On the (In)Security of IPsec in MAC-then-Encrypt Configurations

Jean Paul Degabriele[*]
Information Security Group
Royal Holloway, University of London
Egham, Surrey, TW20 0EX, UK
j.p.degabriele@rhul.ac.uk

Kenneth G. Paterson[†]
Information Security Group
Royal Holloway, University of London
Egham, Surrey, TW20 0EX, UK
kenny.paterson@rhul.ac.uk

ABSTRACT

IPsec allows a huge amount of flexibility in the ways in which its component cryptographic mechanisms can be combined to build a secure communications service. This may be good for supporting different security requirements but is potentially bad for security. We demonstrate the reality of this by describing efficient, plaintext-recovering attacks against all configurations of IPsec in which integrity protection is applied *prior* to encryption – so-called MAC-then-encrypt configurations. We report on the implementation of our attacks against a specific IPsec implementation, and reflect on the implications of our attacks for real-world IPsec deployments as well as for theoretical cryptography.

Categories and Subject Descriptors

C.2.0 [**Computer-Communication Networks**]: General—*Security and protection (e.g., firewalls)*

General Terms

Security

Keywords

IPsec, ESP, AH, MAC-then-encrypt, Traffic Flow Confidentiality, Fragmentation

1. INTRODUCTION

IPsec is a notoriously complex protocol suite, but one of great importance in today's Internet where it is deployed widely to build Virtual Private Networks (VPNs) and secure remote access solutions. IPsec offers security at the IP layer of the TCP/IP protocol stack, meaning that IPsec provides cryptographic protection for IP packets (or their payloads). Part of IPsec's complexity arises from a deliberate attempt by IPsec's designers to provide a flexible and highly configurable approach to providing security services for IP traffic.

The RFCs specifying the major component protocols ESP, AH, IKE [14, 15, 9] and that describing the IPsec architecture [13] offer only limited guidance to end users about how best to configure IPsec to achieve their desired security goals. Moreover, little security analysis of IPsec seems to have been published. In particular, whilst it is by now well-established that using ESP in "encryption-only" configurations is insecure in general [1, 18, 5], there appears to have been no *systematic* security evaluation of the many different ways of combining encryption and integrity protection that are allowed by IPsec:

- ESP may provide its own integrity protection, in which case it is provided by a MAC algorithm that is applied after ESP's encryption – an encrypt-then-MAC construction.
- Alternatively, AH can be used to provide the protection, again using a MAC algorithm, though with the MAC algorithm having a greater scope than in ESP. In this case, packets may be first integrity protected by AH and then encrypted using ESP, or first encrypted by ESP and then integrity protected by AH (where now the extended scope of AH's integrity protection means that more fields of the IP header are protected than would be the case with ESP-provided integrity protection).
- It is even possible to achieve a MAC-then-encrypt construction using two layers of ESP processing.
- Further, the current version of ESP allows combined-mode algorithms to be used, wherein encryption and integrity protection are rolled into a single processing step.
- In all of the above configurations, AH and/or ESP may each be applied in either tunnel mode or transport mode.
- To add a final dimension, both AH and ESP allow sequence number checking to be performed as an op-

[*]This author's research is funded by Vodafone Group Services Limited, a Thomas Holloway Research Studentship, and the Strategic Educational Pathways Scholarship Scheme (Malta), part-financed by the European Union – European Social Fund.
[†]This author's research supported by an EPSRC Leadership Fellowship, EP/H005455/1.

tion, in order to provide protection against replay attacks. This replay protection service should be disabled if manual keying is used (see [14, Section 5] and [15, Section 3.3.3]), is recommended to be disabled for multicast traffic ([14, Section 3.4.3]), and may be problematic when differentiated classes of traffic are protected by a single SA ([13, Section 4.1]). As we shall see, whether the replay protection service is disabled or not has a significant impact on some of our attacks.

It is notable that the previous version of the IPsec architecture [10] was *more* specific about which combinations must or must not be supported in IPsec implementations than is the current version [13]: the former required support for some basic configurations and explicitly outlawed the combination of AH followed by ESP both in transport mode, while the latter makes no prohibitions.

What guidance can be extracted from the literature? Theoretical support for the encrypt-then-MAC options comes from [2, 16], where it is shown that this approach generically provides IND-CCA security if the component encryption algorithm is IND-CPA secure (as is the case, for example, for CBC mode encryption with a random IV) and the component MAC algorithm is strongly unforgeable. We have also seen many on-line tutorials giving example configurations of this type.

Concerning MAC-then-encrypt options, it is noted in [15] that *"an underlying integrity service, such as AH, applied before encryption does not necessarily protect the encryption-only confidentiality against active attackers"*, suggesting that such configurations should be avoided. Here, [15] cites [16] for theoretical support. However, a closer examination of [16] shows that it contains positive security results about the MAC-then-encrypt construction when the encryption scheme is implemented using either a secure stream cipher or CBC mode of a block cipher. These are the primary encryption schemes currently supported by IPsec standards. Moreover, the known examples in [2, 16] showing that MAC-then-encrypt constructions are not generically secure are rather artificial. Thus the results of [16] could be interpreted as providing support for MAC-then-encrypt configurations of IPsec. Further support comes from a widely-cited critique of IPsec by Schneier and Ferguson [6], which states *"When both encryption and authentication are provided, IPsec performs the encryption first, and authenticates the ciphertext. In our opinion this is the wrong order"* and later goes on to say *"The ordering of encryption and authentication in IPsec is dangerous."* In [6] the argument is made that a protocol should authenticate what was meant, not what was said, with SSL as analysed in [23] being given as an example of a protocol adopting the "correct" approach of MAC-then-encrypt. Moreover, a putative attack against encrypt-then-MAC configurations of IPsec is given in [6], lending further support to the MAC-then-encrypt choice for IPsec[1]. A standard textbook on network security [21] discusses several benefits that accrue from using a MAC-then-encrypt configuration of IPsec, including the ability to store MAC values along with plaintexts for later checking. A textbook aimed at implementors of cryptography [7] extensively discusses the merits and demerits of the MAC-then-encrypt approach to building secure channels, and eventually recommends this construction over other choices.

Given the arguments on both sides, and in the absence of firm guidance from the RFCs or other sources, it seems plausible that a network administrator might well be tempted into selecting a MAC-then-encrypt configuration of IPsec.

1.1 Our Contributions

This paper focusses on the security of MAC-then-encrypt configurations of IPsec. For concreteness, we study the common use case of using IPsec to build a simple site-to-site VPN. We describe practical attacks against *all* MAC-then-encrypt configurations of IPsec for this common application, including the most natural configurations as well as more "exotic" ones. We assume that all cryptographic processing is carried out at a pair of security gateways, but our attacks also extend to situations where AH processing is carried out at hosts behind the gateways. Our attacks come in three basic flavours, each with two main variants depending on whether IPsec's optional replay protection is enabled or not. Our attacks are powerful in the sense that they can be used to recover plaintext from arbitrary IPsec-protected packets. But they each have different characteristics in terms of their complexity, their requirements for the attacker's degree of control over the network, and their plaintext requirements. We stress that we have not found any attacks against ESP's encrypt-then-MAC construction.

In developing our attacks against IPsec, we assume that the relevant RFCs have been carefully followed by an implementor. For example, our attacks exploit the recommendation of the ESP RFC [15] to perform full padding checks when decrypting, and two of them rely on support for Traffic Flow Confidentiality (TFC) padding that is mandated in [15]. One of the attacks depends on the details of IPsec's treatment of fragmented packets, while all depend on the manner in which IPsec handles ICMP traffic. Our attacks are developed with the RFC specifications in mind, but our previous experience [5] indicates that IPsec implementations do deviate significantly from the RFCs in ways that can stop attacks from working in practice. To compensate for this, we report on the experimental validation of our attacks against the OpenSolaris implementation of IPsec, showing that two out of three of our attacks "on paper" can be converted into working attacks against a real implementation. We emphasize that our choice of OpenSolaris was driven by the high quality of its code and its close adherence to the IPsec RFCs, and not because it has any particular weaknesses that we wanted to exploit. We believe that our attacks would apply to any comparably careful implementation of IPsec.

1.1.1 *Practice*

This paper makes a significant contribution to network security practice: it shows that certain configurations of IPsec are insecure and should be avoided, namely those involving the application of AH followed by ESP. This confirms and strengthens the limited guidance in [13] and firmly contradicts the recommendations concerning ordering in [6, 7]. In addition, our work has implications for designers of new protocols: our attacks highlight security deficiencies that result from IPsec's modular approach to realising secure channels, the introduction of TFC padding, and the interplay between IP and IPsec (in particular, fragmentation issues).

[1]However this attack requires the receiver to use the wrong key when decrypting, and it is hard to envisage the circumstances under which this could occur in IPsec, except perhaps with re-use of SPIs in a manually-keyed deployment.

The take-away from this paper for practitioners is that ESP with encryption and integrity protection should be used in preference to any other configuration when confidentiality is required.

1.1.2 *Theory*

This paper also has implications for cryptographic theory. It is already known that Krawczyk's positive results [16] concerning the security of the MAC-then-encrypt construction used in SSL/TLS need to be interpreted carefully in the light of attacks against SSL/TLS in [4]. Our results provide a similar demonstration in the context of IPsec. More generally, our work highlights the limitations of current cryptographic theory in answering the apparently simple question of how best to combine encryption and integrity protection to build a secure channel. While theoretical attack models and security proofs such as those in [16] may rule out many classes of attack, they do not always translate into strong security guarantees for real network protocols with all their messy – but essential – features. For example, such models usually assume that all cryptographic processing takes place in an atomic fashion, while our attacks exploit non-atomicity in MAC-then-encrypt configurations of IPsec. Security models do not typically consider padding, error messages, the possibility of ciphertext fragmentation, or interactions with encapsulated protocols, while our attacks exploit such features. It is notable that, while two of our three attacks exploit the fact that AH's MAC does not cover all the plaintext to be encrypted, the third attack based on fragmentation would still work even if it did. This raises an interesting theoretical question about the achievable security of generic compositions of encryption and integrity protection mechanisms in a model where the channel allows fragmentation of ciphertexts.

1.2 Related Work

Previous work analysing IPsec [1, 5, 18] has focussed on encryption-only configurations, showing them to be fatally insecure. Our attacks on MAC-then-encrypt configurations build on techniques developed in [5, 18], but we need to significantly extend them to cater for the integrity protection and replay protection services supplied by AH. We note that [5, 18] also mentioned the possible extension of the encryption-only attacks to a limited class of other configurations where AH and ESP processing are carried out by different machines. However, our work seems to be the first to make a systematic exploration of the security of MAC-then-encrypt configurations of IPsec. An attack against SSL's MAC-than-encrypt construction was reported in [4]. This attack has a similar flavour to our attacks, but differs markedly in its details and realisation. In particular, because of the way SSL behaves when errors arise, the attack of [4] can only slowly recover a fixed plaintext that is repeated across many SSL connections. In contrast, our attacks can recover arbitrary plaintext in an efficient manner. Other related work [6, 16] is already discussed above.

2. BACKGROUND ON IP AND IPSEC

We assume the reader is familiar with the main IPsec concepts (tunnel and transport modes, Security Associations (SAs)) and protocols (AH and ESP). We also assume the reader is familiar with IP, in particular, the format of the IP header and the function of each of its fields. Here, we only consider IPv4. For useful introductions to these topics, we refer the reader to [5, 18, 21]. Below, we relate some of the finer points concerning IPsec that are needed to understand our attacks.

2.1 ESP

We recall that ESP usually makes use of a block cipher algorithm operating in CBC mode: RFC 4385 [17] mandates support for AES-CBC with 128-bit keys and TripleDES-CBC, while no other encryption algorithm is mandated. No combined mode (authenticated encryption) algorithms are required to be supported. In future, we may expect combined mode algorithms and AES-CTR to increase in popularity. A modification of our attacks would work against AES-CTR, if it were not for the fact that [8] specifying AES-CTR requires that it must be used in combination with ESP-provided integrity protection, implicitly in an encrypt-then-MAC construction. We assume throughout the remainder of this paper that CBC mode is in use.

Our attacks depend in a delicate way on how padding and CBC mode encryption (and the reverse operations of decryption and depadding) are performed by IPsec. The variant of CBC mode that is used by ESP is described in full in [15, 5]. The plaintext to be protected is either an inner IP packet (in tunnel mode) or the IP packet payload (in transport mode). This data is treated as a sequence of bytes. It is padded with a particular pattern of bytes and then a Pad Length (PL) byte and a Next Header (NH) byte are appended; this collection of bytes is called the *ESP trailer*. The default padding method specified in [15] is universally used in practice. This method adds bytes so that:

1. The total number of bytes present (including the PL and NH byte) is aligned with a block boundary; and

2. The added pattern of padding bytes is either a null string or t bytes of the form $1, 2, \ldots, t$ for some t with $1 \leq t \leq 255$.

According to [15, Section 2.7] it is permissable to precede this padding with an arbitrary amount of Traffic Flow Confidentiality (TFC) padding of unspecified format. This is intended to aid in preventing traffic analysis by disguising the true length of the inner packet. Some of our attacks exploit support for this padding. The NH byte is present in order that the decrypting IPsec entity can know to which protocol it should pass the bytes that precede the padding bytes. In tunnel mode, this value should be `04` indicating IP; in transport mode, a variety of values will be found here, with local IPsec policies determining which values are acceptable.

After adding the ESP trailer, the data is encrypted using CBC mode. Let us assume that the byte sequence after padding consists of q blocks, each of n bits (where $n = 64$ for triple-DES and $n = 128$ for AES, for example). We denote these blocks by $P_1, P_2, \ldots, P_q$. We use K to denote the key used for the block cipher algorithm and $e_K(\cdot)$ ($d_K(\cdot)$) to denote encryption (decryption) of blocks using key K. An n-bit initialization vector, denoted IV, is selected at random. Then ciphertext blocks are generated according to the usual CBC mode equations:

$$C_0 = IV, \quad C_i = e_K(C_{i-1} \oplus P_i), \quad (1 \leq i \leq q).$$

The encrypted portion of the packet is then defined to be the sequence of $q+1$ blocks $C_0, C_1, \ldots, C_q$. The basic format of an ESP-protected packet is shown in Figure 1.

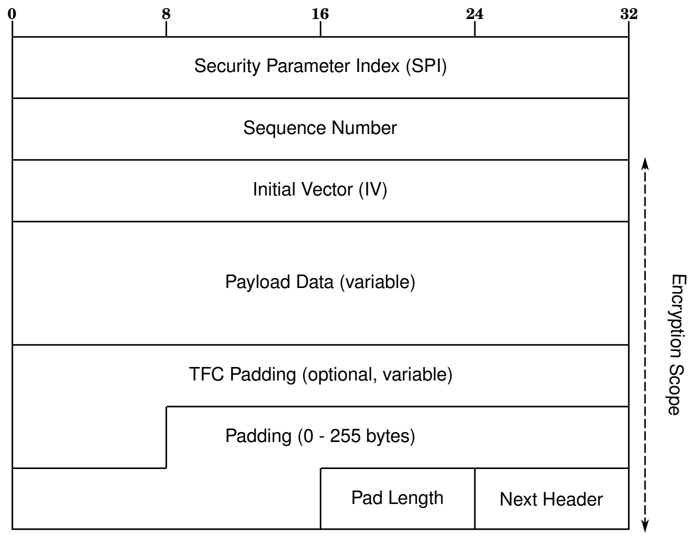

Figure 1: Structure of ESP protected-packet (adapted from RFC 4303 [15] for CBC mode without integrity protection and to show encryption scope).

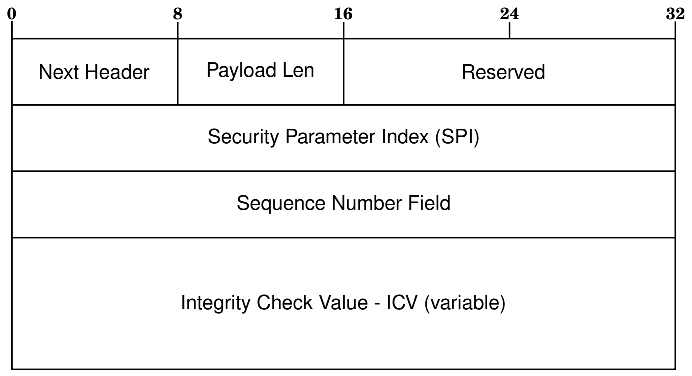
0
8
16
24
32
Next Header
Payload Len
Reserved
Security Parameter Index (SPI)
Sequence Number Field
Integrity Check Value - ICV (variable)

Figure 2: AH format according to RFC 4302 [14].

At the entity performing IPsec decryption (which is also in possession of the key K), the padded plaintext can be recovered using the equations:

$$P_i = C_{i-1} \oplus d_K(C_i), \quad (1 \leq i \leq q).$$

Any padding along with the PL and NH bytes can then be stripped off, revealing the original inner packet/payload. Section 2.4 of both the ESP RFCs [12, 15] states that "*the receiver SHOULD inspect the padding field*", because certain cut-and-paste attacks are prevented if "*the receiver checks the padding values upon decryption*". We assume that an RFC-compliant implementation performs a strict check, ensuring that the padding conforms exactly to what is expected given the value of the PL field, and dropping the inner datagram if the check fails[2].

The receiver then reconstructs the original IP packet, with the exact steps depending on the mode (transport or tunnel) and being described in [13, Section 5.2]. This processing also discards any TFC padding. In tunnel mode, this can be achieved by using the length field in the header of the inner IP packet to determine where to trim the packet. In transport mode, this relies on the upper layer protocol format including a length field which can be used for the same purpose. Some of our attacks depend on support for TFC padding at the receiver side, and we will note where this is the case. The original IP packet is then compared with the used SA's traffic selectors to check that the correct cryptographic processing was applied. This check will (implicitly) ensure that in tunnel mode the NH byte does contain 04, for example, with the packet being dropped if it does not. Likewise, in transport mode, the NH byte will need to be consistent with the upper layer protocols allowed for that SA. Finally, the packet is forwarded, either to the upper layer protocol specified in the NH byte, or for further cryptographic processing (when nested SAs are in use).

Note that when ESP is employed without integrity protection, the sequence number in the ESP header must not be checked by the recipient [15, Section 3.4.3].

2.2 AH

AH, as specified in [14], provides an integrity protection service and, in combination with sequence number checking, a replay protection service for IP packets. The AH format is

[2]Note that weaker forms of padding check such as BSD-style checks [5] still allow us to recover partial blocks of plaintext in our attacks.

shown in Figure 2. AH processing involves inserting the AH fields between the IP header and payload/inner IP packet, and then applying a MAC algorithm to selected fields of the (outer) IP header, the AH fields, and either the payload (in transport mode) or the inner IP packet (in tunnel mode); a typical MAC algorithm for AH is HMAC-SHA1-96. The calculated MAC value (ICV value) is placed in the relevant AH field. Not all of the (outer) header fields are included in the MAC computation because they may change in unpredictable ways as the packet traverses an IP network; the unprotected fields include the header checksum and TTL fields.

At the receiver, the MAC is checked and the packet discarded if the MAC is incorrect. In addition, when replay protection is enabled, the 32-bit sequence number carried by AH is compared to a sliding window of recently received sequence numbers. The packet is again rejected if the sequence number has already been received or if it is deemed to be too old by falling to the left of the current window. A packet having a valid MAC and a sequence number greater than the largest previously accepted will always be accepted, causing the window to be shifted to the right. RFC 4302 [14] also supports the use of 64-bit extended sequence numbers. We assume these are not selected for simplicity of presentation, but our attacks still work if they are.

2.3 IP

A useful overview of IP sufficient to understand our attacks can be found in [5]. In this paper, we will be mostly concerned with the TTL field in the IP header and the IP header fields related to packet fragmentation.

We recall that the TTL field is set to some initial, OS-dependent value when a packet is first generated, and then decreased by 1 at each router that the packet traverses. When the TTL field reaches 0, the packet is discarded, and an ICMP error message (of type 11 and code 0) indicating a "time to live exceeded" event is generated and sent to the host indicated by the original packet's source IP address.

Support for fragmentation is a necessary part of IP implementations arising from the need to cater for a variety of lower layer protocols. The second 32-bit word in the IP header is used to handle fragmentation issues. In particular, the 16-bit Identification (ID) field is used to identify all the fragments coming from a single initial packet, the MF bit indicates that more fragments are expected after this fragment, the DF bit indicates that this packet should not be fragmented, one bit is unused, and the remaining 13 bits are used to carry the fragment offset which is used to order fragments during packet reassembly. A fragment is indicated by a non-zero MF bit or a non-zero offset field. According to [3], to avoid unacceptable delays in reassembling fragments, the reassembly process must eventually time-out, with the wait period being a fixed period that is recommended to lie between 60 and 120 seconds. If this timeout expires, the partially-reassembled packet must be discarded and an ICMP Time Exceeded message (of type 11 and code 1) must be sent to the source host.

The IPsec architecture [13] notes that AH and ESP cannot be applied using transport mode to packets that are fragments; only tunnel mode can be employed in such cases. RFC 4302 [14] requires that *"An IPv4 packet to which AH has been applied may itself be fragmented by routers en route, and such fragments must be reassembled prior to AH processing at a receiver."* This feature is exploited in our attacks based on fragmentation.

2.4 ICMP and IPsec

The IPsec architectural RFC [13] explains in detail how IPsec should handle ICMP messages, distinguishing between error and non-error messages. Our attacks use ICMP messages of both types, and the specific messages used in our attacks are not blocked by IPsec. However, they are only visible to the attacker in encrypted form and so typically need to be detected by their characteristic (though implementation-dependent) lengths, or via timing correlation.

2.5 Bit Flipping in CBC Mode

We recall the following well-known property of CBC mode. Suppose an attacker captures a ciphertext $C_0, C_1, \ldots, C_q$, then flips (inverts) a specific bit j in C_{i-1} and injects the modified ciphertext into the network. Then the attacker induces a bit flip in position j in the plaintext block P_i as seen by the decrypting party. This tends to randomize block P_{i-1}, but if the modification is made in C_0 (equal to IV), then no damage to plaintext blocks will result. This obviously extends to flips applied to multiple bits simultaneously.

In our attacks, we will flip certain bits in the headers of inner datagrams. Any such modifications will require further compensation to be made elsewhere in the header so that the Header Checksum (calculated as the 1's complement of the 1's complement sum of the 16-bit words in the IP header) is still correct – otherwise the inner datagram will be silently dropped. In [5, 18], a number of techniques were developed for "correcting" checksums in an efficient manner. We need to further develop these techniques so that our attacks are efficient for the IPsec configurations considered here.

Considering each 16-bit field in the IP header as an unsigned integer, suppose we wish to subtract the value δ from one of these 16-bit fields. Let S represent the 1's complement sum of all the 16-bit fields over which the checksum is computed, then the IP header checksum is given by $\overline{S}$ (the complement of S). Thus the new value of the IP header checksum should be set to $\overline{(S \boxplus \overline{\delta})}$ where $\boxplus$ denotes 1's complement addition. Then we need to select a 16-bit value `mask` such that:

$$\texttt{mask} \oplus \overline{S} = \overline{(S \boxplus \overline{\delta})}$$

and XOR this value `mask` to the appropriate field in the IV. We can rewrite this equation as:

$$\texttt{mask} = \overline{S} \oplus \overline{(S \boxplus \overline{\delta})} = S \oplus (S \boxplus \overline{\delta}).$$

Hence we can, for a fixed value of δ, compute all possible solutions `mask` to the above equation along with their probabilities of success in correcting the checksum, assuming that S is a uniformly distributed 16-bit value. We then use the list of possible values `mask` in order of decreasing probability when trying to correct the checksum.

An example is in order. Suppose we wish to decrease the TTL field from a known value `0xFF` to the value `0x00` and correct the checksum. Because of the position of the TTL field in the IP header, this implies a 16-bit value $\delta = \texttt{0xFF00}$. Some of the resulting 66 masks having non-zero probability are shown in Table 1 along with their probabilities. In this case, the number of trials required is decreased from the average of 2^{15} that would be needed using the methods of [5,

mask	probability
0000 0001 0000 0001	2^{-2}
0000 0011 0000 0001	2^{-3}
0000 0001 0000 0011	2^{-3}
0000 0111 0000 0001	2^{-4}
0000 0011 0000 0011	2^{-4}
0000 0001 0000 0111	2^{-4}
0000 0001 0000 1111	2^{-5}
⋮	⋮
1111 1111 1111 1111	2^{-16}

Table 1: Table of masks and probabilities for $\delta = 0xFF00$

18] to an average of only 6.75. On the other hand, assuming nothing about the TTL field except that it is uniformly distributed, then a simple calculation using a variant of this approach shows that the expected number of trials needed to set the TTL field to `0x00` and correct the checksum is only 382.

This idea can be combined with the idea from [5] of using the ID field to compensate for the bit flips, rather than the checksum field itself. Because of the location of this field in the second 32-bit word of the IP header, this allows the above improvements to be deployed even for a 64-bit block cipher.

2.6 ESP Trailer Oracles

Our attacks will make use of ESP trailer oracles, a concept introduced in [5] as an extension of the padding oracle concept from [22]. Such an oracle tells the attacker whether or not the trailer fields (including padding, pad length and NH bytes) of an encryption-only ESP-protected packet are correctly formatted. It is shown in [5] how repeated access to such an oracle allows an attacker to decrypt ESP-protected ciphertext blocks in a byte-by-byte fashion, at a cost of at most 2^{16} queries to the oracle to extract the rightmost 2 bytes of the target block and at most 2^8 queries for each remaining byte of the target block. We describe this attack in outline next.

Suppose we have a *carrier packet* that is protected by ESP in tunnel mode, and a target ciphertext block C_i^* (from any packet protected by the same key K). The rightmost 2 bytes of C_i^* are extracted as follows. We splice blocks R, C_i^* onto the end of the carrier packet, and submit this new packet to the oracle. Here R is a randomly selected block. By varying the rightmost 2 bytes of R in a systematic fashion, we can explore all possible values of the rightmost 2 plaintext bytes in the block $R \oplus d_K(C_i^*)$; these are interpreted as the PL and NH bytes of the ESP trailer by the oracle, and it is argued in [5] that, with high probability, only the values `00,04` for these bytes will produce a positive response from the oracle. Once the oracle responds positively, the corresponding original plaintext bytes from $C_{i-1} \oplus d_K(C_i^*)$ can be easily recovered by simple XOR arithmetic. The attack is then extended to plaintext bytes further to the left in the block by trying to construct longer valid trailer byte patterns, starting with `01,01,04`. Further details can be found in [5].

This leaves the question of how to construct an ESP trailer oracle. This problem was solved in [5] for the encryption-only case by constructing a special packet that, providing the packet was not dropped because of a failure of ESP's padding checks, always generated some kind of error response. Usually this takes the form of an ICMP message. In [5], for tunnel mode, encryption only ESP, this was done by using CBC bit flipping and checksum correction to create a packet whose inner packet had an unsupported protocol field. The resulting ICMP message is usually transmitted in encrypted form on the IPsec tunnel, but it was shown in [5] how such messages can be detected based on characteristic lengths or via timing correlation.

In summary, to mount this kind of attack, we need a carrier packet that produces a detectable response whenever ESP's trailer formatting checks pass. In [5], this required modification of IP header fields in the inner packet. This clearly creates a problem when the inner packet is protected by AH, as it is in the situations we are interested in here: now modifications to header fields may be detected by AH processing and the packets dropped, causing the oracle to be lost. An extra level of complication arises if AH's replay protection is enabled: now, each carefully-constructed carrier packet can only be used once, since if it were to be repeated, its inner packet would be deemed to be a replay during AH processing and so dropped, again causing the ESP trailer oracle to be lost. Finally, we also want to consider transport mode configurations of IPsec, and additional ideas are needed to cater for this.

As we explain in the sections that follow, all of these problems can be overcome and appropriate ESP trailer oracles constructed.

3. ATTACKS

Requirements.

We have the following requirements for all of the attacks in this paper:

1. IPsec is used between a pair of security gateways G_A and G_B; these gateways could be stand-alone or could be providing security for pairs of communicating hosts H_A and H_B located behind the gateways as illustrated in Figure 3. This is a common, basic VPN configuration of IPsec.

2. The cryptographic keys used in AH and ESP for IPsec processing at G_A and G_B are fixed during the attack;

3. The attacker can monitor and record traffic flowing between the two gateways;

4. The attacker can inject modified datagrams into the network between G_A and G_B.

In addition, for some of the attacks where the optional AH sequence number checking is enabled, we will require that the attacker is able to control the flow of legitimate packets between G_A and G_B to some extent. In practice, this could be achieved by an attacker who controls a router located between the gateways.

We begin by describing our three basic attack ideas in the context of the IPsec configuration that first applies AH in transport mode and then ESP (encryption only) in tunnel mode to packets flowing from G_A to G_B. This seems to us to be the most natural MAC-then-encrypt configuration,

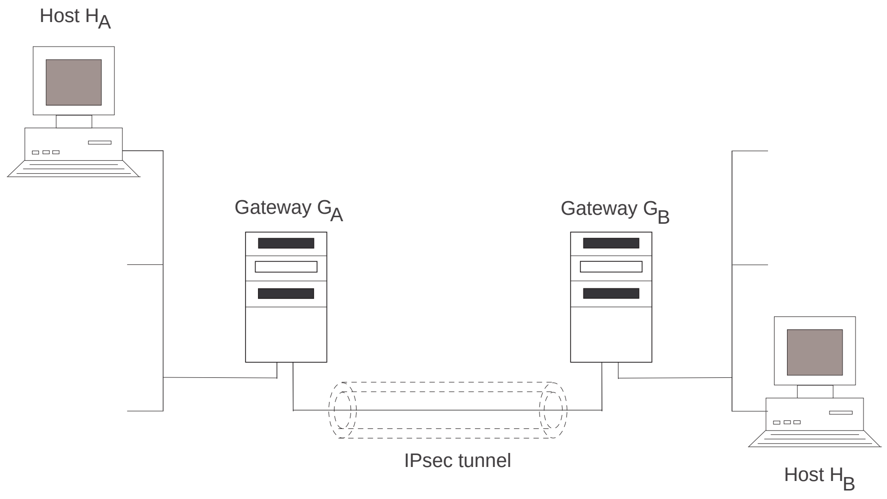

Figure 3: Network set-up.

and it also turns out to be the easiest to attack. We then go on to explain how to extend the attacks to other MAC-then-encrypt configurations. In each case, we explain how to recover the plaintext block corresponding to a single target ciphertext block C_i^*. Of course, all of the attacks extend to multiple blocks in the obvious way.

3.1 Attacks Against AH Transport + ESP Tunnel

Here, the sequence of headers and trailers in the packets that are generated by IPsec is:

oIP	ESPh	iIP	AH	ULP	TFC	ESPt

where:

- oIP refers to the outer IP header,
- ESPh refers to the ESP header, following which all data fields are encrypted.
- iIP refers to the inner IP header,
- AH refers to the AH fields, including the AH MAC field and sequence number, with the scope of the MAC covering iIP, AH and ULP fields.
- ULP refers to the upper layer protocol data unit (e.g. a TCP or UDP message),
- TFC refers to optional TFC padding, and
- ESPt refers to the ESP trailer fields, including padding, and the PL and NH bytes.

3.1.1 *Attack 1: A Chosen Plaintext Attack*

Our first attack requires a single chosen plaintext and can recover arbitrary IPsec-protected plaintext. The attack exploits the fact that neither TFC padding nor ESP's normal encryption padding are protected by AH's MAC, and that, in accordance with [15], these bytes are discarded by the receiver before the inner packet is passed to AH.

Suppose for now that AH replay protection is disabled, and recall that ESP replay protection will always be disabled in this configuration. Suppose the attacker has available a single IPsec-protected packet of the above form, for which the inner packet has as its payload an ICMP echo request (i.e. the ULP block contains such an ICMP message), which can be directed either to the gateway G_B itself or to a host behind that gateway. Clearly, if this packet is injected into the network towards G_B, we will see an (encrypted) ICMP echo reply message in the reverse direction on the VPN between G_A and G_B. Moreover, because AH and ESP sequence number checking is disabled, this packet, if repeatedly injected into the network, will always cause such a response. This packet can be used directly as a carrier in an ESP trailer oracle attack, as described in Section 2.6. Here, ESP's handling of TFC bytes ensures that the inner packet presented to AH after ESP processing at G_B always passes AH's MAC check, even after the blocks R, C_i^* have been spliced onto the carrier packet. This is because after the ESP trailer is checked and removed, any remaining plaintext resulting from the spliced blocks together with the original ESP trailer will be interpreted as TFC padding and discarded. Moreover, none of these discarded bytes are covered by AH's MAC. So, with a single chosen plaintext and an average of slightly more than 2^{15} trial packet injections, any complete block of plaintext can be recovered.

This attack applies no matter what are the key-size and block-size of ESP's encryption algorithm. It can also be applied if the ULP block carries TCP instead of ICMP: now every received TCP segment provokes a TCP ACK packet of some type in response, so every modified carrier packet that passes ESP processing at G_B will generate a detectable message in the reverse direction. Even if the TCP connection for the TCP segment in the carrier packet is already closed, a TCP RST packet will be sent in response, so our attacker will always get the response he requires. This applies whether the endpoints for the TCP connection are the gateways themselves or hosts behind these gateways. Assuming that ULP carries a TCP message is a mild chosen plaintext assumption. This can be replaced by an even weaker assumption by simply observing packets to see which ones generate replies, and then using one of those packets as the carrier packet.

Attack 1 with AH replay protection enabled.

We can extend the above attack to the case where sequence number checking is on. The attacker first gathers, for each byte (or pair of bytes in case of the rightmost bytes) of plaintext that he wishes to extract, a packet that is expected to generate a reply. These packets might carry ICMP or TCP, for example. We make the assumption that the attacker can put these carrier packets in order of (roughly) increasing AH sequence number. This is reasonable, since they are likely to be intercepted in such an order. The attacker also needs to control the flow of packets on the network so that the sequence numbers in his carrier packets are always seen as being "fresh" during any AH processing at G_B for the duration of his attack. This can be achieved by firstly blocking all other packets from G_A to G_B except the attacker's carrier packets during the attack, and secondly by switching to the next carrier packet each time a response packet is detected on the VPN between G_A and G_B. The latter step coupled with our assumption about AH sequence number ordering ensures that, each time ESP trailer processing completes and AH processing is done, packets are not rejected by AH because they have repeated (or old) sequence numbers. Otherwise, the attack is as before.

For ease of presentation, we have described a simple version of the attack that requires the attacker to control the flow of traffic during the attack. It can be adapted to be less disruptive to traffic flow by making use of carrier packers as they become available to the attacker, but this would be more complex to implement.

The only drawbacks of Attack 1 are its very mild assumptions about the nature of plaintexts, its consumption of multiple carrier packets when AH replay protection is enabled, and the complexity of implementing the attack in a non-disruptive manner in this case.

3.1.2 Attack 2: TTL Expiry

Our second attack exploits the fact that the AH MAC cannot cover all the fields of the inner header, iIP. In particular, the TTL and checksum fields are unprotected and so can be manipulated by the attacker. This attack allows us to relax the plaintext requirements in comparison to the previous attack. However, we require that IP packets on the VPN are directed to hosts behind G_B. Again, we begin by assuming that AH replay protection is disabled.

Attack 2, Step 1.

We begin with a one-time preparation step. Suppose the attacker captures an arbitrary IPsec-protected packet intended for a host behind G_B. The attacker can manipulate bits in the IV of the CBC-mode ciphertext after the ESP header, with the effect of reducing the TTL field in the inner header to 0. This requires the header checksum to be corrected, and here we rely on the improved method described in Section 2.5. For example, supposing the TTL field's original value is `0x40`, then on average 2 trials are needed, while if the original value is `0xFF`, then on average 6.75 trials are needed. Alternatively, we might only assume that the TTL field is uniformly distributed; then, by carefully scheduling the bit flips applied to the TTL and checksum fields in an extension of the method of Section 2.5, we can simultaneously reduce the TTL field to 0 and correct the checksum using an expected number of 382 trials.

In each case, after a certain expected number of trials, the attacker succeeds in creating an IPsec-protected packet for which the TTL field in iIP is 0, the checksum for iIP is correct, and the AH MAC on the inner packet verifies. Because the inner packet should be forwarded to a host behind the gateway G_B, such an IP packet should always induce G_B to produce an ICMP response (of type 11 and code 0). We will use this IP packet in step 2 as a carrier packet[3].

Attack 2, Step 2.

The attacker now mounts an ESP trailer oracle attack using the carrier packet constructed in step 1, splicing blocks R, C_i^* onto the end of the carrier packet for different values R, starting with the 2^{16} variants in the rightmost 2 bytes of R. As with Attack 1, we rely on ESP's handling of TFC bytes to ensure that the inner packet presented to AH after successful ESP processing at G_B always passes AH's MAC check, even with the blocks R, C_i^* spliced onto the carrier packet. On average, after 2^{15} trials, an ICMP response will be detected in the reverse direction on the VPN between G_A and G_B. This indicates a particular value of R for which the packet ending in R, C_i^* passed the ESP trailer checks. The attack now continues in the usual way.

This attack only applies for encryption algorithms with 128-bit block size, because we must be able to manipulate the TTL field in the inner IP header, and this is located beyond the first 64 bits of the header. In step 2, the attack requires an average of $2^{15} + 14 \cdot 2^7$ trial packet injections to recover any complete 128-bit block of plaintext.

Attack 2 with AH replay protection enabled.

We can modify the above attack to cope with the situation where AH replay protection is enabled. The main difference is that we can no longer re-use a single carrier packet constructed in a first step, because once AH processing has been triggered (after successful ESP processing), a fixed carrier packet's AH sequence number would always be rejected thereafter. To overcome this, we combine the carrier packet generation and ESP trailer oracle steps. Thus, for each choice of R used in a normal attack, we must splice R, C_i^* onto a sequence of trial packets, with each trial starting with a base packet and attempting to manipulate the TTL field, correct the checksum, pass ESP trailer processing, pass AH processing, and finally generate an ICMP message. Clearly, for each success in this endeavour, the attacker can extract 2 or 1 plaintext bytes (depending on whether the rightmost bytes are being targetted or not), and must move on to a new base packet with a fresh sequence number for each success.

For an assumed inner TTL field of, say `0xFF`, an average of 6.25×2^{16} trials are needed to extract the rightmost 2 bytes of any block, and an average of 6.25×2^8 trials for each byte thereafter. Extracting each block of plaintext requires the attacker to have gathered 16 IPsec-protected packets with roughly increasing AH sequence numbers, and also to block

[3]In the case when the starting value of the TTL field is not known, we need to be careful to distinguish this ICMP response from any other replies that may arise when the IP header checksum is correct but the TTL has not been successfully set to 0.

other traffic on the VPN while the attack is in progress. If nothing is assumed about the starting TTL value, then the attacker would first conduct a reconnaissance phase to ascertain likely TTL values (since only a few possible different values would be expected, depending on the particular OS involved and the number of hops between the end host generating the inner packet and the gateway G_A). This would involve testing possible TTL value and checksum correction masks in a systematic manner in an effort to produce an ICMP response, with an expected number of 382 trials being needed (assuming the TTL field is uniformly distributed). Once the likely TTL values have been determined, the attack can proceed as just described for known TTL values. The attack can still be mounted without a reconnaissance phase, or with unstable inner TTL field values, but it becomes rather expensive in terms of the number of packet injections needed.

3.1.3 Attack 3: Fragmentation

In the previous two attacks we endeavoured not to tamper with authenticated portions of payloads, instead making use of intercepted packets that generate some form of reply at the receiver, or by manipulating portions of the ESP payload that are not protected by AH. Our third attack adopts a different approach, managing to avoid the plaintext requirements of the previous two attacks. We now craft packets that will generate replies whilst completely bypassing AH processing at the receiver. The basic idea is that, after ESP decapsulation of a crafted packet, the receiver discovers that the ESP payload contains only a fragment of the packet that was originally protected by AH; since AH's MAC cannot be verified unless the receiver has the complete packet, the MAC check will not occur and AH will enter a state in which it waits for further fragments. Eventually, this state will time-out, and generate an error message that is detected by the adversary.

Attack 3, Step 1.

We begin with a one-time preparation step. Suppose the attacker captures an arbitrary IPsec-protected packet intended for G_B. The attacker can manipulate bits in the ID field and the MF and DF bits by flipping bits in the IV of the CBC-mode ciphertext after the ESP header, with the effect of turning the inner packet (that is still protected by AH) into something that is interpreted by the receiver as a fragment. Here, we need to set the MF bit, possibly unset the DF bit, and then use the ID field to compensate the checksum, as discussed in Section 2.5. (Alternatively, we can manipulate the fragment offset and ID fields with similar results.) This can be done even for a block cipher having a 64-bit block, and with a small number of trial masks to determine how to flip bits in the IV. In fact, because of the specific bit flips involved, at most 17 trial packets are needed. The attacker injects all the trial packets in rapid succession, then waits. All the packets will be successfully processed by ESP at G_B, where all but one will have incorrect checksums and be dropped silently by the gateway. The one that has a correct checksum will be interpreted as a fragment, so IPsec will wait for the arrival of further fragments in an attempt to reassemble the original packets *before* any further AH processing takes place at G_B. Eventually, because the further fragments never arrive, the first remaining fragment provokes the production of an ICMP fragment reassembly time exceeded message (of type 11 and code 1) in the reverse direction on the VPN between G_A and G_B, as per Section 2.3. Because of the predictability of the time-out interval, the attacker can correlate the time of appearance of this packet with the time of injection of the trial packets to determine exactly which trial packet was the first one with a correct checksum. This trial packet will be the attacker's carrier packet for the second step in the attack. Note that, whenever this packet is injected into the network towards G_B, it will eventually produce an ICMP response after a suitable time-out period.

Attack 3, Step 2.

Now that the preparation phase is complete, the attacker has a carrier packet that can be used to create an ESP trailer oracle. This step works largely as before: the attacker splices blocks R, C_i^* onto the end of the carrier packet for different values R, starting with the 2^{16} variants in the rightmost 2 bytes of R. Here C_i^* is any target block. He injects these 2^{16} trial packets into the network towards G_B, looking for an ICMP message in the reverse direction. He correlates the appearance time of the ICMP message with the injection time of trial packets in order to identify the value of R which led to the ICMP message being produced, again using the predictable nature of the fragmentation time-out. The packet with this value of R must have passed ESP processing, indicating that its trailer field ended with the bytes `00,04`. From this, the rightmost 2 bytes of C_i^* can be deduced in the usual way. The attacker now continues to extract bytes further to the left, again by modifying R, creating trial packets, injecting them and correlating the appearance time of the ICMP message with the injection time of trial packets to identify the successful value of R. Each subsequent plaintext byte that is extracted needs the injection of 2^8 trial packets.

The modifications made to the inner packet in this attack do not cause any problems for AH processing, because the attack bypasses this processing. In this sense, the attack exploits the non-atomic nature of IPsec processing, and the complexities arising from IPsec needing to support IP fragmentation. It works for 64-bit and 128-bit ciphers (using the fact that checksums can be corrected by manipulating the ID field for the 64-bit case). It has no known or chosen plaintext requirements and extracts complete plaintext blocks. Its only disadvantage is that, no matter how fast the attacker can inject the (roughly) 2^{16} trial packets needed, he must wait for the IP fragmentation time-out after each pair of bytes/individual byte. As noted previously, this time-out is recommended to be 60-120 seconds, though it is only 15 seconds in the OpenSolaris implementation. This, then, is the limiting factor for the rate at which the attacker can extract plaintext.

Attack 3 with AH replay protection enabled.

The key feature of this attack is that AH processing is bypassed altogether. Thus, the carrier packet created in step 1 of the attack continues to produce IP fragmentation time-outs even when used repeatedly in step 2. So, in this case, enabling AH replay protection does not present any additional barrier to the attack. In fact this attack is much easier to mount than our first two attacks when AH replay protection is enabled, because it has no chosen plaintext assumptions, only a single packet is needed in the attack, no control over the traffic flow is needed, and it avoids the complications required to implement the previous attacks without disrupting the traffic flow.

3.2 Attacking Other Configurations

Having given a detailed discussion of three different attack types against the "AH Transport + ESP Tunnel" configuration, we move on to other configurations in which AH is followed by encryption-only ESP. We omit discussion of "ESP (auth only) + ESP (enc only)" configurations: since the scope of AH's integrity protection is always greater than that of ESP, it is easy to see that any attack against some "AH + ESP" configuration will also apply to the corresponding "ESP (auth only) + ESP (enc only)" configuration.

AH Tunnel + ESP Tunnel.

Here the format of the IPsec-protected packets is:

oIP	ESPh	iIP	AH	iiIP	ULP	TFC	ESPt

where now there are 3 IP headers, an outer header, an inner header and an "inner-inner" header. Here, Attacks 1 and 3 still work with simple modifications, but Attack 2 does not, since the TTL field that needs to be manipulated is the one in iiIP, and this is protected by AH (and cannot be reached from ESP's IV any more).

AH Tunnel + ESP Transport.

Here the format of the IPsec-protected packets is:

oIP	ESPh	AH	iIP	ULP	TFC	ESPt

Here, Attack 2 does not work, since this attack needs to manipulate fields in iIP which can no longer be reached from ESP's IV because of the intervening AH bytes. In Attack 3 we forge an ESP datagram whose payload contains only a fragment of an AH-authenticated IP packet. This is only allowed to happen when ESP is in tunnel mode; in fact there is no way of indicating such an instance when ESP in transport mode is used. As such, Attack 3 cannot be mounted either.

Attack 1 requires some extra assumptions to make it work in this configuration. Firstly, the "expected" value of the NH byte in ESPt is `51`, indicating AH as the next protocol, rather than `04` as before. However, it may be that more byte values are accepted here by IPsec processing, depending on how liberal the IPsec policies are at the gateway. This increases the success probability when extracting the rightmost 2 bytes, but may leave some uncertainty about the exact value of the rightmost byte of the recovered plaintext block. In practice, only `51` for the NH byte will lead to the production of a response message, since other values will lead to the AH data bytes being misinterpreted as coming from a different upper layer protocol, and the data will most likely not be correctly formatted for that protocol.

Secondly, the attacker relies on ESP processing at G_B to interpret the original data in ESPt and some of the bytes in the spliced blocks R, C_i^* as being TFC padding, and to be able to remove these bytes before submitting a packet of the form oIP–AH–iIP–ULP to AH processing. For, otherwise, the packet would contain extra bytes and these would cause the AH MAC verification to fail. This requires the ESP implementation at G_B to support TFC padding for transport mode ESP, and to know how to inspect the AH and iIP length fields to calculate how many bytes of data should remain after TFC padding has been removed. This places greater expectations on the IPsec implementation, though [15, Section 2.7] states that an IPsec implementation SHOULD be capable of this behaviour.

AH Transport + ESP Transport.

Here the format of the IPsec-protected packets is:

IP	ESPh	AH	ULP	TFC	ESPt

where now only a single IP header is present. This configuration was explicitly ruled out in the previous IPsec architecture [13], and so is not supported by some implementations (e.g. OpenSolaris) but is by others (e.g. Linux). Here Attack 2 fails because there is no inner IP header to manipulate, and Attack 3 does not work, since ESP is in transport mode.

Attack 1 requires some modification in order to work. As with the previous attack, the NH byte in ESPt is now expected to have value `51`, and the attacker must rely on ESP processing at G_B to accurately handle TFC padding. This requires the ESP implementation at G_B to support TFC padding for transport mode ESP and the upper layer protocol to include an explicit length field, ruling out the use of TCP in the payload of the carrier packet. This means that we are restricted to using ICMP (or perhaps some kind of UDP packet that always produces a response). Otherwise, the attack works as described previously.

4. EXPERIMENTAL RESULTS

Having described how our attacks should operate for an RFC-compliant implementation of the RFCs, we now turn to their experimental validation.

Our experimental set-up is composed of two desktop machines acting as the two stand-alone gateways, a laptop acting as the attacker's platform, and a 10 Mbit Hub. The gateways run OpenSolaris build 134, whereas the attacker's platform runs Linux 2.6. We implemented our attacks in Python 2.6.4 and used Scapy 2.0 to intercept and manipulate IP packets and to re-inject them into the network. We decided to use OpenSolaris because it can be configured to perform full padding checks on ESP-protected packets as recommended by [15]. In our experiments, the two gateways were configured to protect their communications using AH in transport mode followed by ESP in tunnel mode, as in the example configuration of Section 3.1. We have not tested our attacks on the other MAC-then-encrypt configurations, but we see no reason why they should not be successful.

We successfully implemented Attacks 1 and 3 on the aforementioned configuration, with the AH replay protection service both disabled and enabled. In OpenSolaris, when keys are set up manually, then replay protection is disabled and there is no way of enabling it (in conformance with [14]). Thus we used manual keying for the scenario where replay protection is disabled, and enabled automated key exchange using IKE in order to turn on the replay protection service. Attack 2 works by generating an ICMP message at the point where the IPsec gateway is about to forward the decrypted packet to the end host in the protected network. As mentioned above, this attack works only when AH is applied in transport mode followed by ESP in tunnel mode. However we discovered that in OpenSolaris, it is not possible to forward packets that are protected by AH in transport mode, preventing us from testing Attack 2 in our experimental set-up. This seems to be a design decision by the OpenSolaris developers: such configurations are perfectly in line with the IPsec RFCs.

All of our attacks rely on the production of ICMP messages, so one might be concerned about the effects of ICMP rate limitation. However, this is not an issue in practice because of the relatively slow speed at which the ICMP packets are produced. In fact the main complication that arises in practice for our attacks is the problem of distinguishing the desired response packets from other IPsec-protected traffic on the VPN. This of course depends on the amount of traffic present on the network. As a first step, if the attacker is able to predict the length of the response packet, then he can filter out all packets whose length does not match this value, and thereby significantly reduce the rate of false positives. If length filtering is not enough or not possible, then one can filter on the basis of "causation": assuming no network congestion, a response is expected to be seen almost instantly after the packet that caused it was received by the gateway. That is, with sufficiently high probability, the attacker can expect to observe the response within a short time interval of the packet having a correct ESP trailer being sent. The time interval should be short enough that the probability of a false positive appearing within the interval is low. Thus the attacker allows a time interval δt between each attack packet that he sends. Once the attacker has detected what he suspects to be a response packet, he can confirm that this was indeed the case by retesting and checking whether a response is again sent within time δt (this will require the use of a fresh carrier packet whenever replay protection is enabled). This can be repeated multiple times in order to boost the confidence of the detection procedure. Thus in scenarios with high network traffic levels, the attacks may still be realised at the expense of efficiency. Alternatively, if replay protection is disabled or Attack 3 based on fragmentation is used, the attacker can simply capture the packets that are of interest to him and wait for a period of low network traffic in order to carry out his attack.

In our set-up we had minimal spurious network traffic and thus basic filtering based on packet lengths was sufficient. The most computationally intense part of each attack is to extract the rightmost two bytes of the target ciphertext block. Given that we could distinguish a response packet from other traffic accurately enough, we adopted the following strategy in order to speed up the Attack 1: we transmitted all 2^{16} packets at a rate almost equal to the network's capacity. As soon as a response packet was detected, we replayed the last few packets spaced at a greater interval, in order to pinpoint the exact packet which generated the response. We also followed this approach in order to extract the rest of the bytes. On a 10Mbit hub, Attack 1 took on average 70 seconds to recover a 128-bit block of plaintext using a 140-byte carrier packet. It should be noted that if replay protection is enabled, then Attack 1 needs 30 fresh packets to recover a block of plaintext in the manner just described. On the other hand it is possible to sacrifice the attack's time efficiency by transmitting packets at a lower rate such that a packet generating a response can be immediately identified, thereby requiring only 15 fresh packets per 128-bit block of plaintext.

For Attack 3, a similar strategy was adopted. Now the oracle response is only output after the IP fragment reassembly has timed out. In OpenSolaris the default time-out value is 15 seconds. In order to match a response to the packet that generated it, we keep a list of the time instants at which each packet was sent. Then if a response is seen at some time t we search our list for packets that were sent near to the time $t - t_{\text{time-out}}$. This scheme was combined with the method described above where packets are initially sent in a burst, and then two replies are required to accurately locate the packet generating the response. In our experiments, we could locate the packet to lie within a range of roughly 20 packets with the first response, and then replay each packet at intervals of 0.2 seconds and use the second response to locate the desired packet exactly. Following this approach with our experimental set-up Attack 3 recovered a 128-bit block of plaintext in roughly 10 minutes.

5. CONCLUSIONS

In this paper, we have demonstrated attacks against all MAC-then-encrypt configurations of IPsec. These show that such configurations should be avoided in IPsec deployments. We have not found any attacks against encrypt-then-MAC configurations of IPsec.

Support for AH is no longer required in IPsec implementations *"because experience has shown that there are very few contexts in which ESP cannot provide the requisite security services"* [13]. Our work shows that the only configurations where ESP alone cannot easily mimic what can be done using AH and ESP in combination, namely those using AH followed by ESP, are actually *in*secure. So not only is AH not very useful, it could actually be considered harmful. The removal of AH from the IPsec standards is already under way: support for AH is not required in the current IPsec architectural RFC [13], whereas it was in the previous version [10]. Our results should provide motivation to accelerate this process.

Our attacks demonstrate the dangers inherent in exposing cryptographic flexibility to users. IPsec in particular places a significant burden on network administrators, requiring them to have sufficient cryptographic expertise in order to select secure configurations. Nothing prevents curious administrators from going "off piste" or protects them from bad advice, such as that to be found in [6, 7, 21] for example. We hope that the attacks given here will illustrate some of the dangers in an accessible form.

Our view of IPsec echoes that expressed in [6]: IPsec, in attempting to be "all things to all men" ends up compromising on security. It would be helpful to standardise IPsec profiles addressing particular application scenarios rather than allowing a set of components that can be combined and configured in different ways to achieve arbitrary goals. This is because predicting the combined security of distinct cryptographic primitives is quite difficult and requires thorough analysis. While theoretical cryptography has much that is useful to say on this subject [2, 16], it currently falls short of being able to give truly meaningful security guarantees for cryptographic primitives as they are deployed in real protocols. Recent work [19, 20] has started to address this gap, but there is much still to be done to bridge theory and practice in this area.

6. REFERENCES

[1] S. Bellovin, "Problem Areas for the IP Security Protocols." In *Proceedings of the Sixth Usenix Unix Security Symposium*, pp. 1–16, San Jose, CA, July 1996.

[2] M. Bellare and C. Namprempre, "Authenticated Encryption: Relations Among Notions and Analysis of the Generic Composition Paradigm." In *T. Okamoto, ed., Asiacrypt 2000*, LNCS Vol. 1976, Springer, 2000, pp. 531-545.

[3] R. Braden, editor, "Requirements for Internet Hosts – Communication Layers", RFC 1122, Oct. 1989.

[4] B. Canvel, A.P. Hiltgen, S. Vaudenay and M. Vuagnoux, "Password Interception in a SSL/TLS Channel." In *D. Boneh (ed.), CRYPTO 2003*, LNCS Vol. 2729, Springer, 2003, pp. 583-599.

[5] J.P. Degabriele and K.G. Paterson, "Attacking the IPsec Standards in Encryption-only Configurations." In *IEEE Symposium on Privacy and Security*, IEEE Computer Society, 2007, pp. 335-349.

[6] N. Ferguson and B. Schneier, "A Cryptographic Evaluation of IPsec", 2003. Available from `http://www.schneier.com/paper-ipsec.pdf`

[7] N. Ferguson, B. Schneier and T. Kohno. Cryptography Engineering. John Wiley & Sons, 2010.

[8] R. Housely, "Using Advanced Encryption Standard (AES) Counter Mode With IPsec Encapsulating Security Payload (ESP)", RFC 3686, Jan. 2004.

[9] C. Kaufman, editor, "Internet Key Exchange (IKEv2) Protocol", RFC 4306, Dec. 2005.

[10] S. Kent and R. Atkinson, "Security Architecture for the Internet Protocol", RFC 2401, Nov. 1998.

[11] S. Kent and R. Atkinson, "IP Authentication Header", RFC 2402 (obsoletes RFC 1826), Nov. 1998.

[12] S. Kent and R. Atkinson, "IP Encapsulating Security Payload (ESP)", RFC 2406, Nov. 1998.

[13] S. Kent and K. Seo, "Security Architecture for the Internet Protocol", RFC 4301 (obsoletes RFC 2401), Dec. 2005.

[14] S. Kent, "IP Authentication Header", RFC 4302 (obsoletes RFC 2402), Dec. 2005.

[15] S. Kent, "IP Encapsulating Security Payload (ESP)", RFC 4303 (obsoletes RFC 2406), Dec. 2005.

[16] H. Krawczyk. The Order of Encryption and Authentication for Protecting Communications (or: How Secure is SSL?). In *J. Kilian, ed., CRYPTO 2001*, LNCS Vol. 2139, Springer, 2001, pp. 310-331.

[17] V. Manral, "Cryptographic Algorithm Implementation Requirements for Encapsulating Security Payload (ESP) and Authentication Header (AH)", RFC 4835, April 2007.

[18] K.G. Paterson and A.K.L. Yau, "Cryptography in Theory and Practice: The Case of Encryption in IPsec." In *S. Vaudenay (ed.), EUROCRYPT 2006*, LNCS Vol. 4004, Springer, 2006, pp. 12-29.

[19] K.G. Paterson and G.J. Watson, "Plaintext-Dependent Decryption: A Formal Security Treatment of SSH-CTR." In *H. Gilbert (ed.), EUROCRYPT 2010*, LNCS Vol. 6110, Springer 2010, pp. 345-361. Full version available from `http://eprint.iacr.org/2010/095`.

[20] P. Rogaway and T. Stegers, "Authentication without Elision: Partially Specified Protocols, Associated Data, and Cryptographic Models Described by Code." In *CSF 2009*, IEEE Computer Society, pp. 26-39.

[21] W. Stallings. Network Security Essentials: Applications and Standards, 3rd edition. Pearson Education, 2008.

[22] S. Vaudenay, "Security Flaws Induced by CBC Padding – Applications to SSL, IPSEC, WTLS..." In *L.R. Knudsen (ed.), EUROCRYPT 2002*, LNCS Vol. 2332, Springer, 2002, pp. 534-545.

[23] D. Wagner and B. Schneier, "Analysis of the SSL 3.0 Protocol." In *The Second USENIX Workshop on Electronic Commerce*, USENIX press, 1996.