype is faster than a comparable highly-optimized implementation. This demonstrates that the performance issues we experienced with ESP are not the result of an insufficient implementation.
- The large difference between the data rates of small and large packets backs our claim that with modern CPUs, packet handling is much costlier than the cryptographic operations themselves (see Section 3.2).
- Without parallel processing, the performance benefits of VPE are already noticeable, but relatively small. We will investigate these optimizations more closely in Section 5.5. Furthermore, please note that the experiment was not designed to trigger the edge case described in Section 3.2 which are especially costly for ESP, i.e. segmented jumbo frames and fragmentation.

However, VPE's primary design goal regarding throughput was to allow scalability, which will be evaluated in the next experiment.

5.4 Scalability

In our second experiment, we compared the scalability of VPE with the scalability of our multi-threaded ESP implementation. To the best of our knowledge, there is no other publicly available ESP implementation for modern hardware that can perform parallel processing of a single SA while providing replay protection (see Section 6). Therefore, we could not compare VPE to an independent IPsec implementation as it was possible in the previous experiment. We started our measurement of VPE with two processing threads, one for encryption and one for decryption, i.e. essentially the same setup as in Section 5.3. For ESP, we had to begin with three threads due to the threading structure depicted in Fig. 4. Both implementations could then be scaled by successively assigning two more threads to the processing until our hardware was exhausted. Again, we measured the throughput of packets sized 64 and 1420 bytes in 60 one-second measurements each.

With VPE, there is no need to synchronize between threads. Given that the traffic generator produces a sufficient amount of flows, and RSS is working properly, we expected the throughput to scale linearly with the number of threads. The case of ESP is more complex. For low thread counts, we expect a lower throughput than the one obtained in Section 5.3 due to the initial costs associated with an application capable of multi-threading. Furthermore, due to the complex threading structure, we cannot expect linear scaling.

The results are depicted in Fig. 6. VPE with small packets met our expectations, starting with the same throughput as obtained in the previous experiment and peaking at 29.8 Gbit/s using 38 cores. The slope of the approximated linear function is less than 1. We presume this is primarily due to the fact that utilizing more cores

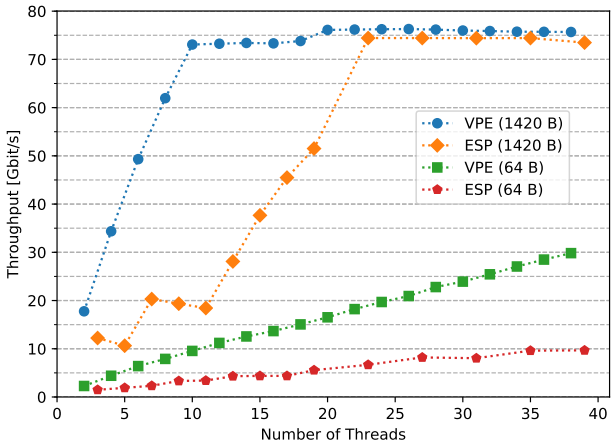

Figure 6: Parallel Throughput of ESP and VPE for Packets sized 64 and 1420 bytes.

leads to a lower maximum frequency as the processor approaches its current, voltage, and thermal budget (*Turbo Boost Technology* in Intel's terms). VPE processing large packets shows a similar linear scaling behavior but appears to reach a threshold of 73 Gbit/s using 10 threads. This appears to be the saturation point of the system, although we can only speculate about the precise bottleneck, which could be located in the NICs, the PCI bus or the QuickPath Interconnect (QPI) between both CPU sockets. The same applies to the slight throughput increase when using more than 18 threads. More experimentation with upcoming hardware, e.g. Ultra Path Interconnect (UPI) and PCIe 4.0, may be able to provide further insights.

ESP processing small packets did also scale with the number of packets, but peaked at 9.7 Gbit/s. Significant throughput is lost while passing packets between the worker threads, e.g. due to a worse cache efficiency. Large packets did reach saturation at about 75 Gbit/s, but, for the same reason, required more than 20 threads to do so. Furthermore, the scaling behavior of ESP is less consistent. For large packets, on some low thread counts a throughput decrease can be observed. This is caused by the threading model that is required to provide replay protection (see Fig. 4), as e.g. adding a distinct sequence numbering thread only pays off after more encryption threads are added later on.

In summary, we can conclude that

- for small packets, i.e. when the throughput is CPU bound, VPE does reach significantly higher throughput,
- for large packets, i.e. when the throughput is I/O bound, VPE is substantially more efficient, and
- VPE's scaling behavior is more predictable, allowing to use a wider range of CPUs efficiently.

5.5 Hardware Architecture Optimizations

Finally, two experiments quantify the impact of the reorganization and alignment of the packet header. To isolate the effects, only the encryption of packets on a single server was measured, using the same implementation that was used in the previous experiments.

First, we measured the time required to encrypt 10 million IP packets, averaged over 25 runs. In agreement with Section 5.3, we expected the time required by VPE to be lower. Furthermore, we anticipated an increase of the required time with larger packets, simply because there is more data to encrypt.

Figure 7 shows the results. VPE shows the expected faster encryption and consistent increase. In contrast, ESP's times are lowest at packet lengths of 1196, 1260, 1324 and 1388 bytes. These numbers are multiples of 64 bytes, the platform's cache line size. We assume that the two partial writes to the packet's last cache line, i.e. encrypting and appending the trailer, particularly impact the caching hierarchy.

In Section 3.2, the ability to adjust the packet buffers' headroom has been discussed, because the AVX operations used to implement the encryption are most efficient when the data is aligned to 32-byte boundaries. Therefore, we would expect a local minimum of the processing time, repeating with every headroom increase of 32 bytes. For ESP, we could not predict whether the insufficient alignment or the appended trailer dominates the results.

The obtained results are depicted in Fig. 8. Contrary to our expectations, VPE exhibits a small local minimum repeating every 16 bytes, i.e. for 2, 18, 34, and 50 bytes. The small absolute improvement could be caused by the fact that on modern platforms the NIC's Direct Memory Access (DMA) operations are sent to the L3 cache, i.e. not subject to the full memory latency and bandwidth limitations. Furthermore, it appears that at least in the examined situation, an alignment to 16 bytes is sufficient. A possible reason could be the 16-byte SSE instructions, operating on the lower half of the 32-byte AVX registers, leading to loads and stores aligned to 16 bytes being also highly optimized. The variations shown by both protocols between 30 and 44 bytes can probably be attributed to changes in the number of required cache line writes. Especially partial cache line writes appear to harm VPE's performance. However, due the multiple superposed influences, clear attribution remains difficult.

Despite those difficulties, we propose the following general conclusions:

- Cache line accesses appear to influence the throughput to a greater extent than register alignment.
- Optimization on this level is highly dependent on the system architecture and the exact nature of the traffic, e.g. UDP encapsulation for Network Address Translation (NAT) shifts the values.
- VPE allows considerably better optimization, as ESP's trailer causes costly cache line accesses to depend on the uncontrollable packet lengths.

6 RELATED WORK

Although to the best of our knowledge, VPE is the first effort to modernize ESP at this scale, there have been other developments in the field of VPN protocols. We will exclude non-encrypting VPNs from the discussion, e.g. Multiprotocol Label Switching (MPLS) and Virtual Private LAN Service (VPLS), as they are built with a different attacker model in mind and provide a functional rather than secure separation.

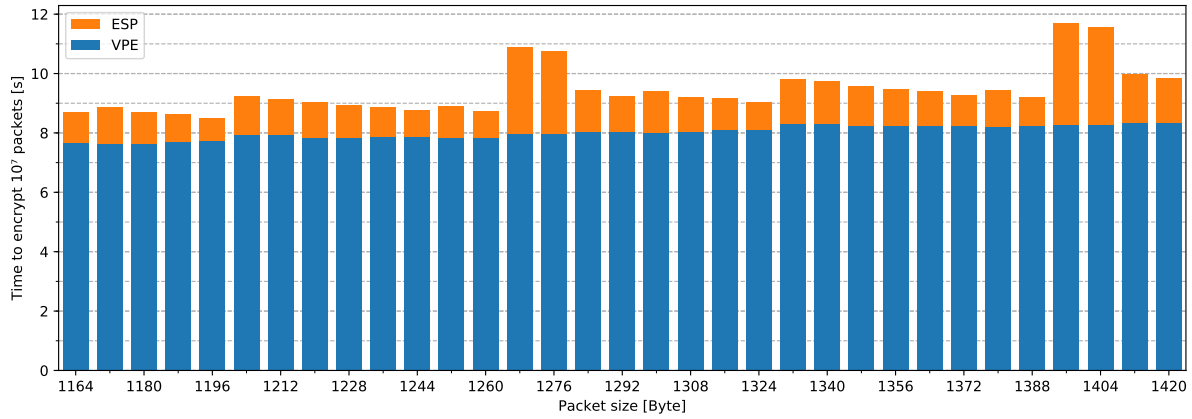

Figure 7: Packet size vs. encryption time average over 25 runs.

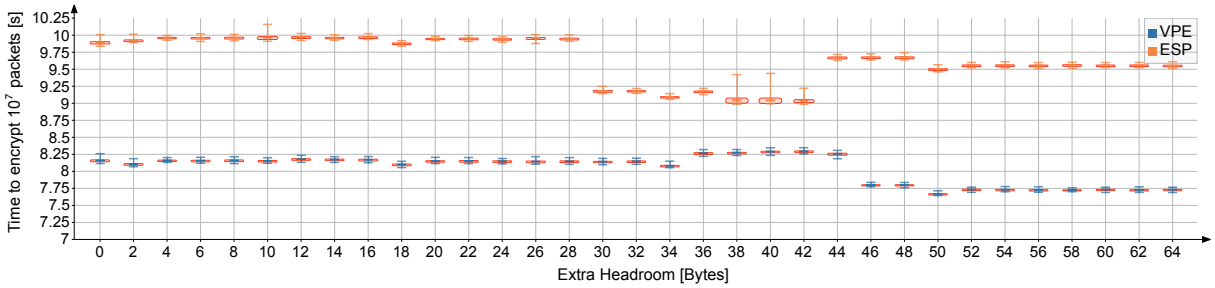

Figure 8: Extra headroom vs. processing time

On the link layer, the most common protocol is MACsec (IEEE 802.1AE) [4]. Although MACsec provides confidentiality, integrity and authenticity, it can only be used between directly connected devices. The Ethernet Security Specification (ESS) lifts this limitation, but link layer protocols still cannot be used in routed networks, e.g. the Internet. Additionally, we propose the growing scale of data center and campus networks will make routed networks more common, making link layer protocols less relevant.

On or above the transport layer, there are multiple protocols, including the ubiquitous Transport Layer Security (TLS) [17]. TLS is used to secure traffic over wide-area networks and within data centers alike but is focused on securing application-to-application communication. Securing host-to-host or network-to-network traffic can only be implemented by deploying TLS for all applications and is difficult and error-prone to enforce. Transport layer VPNs that tunnel Ethernet frames or IP packets, such as Secure Shell (SSH) [22], OpenVPN [**?**] or Secure Socket Tunneling Protocol (SSTP) [13] support neither parallelism nor group communication. Those that use TCP as underlying protocol limit throughput even further, as mechanisms such as congestion control of the tunneled TCP connections interact with those of the underlying connection [3].

Recently, WireGuard [2] received a degree of attention, especially within the Linux kernel community. Both WireGuard and VPE reason that VPNs should be built upon network layer, and that the existing IPsec standard and implementations are rather complex. But WireGuard takes a revolutionary instead of evolutionary approach, defining cryptographic algorithms and timer values unchangeably into the protocol. We deem this approach unsuitable. Simply image the level of confidentiality IPsec could offer today if the DES-CBC encryption, state-of-the-art at the time of ESP's original standardization [5], had been fixed into the standard. Furthermore, the issues concerning parallel processing, QoS and multicast described in this paper are not addressed at all.

Finally, to the best of our knowledge, no existing IPsec implementation provides an alternate, efficient way for parallelizing sequence numbers and replay windows. Most limit single SAs to single cores, e.g. Linux kernel, DPDK, or VPP[1]. Others, such as [12], do not elaborate on this central issue.

7 CONCLUSION & FUTURE WORK

This paper proposed a careful evolution of the ESP security protocol. To this end, objectives that allow a packet format to provide security in today's scenarios and data rates were formulated. The shortcomings of ESP facing these objectives are evident in modern networks, but most of them were not conceivable at the time ESP was invented.

Based on these findings, VPE's packet format has been developed. Its central ideas are to address parallelism, QoS and multicast all

[1]https://github.com/FDio/vpp/commit/f62a8c013c6e22c012b9d7df2ef463a6370cf1ce

at once by allowing multiple replay windows per SA. The implicit derivation of IVs, a proper alignment of the header, the abandonment of the trailer and an explicit transferal of ESNs further simplify implementations. Arguably, each of these adjustments on their own do not justify breaking backwards compatibility with existing implementations. However, in combination they provide a unique chance enabling IPsec to continue providing reliable security.

Further research may answer the following questions in order to open more scenarios for VPE:

- How does VPE behave on CPUs from different generations, vendors or architectures? This could clarify some of the questions remaining from Section 5.5.
- How can VPE be implemented efficiently in hardware? While we believe our results show that hardware accelerators are not necessary to achieve high performance, they may be useful for energy-constrained devices. The parallel architecture should be well suited for hardware implementation, but we have not conducted further examination.
- How can the transport mode be integrated into VPE? This would require to transmit *Next Header* information in the VPE header without impacting performance.

ACKNOWLEDGMENTS

The authors would like to thank secunet Security Networks for funding this research.

REFERENCES

[1] Tom Barbette, Cyril Soldani, and Laurent Mathy. 2015. Fast Userspace Packet Processing. In *ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*. 5–16. https://doi.org/10.1109/ANCS.2015.7110116

[2] Jason A. Donenfeld. 2017. WireGuard: Next Generation Kernel Network Tunnel. In *Network and Distributed System Security Symposium*. https://www.ndss-symposium.org/wp-content/uploads/2017/09/ndss2017_04A-3_Donenfeld_paper.pdf

[3] Osamu Honda, Hiroyuki Ohsaki, Makoto Imase, Mika Ishizuka, and Junichi Murayama. 2005. Understanding TCP over TCP: Effects of TCP Tunneling on End-to-End Throughput and Latency. In *Performance, Quality of Service, and Control of Next-Generation Communication and Sensor Networks III*, Mohammed Atiquzzaman and Sergey I. Balandin (Eds.). https://doi.org/10.1117/12.630496

[4] IEEE. 2018. *Media Access Control (MAC) Security*. Standard 802.1AE-2018. https://doi.org/10.1109/IEEESTD.2018.8585421

[5] Phil Karn, Perry Metzger, and William Allen Simpson. 1995. *The ESP DES-CBC Transform*. RFC 1829. https://doi.org/10.17487/rfc1829

[6] Charlie Kaufman, Paul Hoffman, Yoav Nir, Pasi Eronen, and Tero Kivinen. 2014. *Internet Key Exchange Protocol Version 2 (IKEv2)*. RFC 7296. https://doi.org/10.17487/rfc7296

[7] Stephen Kent. 2005. *Extended Sequence Number (ESN) Addendum to IPsec Domain of Interpretation (DOI) for Internet Security Association and Key Management Protocol (ISAKMP)*. RFC 4304. https://doi.org/10.17487/rfc4304

[8] Stephen Kent. 2005. *IP Authentication Header*. RFC 4302. https://doi.org/10.17487/rfc4302

[9] Stephen Kent. 2005. *IP Encapsulating Security Payload (ESP)*. RFC 4303. https://doi.org/10.17487/rfc4303

[10] Stephen Kent and Karen Seo. 2005. *Security Architecture for the Internet Protocol*. RFC 4301. https://doi.org/10.17487/rfc4301

[11] David A. McGrew and Brian Weis. 2010. *Using Counter Modes with Encapsulating Security Payload (ESP) and Authentication Header (AH) to Protect Group Traffic*. RFC 6054. https://doi.org/10.17487/rfc6054

[12] Jinli Meng, Xinming Chen, Zhen Chen, Chuang Lin, Beipeng Mu, and Lingyun Ruan. 2010. Towards High-Performance IPsec on Cavium OCTEON Platform. In *INTRUST 2010: International Conference on Trusted Systems*. 37–46. https://doi.org/10.1007/978-3-642-25283-9_3

[13] Microsoft. 2018. Secure Socket Tunneling Protocol (SSTP). https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-sstp/c50ed240-56f3-4309-8e0c-1644898f0ea8

[14] Daniel Migault, Tobias Guggemos, and Yoav Nir. 2019. *Implicit IV for Counter-based Ciphers in Encapsulating Security Payload (ESP)*. Internet-Draft draft-ietf-ipsecme-implicit-iv-11. https://tools.ietf.org/html/draft-ietf-ipsecme-implicit-iv-11

[15] Yoav Nir. 2015. *ChaCha20, Poly1305, and Their Use in the Internet Key Exchange Protocol (IKE) and IPsec*. RFC 7634. https://doi.org/10.17487/rfc7634

[16] Kenneth G. Paterson and Arnold K.L. Yau. 2006. Cryptography in Theory and Practice: The Case of Encryption in IPsec. In *Advances in Cryptology - EUROCRYPT 2006*, Serge Vaudenay (Ed.). 12–29. https://doi.org/10.1007/11761679_2

[17] Eric Rescorla. 2018. *The Transport Layer Security (TLS) Protocol Version 1.3*. RFC 8446. https://doi.org/10.17487/rfc8446

[18] John Viega and David A. McGrew. 2005. *The Use of Galois/Counter Mode in IPsec Encapsulating Security Payload*. RFC 4106. https://doi.org/10.17487/rfc4106

[19] Mao-Yin Wang and Cheng-Wen Wu. 2010. A Mesh-Structured Scalable IPsec Processor. 18, 5 (05 2010), 725–731. https://doi.org/10.1109/TVLSI.2009.2016102

[20] Brian Weis, Sheela Rowles, and Thomas Hardjono. 2011. *The Group Domain of Interpretation*. RFC 6407. https://doi.org/10.17487/rfc6407

[21] Paul Wouters, Daniel Migault, John Mattsson, Yoav Nir, and Tero Kivinen. 2017. *Cryptographic Algorithm Implementation Requirements and Usage Guidance for Encapsulating Security Payload (ESP) and Authentication Header (AH)*. RFC 8221. https://doi.org/10.17487/rfc8221

[22] Tatu Ylonen and Chris Lonvick. 2006. *The Secure Shell (SSH) Protocol Architecture*. RFC 4251. https://doi.org/10.17487/rfc4251