



## Research Center

NRC-TR-2007-010

# Design of IPsec and IKE version 1 and 2

Yogesh Prem Swami

Nokia Research Center  
955 Page Mill Road, Suite 200  
Palo Alto, California  
USA  
<http://research.nokia.com>  
April 10, 2007

### Abstract:

IPsec is a collection of protocols that provides network layer data integrity and confidentiality services. IKEv1 is a versatile key agreement protocol that allows perfect forward secrecy and identity protection (among other things). IKEv2 has similar functionalities as IKEv1, but provides a simpler and better approach to key exchange. Additionally, IKEv2 provides new methods for authentication and traffic selection. The goal of this paper is to analyze IPsec as a solution for IP layer security protocol.

### Index Terms:

Internet Security  
Internet Key Exchange  
IPsec  
Key Exchange Pitfalls  
SIGMA Protocol

## Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>What is Security?</b>	<b>3</b>
<b>3</b>	<b>Internet Threat Model</b>	<b>3</b>
<b>4</b>	<b>IPsec Design Basics</b>	<b>5</b>
4.1	Constraints from the IP layer . . . . .	5
4.2	Replay Protection . . . . .	5
<b>5</b>	<b>Authentication Header (AH)</b>	<b>7</b>
<b>6</b>	<b>Encapsulating Security Payload</b>	<b>8</b>
<b>7</b>	<b>Security Association Management</b>	<b>8</b>
7.1	Security Policy Database (SPD) . . . . .	9
7.2	Security Association Database (SAD) . . . . .	9
7.3	Peer Authorization Database (PAD) . . . . .	10
<b>8</b>	<b>Key Exchange Pitfalls</b>	<b>10</b>
8.1	Diffie-Hellman (DH) Key Exchange . . . . .	11
8.2	Diffie-Hellman Pitfalls . . . . .	12
<b>9</b>	<b>IKEv1</b>	<b>15</b>
9.1	Main Mode . . . . .	18
9.2	Aggressive Mode . . . . .	21
9.3	Quick Mode . . . . .	21
9.4	IKEv1 Problems . . . . .	21
<b>10</b>	<b>IKEv2</b>	<b>22</b>

## 1 Introduction

IPsec and IKE are complex protocols. The goal of this paper is to critically evaluate IPsec from a protocol design perspective. Although this paper is self contained, it is assumed that the reader has at least a basic understanding of network security and cryptography.

## 2 What is Security?

Security, regardless of what form it takes, is about balancing the risk of “value lost,” to the benefits of “value gained.” For the owner of a precious item, the value gained *through security* is the reduction in the *probability* of getting that item stolen; the value lost is the cost and *inconvenience* incurred in protecting that item. Similarly, for an attacker, the value lost is the probability of being caught and prosecuted for the crime, while the value gained is the value derived from owning a particular object.

When calculating value, it is important to take unquantifiable factors, such as emotions, into consideration. For example, the monetary value of a few pictures might not be too high, but those pictures might have extraordinary emotional value. The same applies to the “cost of security.” The cost of buying a crypto-software might not be too high monetarily, however, if the software frustrates the user by the way it works, then the overall cost of security could be very high. For example, a Firewall that asks its user to answer a “yes/no” question every time an application initiates a TCP connection can easily frustrate its users and impose a very high level of emotional cost. Because of this high perpetual emotional cost, it is extremely likely that such a firewall will be ultimately disabled by the user.

Understanding and judging the value of a security solution should be the first crucial step in designing a secure system. Although overestimating the threat is better than underestimating it, too much unwarranted security could easily become a nuisance — and ultimately less secure. In the Firewall example, even though the highly interactive firewall may seem more secure than a non-interactive firewall, yet the high probability that the users will disable it makes the system less secure than a silent firewall (which will remain enabled by the user).

## 3 Internet Threat Model

Key to understanding the value of a security solution is to understand the underlying threat model it attempts to solve. The threat model also provides us with insights into the protocol and explains why the protocol was designed the way it was. Understanding the threat model is essential from another perspective too: It allows us to see when the protocol will work satisfactorily and when it will not. For example, a car lock will work satisfactorily on cars (assuming it’s made well), but not the garage door. Many people have come to inappropriately believe that IPsec is the cure for all network security problem. Unfortunately, IPsec doesn’t explicitly states its threat model, and that is part of the reason for this confusion.

The threat model that I describe here is based on my own understanding of IPsec and might not be complete. I suspect that at least a part this threat model is motivated by the requirements from US military. Following is list of threats that IPsec attempts to address:

- That the bad guys — humans and viruses alike — are lurking behind each router, eager to intercept and analyze messages between the communicating parties. Conceptually this is a valid threat and the fundamental reason for using network security.
- That the bad guys are capable of storing unlimited amounts of encrypted data on their hard-drives to use it later on. Although these bad guys might not have the skills to cryptanalyze the data, they might still try to resend (replay) part of their collection for illegitimate reason (or just for fun). This is also a valid threat. Such attacks are known as replay attacks.
- That the bad guys can in some cases get a complete hold of the traffic between two nodes (Alice and Bob) and might try to impersonate Alice to Bob, and Bob to Alice.

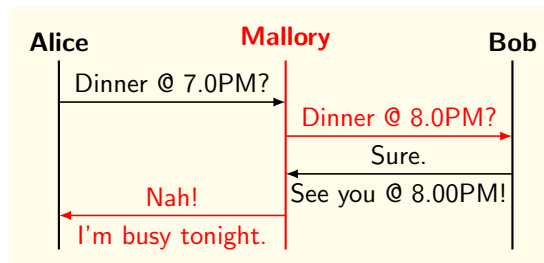


Figure 1: Example of Man-in-the-Middle (MITM) attack.

Let's take an example to understand this threat better (see Figure 1, for details). Let's say Mallory is a bad router and all the traffic between Alice and Bob goes through Mallory. Now let's say Alice sends a message to Bob saying: "Alice here: Dinner @7.00PM?" Since this message goes through Mallory, she can change the message as: "Alice here: Dinner @8.00PM?" (Notice that Mallory did not change the name, only the time.) Bob, who has a lot of free time (and is not interested in writing a paper on IPsec) agrees by responding: "Sure Alice. See you @8:00PM." Since this message also goes through Mallory, she changes the message again as: "Nah! I'm busy tonight." Notice how Mallory has disrupted Alice's plans. Also notice, how extremely difficult it is for Alice or Bob to figure out who — if any one of them — is lying (repudiating).

Man in the middle (MITM) attack — as it's popularly known — is an over blown threat for the Internet! Historically, there have been extremely few reports of such attacks. (And, yes, I am aware that the reason could be that most modern protocols already take MITM into account, so there cannot be any report of MITM attacks, but I still maintain my position.)

- That the principals are trustworthy. In other words, if Bob wants to communicate with Alice, and Alice sends a proof that she is Alice then Bob has no reason to suspect that the person on the other end is not Alice.

This might sound like a reasonable assumption, but there is a small caveat. Let's say Alice was working on her desk and after some time leaves her desk, without logging out, to drink water. In the meanwhile, while Alice is gone, Trudy comes to Alice's desk and starts using Alice's computer. If Alice's security credentials are stored on that laptop — as they usually are — then even Trudy can provide the proof to Bob that she is actually Alice. Note that Trudy doesn't have to be a human — it can be a virus.

This is not necessarily a problem within the scope of IPsec, rather a problem for platform access control and security. Unfortunately, the security of a system is often worse than that of the weakest link in the system and therefore it is important to keep such assumptions in mind, so that the protocol could minimize their dependency on such assumption.

- That end users might need identity protection. In IPsec any "name" that is somehow verifiable could act as an identity and the IP address of the node need not the identity. In fact, its common to have a Fully Qualified Domain Name (FQDN) or an e-mail address as the identity. Typically, the identity is associated with a <public-key, private-key> pair, where the public key and the identity are bound together through a digital signature.

Identity protection might sound like a bizarre requirement. How can a node *X* protect its identity, when that very identity is needed by other nodes to identify *X*. In IPsec identity protection means that a passive observer, who can just snoop IP packets, is not be able to find the name of the communicating parties. Clearly, in case of an active attack (*i.e.*, when the attacker is not just a passive listener but could actively send, intercept, and delete packets), it will be hard to protect the identity of both the sides. (It is still possible to protect the identity of one at least one side.)

Identity protection is a useful feature. But it should take lower precedence over other security requirements.

In addition to these threats, IPsec must protect itself against well known attacks such as the Denial of Service (DoS) [13]<sup>1</sup> attacks.

Although, some of the threats described above might seem far-fetched, it's important to understand that regardless of what threat model you choose, it will not be appropriate in all the million ways that IP can be used. IPsec in my opinion, has taken a reasonable approach of providing strong security that can be used even when the threats are severe. The only disadvantage of this is that IPsec deployment has suffered because of the complexity that such a design engenders.

## 4 IPsec Design Basics

In order to address the above threat model, we need to consider the system constraints that are inherent to the design of IP. First IPsec will provide security at IP layer. However, the IP layer does not have any notion of “connection” or reliability. Therefore, the protocol must be designed to run in a connectionless manner. Furthermore, if any parts of the protocol needs reliability, it must be provide by IPsec or some application layer protocol.

### 4.1 Constraints from the IP layer

IPsec addresses the issue of reliability by separating parts of the protocol that require reliability from those that don't. Internet Key Exchange, which is the only component that requires reliability, is designed to run as an application (thereby preventing any need for IP layer reliability). The rest of IPsec, which provides encryption and authentication service, does not have any special reliability requirements and can run directly on top of IP. Delegating reliability to the application layer is a very smart design choice<sup>2</sup>.

The second issue is that of multicast. Because of multicast, our design cannot make any assumptions about the source-address of the packet because different source addresses can send packets to the same multicast group. In other words, the protocol must be designed based on the destination address only which, in case of multicast could be the multicast group address. (This is of course not to say that the source address cannot be used at all, say for policy purposes.) But if we only take the destination address into consideration, we have a new problem: How does the destination demultiplex different senders that it might be communicating with?

The problem of demultiplexing is solved by using a 32 (or 64) bit number that the destination randomly assigns to each IPsec protected “flow.” This number, called the Security Parameter Index (SPI), is chosen locally by a receiving host and included in each IPsec protected packet by the sending host to facilitate demultiplexing.

### 4.2 Replay Protection

Having addressed the constraints imposed by the IP layer, let's now focus our attention on the threat model. The threat model described in Section 3 could broadly be classified in two parts — attacks that are targeted toward the key-exchange protocol and those toward the IPsec protected data itself. MITM is an example of attack that is typically directed toward the key-exchange protocol. (MITM attacks on IKE are discussed in Section 8). Replay attacks are typically directed toward IPsec protected data, which is discussed below:

In a replay attack, the attacker collects data from an old session and later on tries to inject that old data in hope that it will get reprocessed. In a very simplistic case, assume that two end-points Alice and Bob have agreed on a long term key  $K_{A-B}$ . Let's say Alice sends a packet  $P_1$  authenticated and encrypted using  $K_{A-B}$  to Bob at time  $t$ . Because only Alice and Bob know  $K_{A-B}$ , no one can successfully tamper with  $P_1$ . However, let's say Mallory intercepts packet  $P_1$  at  $t$ , and lets the session continue. Then later at some time  $t + n$  when Alice and Bob have exchanged up to  $P_{n+1}$  packets, Mallory re-sends packet  $P_1$  to Bob, and hopes that Bob will reprocess it. Since  $P_1$  is encrypted and

<sup>1</sup>It seems that the networking community spends far too much time on DoS attacks compared to any other form of attack. However, I don't classify DoS vulnerabilities as a security threat, rather as an example of poorly designed system. This document doesn't discuss DoS attacks in detail mainly because of this reason.

<sup>2</sup>Many new protocols such as mobile-IP and HIP seem to be building their own retransmission and reliability scheme, right at the IP layer. This is unfortunate, since adding reliability is non-trivial and requires an accompanying implementation of congestion control algorithm, all of which could easily lead to a highly complex and inefficient protocol.

authenticated using  $K_{A-B}$ , Bob will gladly reprocess  $P_1$ . However, by accepting  $P_1$ , Bob is tricked into performing the same operation twice. If  $P_1$  contained the instruction: “I’m Alice. Bob please deposit million dollar to Mallory’s account,” then by reprocessing  $P_1$ , Bob would be erroneously depositing two million dollars into Mallory’s account. Clearly, the way to address such an attacks is to have a sequence number in each packet — and have the sequence number included in the packet’s Integrity Check Value (ICV).

IPsec prevents replay protection by adding a per-packet sequence number field in the Authentication and Encryption payload header. (Careful: the sequence numbers are *per IP packet put on the wire*. Therefore, even when the upper layer such as TCP might be retransmitting the same packet, the sequence number at IPsec must be incremented.) Because the header fields used in the protocol must be of finite size, sooner or later the sequence number used will wrap around. However, using the wrapped around sequence number will once again open doors for replay attack. Therefore, in order to deal with wrap around sequence numbers, the two end points must change their session key before the sequence number wraps around. By changing the session key, the old packets ( $P_1$ , in the previous example) will not pass the authentication engine and will be rejected.

Sequence numbers still have one problem: How does the IPsec engine handle out-of-order packets? IPsec uses a sliding window with bit-masking to address out-of-order packets. Let’s assume that the IPsec policy requires that the system could accept up to a maximum of 32 packet reordering (*i.e.*, the assumption is that no network will reorder more than 32 packets). To accommodate 32 packet reordering, IPsec keeps a 32-bit, bit-mask, where each bit represents whether the packet has been received out of order or not. For each incoming packet that has passed the *authentication check*, the IPsec engine checks to see if the packet is out-of-order and if its sequence number falls within the window. If the packet is outside the range of the window, it is silently discarded. If it is within the range, then the bit corresponding to that sequence number is checked to see if that sequence number has already been received. If the packet is not received, then the bit-mask must be zero, and hence the IPsec engine could accept the packet and set the bit. If the bit-mask is non-zero, then the packet must have been received before, indicating that either the packet was duplicated in the network or someone is trying a replay attack. In either case, discarding the packet is the right option. Note that the packet must pass the authentication checks before the bit-mask value is checked.

To illustrate how replay-protection works, let’s assume that the left edge of the IPsec window corresponds to sequence number  $X$ . Therefore, the window within which the packet must arrive is  $[X, X + 32]$ . Let’s assume that the bit mask  $U$  at this point is  $0x0000,0000$  (*i.e.*, no packets are out of sequence). Let’s assume that a packet with sequence number  $X + 3$  arrives that has passed the authentication checks. Because  $X + 3$  has passed authentication checks, and it falls within  $[X, X + 32]$ , the bit mask  $U$  is checked to see if the packet has already been received before. Since, the packet was not received before, the  $3^{rd}$  bit of  $U$  is set to 1, *i.e.*,  $U=0x2000,0000$ . Now let’s assume a packet arrives that passes the authentication check, but has the sequence number  $X + 3$ . Since the  $3^{rd}$  bit is already set, this packet must have been duplicated somewhere in the network (either by a routers or by an attacker), and therefore the IPsec engine can safely discard it. Now let’s assume the packet  $X + 1$  arrives. Since  $X + 1$  will move the window towards right, the system updates the window boundaries from  $[X, X + 32]$  to  $[X + 1, X + 33]$ . However, just updating the window boundary is not enough — the location of bits in the bit-mask also need to be updated. Updating the location of the bit is simple: The bit mask just needs to be rotated towards right by the number of packets the window has shifted toward right. Therefore, on the reception of packet  $X + 1$ , the bit-mask  $U$  will get updated as  $(0x2000,0000 \ll 1) = 0x4000,0000$ . On reception of the packet  $X + 2$ , there will be no gap left since  $X + 3$  is already received. Therefore, the window must jump by two packets, and the bit-mask must be rotated by 2. Therefore at the end of receiving packet  $X + 2$ , the bit mask of  $U$  will become  $(0x4000,0000 \ll 2) = 0x0000,0000$ . For more details, see Appendix-C of RFC2401 [8].

Note that the IPsec window is designed to prevent replay protection, and as such doesn’t need to keep the out-of-order packets queued in its window. In other words, IPsec engine doesn’t try to do in-order packet delivery — the window is only to prevent replay protection. This is an important difference from TCP’s sliding window, for example.

Finally, one of the design decisions for IPsec was to allow a third-party (a gateway or a router), to do the IPsec processing on behalf of other nodes. The reason for this are two fold. First, not all hosts on the Internet (that need network layer security) could be easily upgraded with a new IPsec enabled operating system kernel. Therefore, by delegating security functionality to the gateways, a corporation could provide strong security services without requiring a massive corporate-wide OS upgrade. Second, even if hosts had native IPsec implementation, there is no guarantee

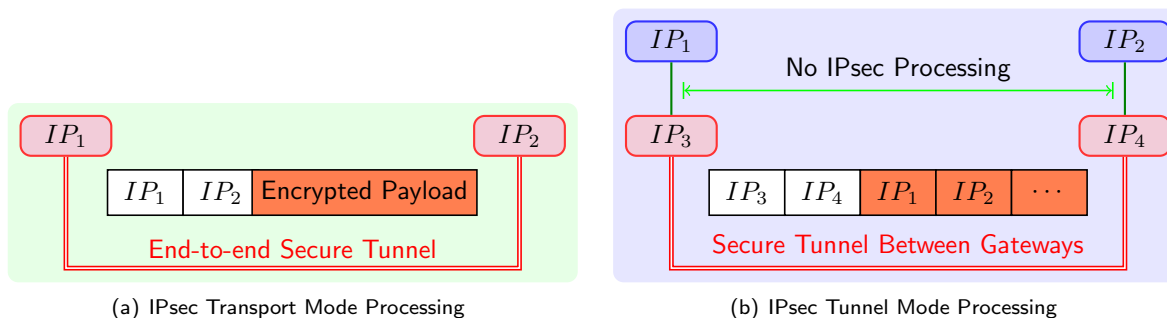


Figure 2: Transport and Tunnel Mode IPsec processing. The IPsec protected parts are shown in red.

that end users will use it — and use it well. End users often disable security functions because it “gets in their way of doing things,” and that leads to poor security. By having a gateway, the security policy could be enforced more aggressively. Using a gateway also allows a form of end user anonymity. An attacker who is sitting between the two IPsec gateways will not be able to tell to whom the traffic belongs (the attacker will of course see the IP address of the gateway, but that is typically a public information anyway).

IPsec documents call the gateway based IPsec processing as tunnel mode. In the tunnel mode, the original IP packet is put inside a new IP packet with the header of the two gateways. The old IP packet is then treated as payload and it can be encrypted and integrity protected just as normal data (with some subtle differences).

When hosts use IPsec directly without the intervention of a gateway, it’s called Transport mode. Figure 2 shows the difference between the transport mode and tunnel modes. As you can see, in transport mode, the original payload of the IP packet is encrypted, but the IP addresses are sent in clear. In the tunnel mode, however, the entire original IP packet is encapsulated inside a new IPsec packet. Note that there is no requirement on IP-3 in Figure 2(b) to be different from IP-1. Therefore, the tunnel mode could run between an end-point (such as a Laptop) and a security gateway. RFC2401 [8] describes different combinations in which the tunnel mode can be used in practical settings.

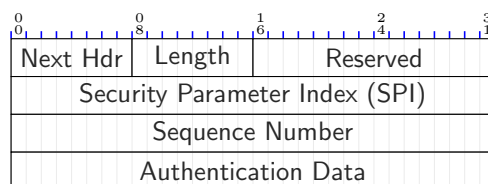


Figure 3: Authentication Header Format.

## 5 Authentication Header (AH)

The Authentication Header (AH) defined in RFC2402 [6] and RFC2402-bis [4] provides data integrity but no confidentiality protection. AH provides replay protection by using a 32 bit sequence number field, as described in Section 4.2. Figure 3 describes the AH header format. This header is inserted between the original IP header and the upper layer header (for example, TCP header) to provide data integrity for the packet. RFC2402 and RFC2402-bis describes the details of how the final AH packet is constructed.

The Authentication Header provides data integrity for the entire IP packet. However, there are fields in the IP packet that change en-route to its destination. These fields are referred to as Mutable Fields. While calculating the

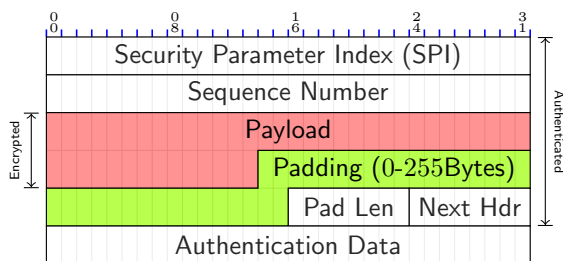


Figure 4: Encapsulating Security Payload Header Formats.

integrity check value of the packet, it's important to exclude the mutable fields at the time of calculating and verifying the Integrity Check Value. Consult RFC2402 and RFC2402-bis for details about the mutable fields.

## 6 Encapsulating Security Payload

Encapsulating Security Payload (ESP), defined in RFC2406 [7] and RFC2406bis [5], provides both confidentiality and integrity protection, although integrity protection is optional. Figure 4 describes how an ESP protected packet looks like. If you are wondering why we need AH when ESP already has integrity protection, then you are not alone in questioning that design decision. In principle, if I only want integrity protection, I can use NULL encryption to achieve the same (or similar) results as AH. The only advantage that AH has is that it contains some parts of the IP header in the Integrity Check Value (ICV) calculation — but that could have been solved by appropriately defining the authentication fields for ESP.

The second design flaw in ESP is that ESP allows encryption without authentication. So if an attacker can send a random set of bits with the right ESP header, *i.e.*, the right sequence number and SPI, (all of which is sent as plain-text) then the IPsec receiver will have no way of knowing whether that packet is valid or not, and in fact the IPsec receiver will gladly decrypt and forward the packet to upper layer.

But that's not a problem in itself, is it? The upper layer will discard the packets since the port numbers will not match. In fact, creating the cipher text that will get decrypted to the right port numbers is very hard. Therefore, the attacker has only 1 in  $2^{32}$  chance of getting both the port numbers right, and even less if you consider the TCP sequence number. But what if the attacker takes a legitimate packet, and leaves encrypted bits corresponding to the TCP header intact, but puts garbage inside the rest of the packet. The packet will now get processed all the way up to the application. But this attack is still not good enough: The checksum, which is encrypted, will not match if the packet contents are arbitrarily changed and upper layer protocol will again discard the packet (assuming the upper layer protocol uses checksum). But what if the attacker doesn't *change* the content, but just reorders the blocks in the encrypted message. In this case, the checksum of the entire packet will not fail because checksum is a commutative operation. While, this attacks may seem of very limited use, consider the case where the attacker replaces 00001000 to 10000000 for his bank transaction (assuming a 32 – *bit* encryption block-side).

Although checksum might seem like a weak form of integrity protection mechanism, the 16 bit of integrity protection field is too small to be useful. In fact, based on birthday paradox, an attacker has a very high probability of finding two messages from a search space of  $2^8$  messages that will have the same checksum.

To summarize, disabling integrity protection in ESP should be strongly discouraged.

## 7 Security Association Management

Encryption and Authentication are simpler (though not simple) to achieve, but the main difficulty lies in creating, maintaining, and enforcing security associations and policies. IKE which we discuss in Section 8 is used to create and manage security associations. In this section we discuss how security mechanisms are enforced.

When a packet arrives from the upper layer, say TCP, the first step is to decide whether the packet should undergo IPsec processing or not. In other words, the IPsec engine needs to work as a packet-filter that decides whether to process, drop, or bypass a packet. This operation must be done on each and every incoming or outgoing packet. Therefore, the first step in designing the system is to design a Security Policy Database (SPD) [8, 9] which describes the properties of the packet filter. This database will typically be populated by the System Administrator and will typically not change too frequently. This database will also indicate what forms of encryption and authentication algorithms should be used and what trust relationships and access control are to be provided.

Once the SPD database is constructed, the system needs to keep track of active security associations — things such as encryption keys, Initialization Vectors (IV), and sequence numbers are maintained in this database. In IPsec this database is known as Security Association Database (SAD) [8, 9].

Both SPD and SAD are databases related to per-packet processing. In addition to these databases, there is a need for a database that describes which hosts are allowed to communicate with which other host on the Internet. This database is known as Peer Authorization Database as described on RFC2401-bis [9]. For manual keying, this database is typically not needed — the peer authorization could be put in SPD where the filter rule is based on the IP address (as opposed to a network prefix). For automated keying, however, such a database is indispensable. The following subsections describe the details of different databases.

## 7.1 Security Policy Database (SPD)

Essentially, this database is not much different from any other Access Control Lists (ACL) that are widely used by firewalls. Entries in this database are ordered, and only the best match (longest prefix if IP addresses are used) to the IP packet should be used to find the right policy for packet processing. RFC2401 recommended using one database per interface, however, it's easier to have a single system wide database (and this is very well noted in RFC2401bis).

Logically, the SPD database can be classified into three other databases, described as SPD-S, SPD-I, and SPD-O. SPD-S contains entries for those packets that must be IPsec protected (*i.e.*, must require encryption or authentication). This database applies to both incoming as well as outgoing packets. SPD-I and SPD-O apply to those packets that don't require encryption or authentication functions. SPD-O is used for Outbound traffic, while SPD-I is used for Inbound traffic. For details about these database see RFC2401bis [9].

## 7.2 Security Association Database (SAD)

SAD contains one entry for each Security Association. Typically, this database is indexed by the Security Policy Index that is used in the AD/ESP header. At the time when security association is established (through IKE for example), the SPD caches the SPI information for the appropriate prefix and later uses the SPI to find entries in the SAD database.

When a packet arrives from upper layer for outbound packet processing, first the SPD database is consulted. Based on the "longest prefix" in the SPD database, the IPsec processing engine checks to see if it already has a cached SPI value for the longest prefix match. If this entry is found, the IPsec engine consults the SAD database and constructs the right ESP/AH packet and sends it on the wire.

For inbound packet processing, the SPI value in the packet is used to locate the SAD entry. Each SAD entry contains various information; the most important ones are listed below:

**Security Parameter Index:** SPI is the index in the SAD table.

**Sequence Number Counter:** These sequence numbers are same as the sequence numbers used in AH/ESP packets. In RFC2401, the default size of sequence numbers was 32 bits. In RFC2401-bis, this field has been extended to 64 bits by default. Consult RFCs for details.

**Anti-Replay Window:** This is the sequence-number range and the bit-mask values described in Section 4.2.

**Authentication and Encryption Algorithm:** These entries describe the actual algorithm to use for encryption and authentication.

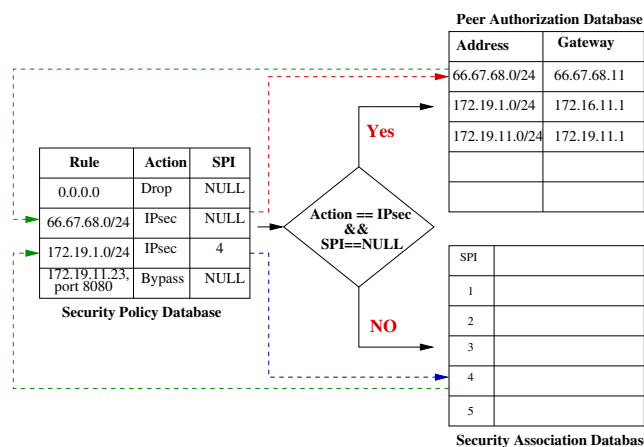


Figure 5: Relationship Between Different IPsec databases. Dotted Lines indicate pointers from one database to another.

**Lifetime of the Security Association:** Shannon's theory of Perfect Secrecy states that for perfect secrecy, the key size of the encryption algorithm should be at least as long as the message size itself. However, if we assume that each Key of the crypto-system gives rise to a family of random functions, then an  $n$ -bit key, could safely and efficiently encrypt messages of up to size  $2^{n/2}$ . This limit means that for every  $2^{n/2}$  bits encrypted using a key, the key must get updated. The idea behind having a lifetime entry in SAD database is derived from this observation. IPsec allows the lifetime to be described in terms of bytes or time-interval; however, it's better to use byte count than time.

### 7.3 Peer Authorization Database (PAD)

Both SPD and SAD are associated with per-packet IPsec processing. However, before an active SAD entry could be created, there is a need to do key-exchange that would populate the entries in SAD. The PAD database contains information that would allow key-exchange protocols to interact with their peers and populate the SPD entries.

One important function of PAD database is to find the right security gateway to communicate with for a given destination address. Assume that a TCP packet arrives with the destination address of 66.67.68.69. In order for SPD to offer security protection, it will request the key-exchange module to create the SA. However, in order to create the security association, IKE needs to know the security gateway which might not be located on 66.67.68.69. The role of PAD is to indicate to the IKE that the gateway for 66.67.68.0/24 is 67.68.69.11. Without the PAD entry, IKE cannot create the Security Association.

In addition to providing configuration information, PAD also contains information on how IKE identities (such as e-mail addresses FQDN) map to the SPD entries. Note that in tunnel mode, the SPD only has access to the IP packet — not to the IKE ID that was used. Therefore, one goal of PAD is to provide mappings for IDs to packet level information.

Figure 5 shows the relationship between different databases. For details, consult RFC2401bis.

## 8 Key Exchange Pitfalls

The first step in designing a Key Exchange protocol is to decide which algorithms to use. Since trust relationships in the Internet are often formed between unknown parties, it's unlikely that a symmetric-key based schemes will scale for a system as large and diverse as the Internet. Therefore, only schemes that depend upon public keys can be used. (Of course, once the key exchange is done, one can use symmetric keys.)

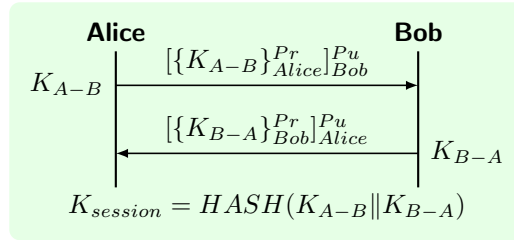


Figure 6: A Simple Key Exchange Protocol without Diffie-Hellman Exchange. Alice and Bob each select a random number  $K_{A-B}$  and  $K_{B-A}$  respectively to derive the session key. Encryption using private keys is shown as  $\{ \}_{user}^{Pr}$ . Encryption with public keys is shown as  $\{ \}_{user}^{Pu}$ .

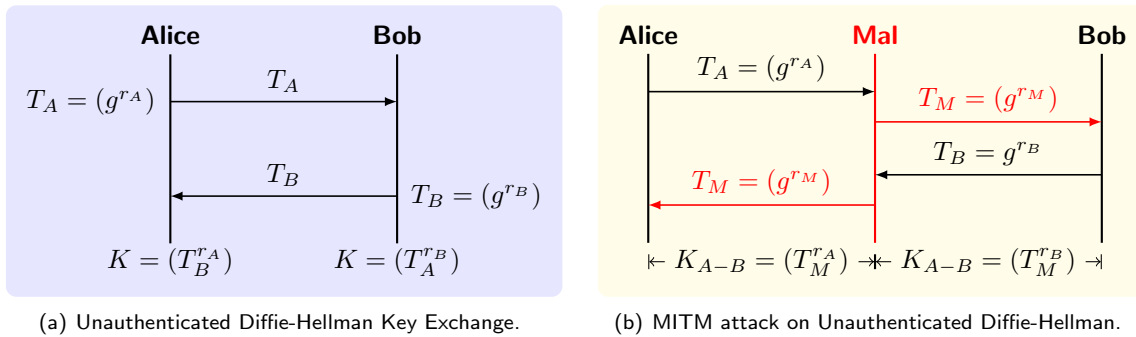


Figure 7: Basic Diffie-Hellman Key Exchange and MITM Attack. Numbers within  $(\cdot)$  indicate  $\text{mod } P$  arithmetic.

Among the public-key based schemes, it's possible to have a very simple scheme described in Figure 6 to create security associations. Note that in this scheme no one other than Alice and Bob can see  $K_{A-B}$  or  $K_{B-A}$ , and therefore the key exchange is secure. Although, there are many (solvable) problems with this key exchange, the one problem that is not solvable is that of Perfect Forward Secrecy (PFS). PFS requires that even if the long term keys (*i.e.*, public-private key pairs) of Alice and Bob broken, the attacker should not be able to glean information about the encrypted messages that took place *before* the keys were compromised. In other words, if the long-term keys of Alice and Bob are compromised at time  $t = T$ , then PFS requires that message exchange that took place before  $t < T$  should still be just as secure had the keys been not compromised. The key exchange in Figure 6 doesn't meet this requirement. If both Alice's and Bob's public/private keys are broken, then the attacker can find out  $K_{A-B}$  and  $K_{B-A}$  for not only the future sessions, but also for past key-exchanges. This is one reason why such a simple key-exchange is not used.

In order to achieve PFS, the best known key exchange protocol is Diffie-Hellman. This is what is used by both IKEv1 and IKEv2. The following sections discuss the details of Diffie-Hellman and its own pitfalls.

### 8.1 Diffie-Hellman (DH) Key Exchange

The first step in DH key exchange protocol is to find a large prime number  $P$ , say of size 2048 bits and select a number  $g < P$ . The numbers  $g$  and  $P$  are published so that everyone can use these numbers. RFC3526 [10] defines a few of these numbers for IKE. When Alice and Bob want to derive a key, they each generate a random number  $r_1$  and  $r_2$  ( $0 < r_1, r_2 < P - 1$ ) and compute the following: (Figure 7(a) shows the details of Diffie-Hellman Key Exchange.)

**Alice:** Compute  $T_A = (g^{r_A}) \text{ mod } P$  and send  $T_A$  to Bob and keep  $r_A$  until a response from Bob is received.

**Bob:** Compute  $T_B = (g^{r_B}) \text{ mod } P$ ; and send  $T_B$  to Alice and keep  $r_B$  until a response from Bob is received.

When Alice and Bob each receive  $T_B$  and  $T_A$  respectively, they compute:

**Alice:**  $K_{A-B} = (T_B^{r_A}) \pmod P$

**Bob:**  $K_{A-B} = (T_A^{r_B}) \pmod P$

As if by magic, the two numbers generated by Alice and Bob are the same. Let's take an example with a small numbers to see this magic in action. Let's say  $g = 2$  and  $P = 13$ . Let's say Alice generates a random number (less than 12)  $r_A = 4$  and Bob does the same with  $r_B = 9$ .

**Alice:** Compute  $T_A = (2^4) \equiv 3$ . Send 3 to Bob and keep  $r_A = 4$ .

**Bob:** Compute  $T_B = (2^9) \equiv 5$ . Send 5 to Alice and keep  $r_B = 9$ .

where  $(\cdot)$  indicates modulo 13 arithmetic.

When Alice and Bob each receive  $T_B$  and  $T_A$  respectively, they compute:

**Alice:**  $K_{A-B} = (5^4) \equiv 1$

**Bob:**  $K_{A-B} = (3^9) \equiv 1$

Okay, so the math works (see [3] for the proof), but why is this secure? The non-technical reason is that given  $g = 2$ ,  $P = 13$ ,  $T_A = 3$  one cannot compute  $r_A$  efficiently. In other words, the most efficient way is to do an exhaustive search for  $r_A$  from 1 all the way to  $P - 1$  and match it with  $T$ . Note that  $r_A$  must be "sufficiently random" for the scheme to be secure. In fact, the security of the key-exchange depends strongly on how unpredictable the random number is.

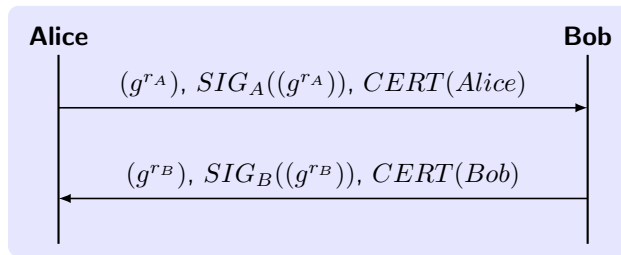
## 8.2 Diffie-Hellman Pitfalls

Let's now design a key exchange protocol based on Diffie-Hellman. Why is a simple Diffie Hellman exchange not enough? Because of the Man-In-The-Middle (MITM) attack (See Figure 7(b)). Let's say Mallory, a rogue/compromised router intercepts a token  $T_A$  from Alice and instead of forwarding it to Bob, generates her own token  $T_M$  and sends it to Bob. When Bob receives  $T_M$ , he cannot tell who sent it. He just sends back his token  $T_B$ , assuming that the token he received came from Alice. Since Mallory is on the end-to-end path, she intercepts  $T_B$  from Bob and generates the key  $K_{M-B}$ . Mallory can now encrypt and decrypt all packets coming from Bob. Furthermore, Mallory can send another token to Alice (to prevent computation she just resents  $T_M$  to Alice) and generates a key  $K_{A-M}$ . Mallory now has a complete control over all the messages that goes between Alice and Bob.

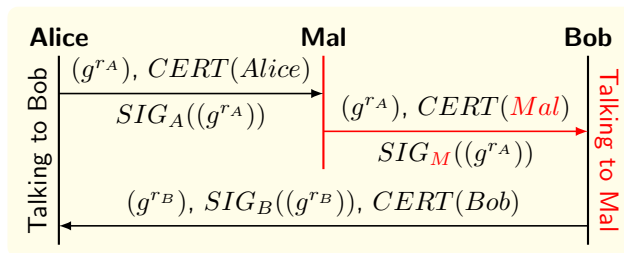
This attack is nothing new. It's just a special case of Figure 1. Every time a key exchange is performed without thoroughly verifying the origin of the message, one risks MITM attack. But this leads to an interesting problem: How can one verify the authenticity of a random source on the Internet? One way to do this is to have a shared secret between Alice and Bob, but this clearly doesn't scale.

Another way is to let Alice and Bob get a "trust token," called a *certificate* from a third party which they both trust. If the trusted third party could digitally vouch that Alice and Bob are who they claims they are, then MITM attack could be prevented. This is what the Public Key Infrastructure (PKI) strives to achieve. A node that wants to use IPsec first generates a private-key and public-key and sends her public-key to a well known trusted third party called the Certification Authority (CA). Once the CA (VeriSign for example), has verified the authenticity of the clients, it binds the identity of the user to her public-key by signing the name and public-key with the CAs private keys. The resulting quadruple of the identity of the user, the identity of certification authority, the public-key of the user, and signature from the certification authority is collectively called a certificate. (Certificates typically include a lot more information than these four essential entities; refer to PKCS 6 [16] for details.)

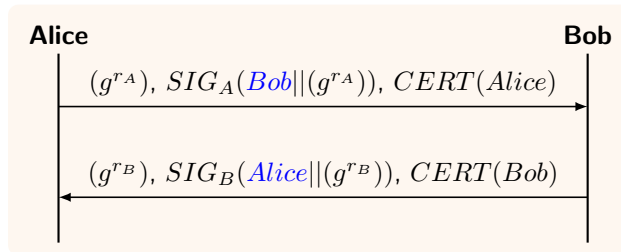
For a CA to verify the authenticity of the user, it needs to validate two things: First, the identity of the person or the host that generated the public-key (this is typically done off line) and second, that the person really owns the private key corresponding to the public-key. To achieve the second goal, the owner of the private-key is required to encrypt a



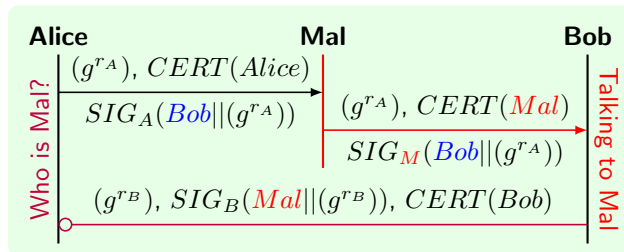
(a) Authenticated Diffie-Hellman Key Exchange.



(b) Identity Misbinding Attack.



(c) Identity Misbinding Attack Defense



(d) Example of ID Misbinding Defense

Figure 8: Authenticated Diffie-Hellman and Identity Misbinding Attack. Numbers within  $(\cdot)$  indicate  $\text{mod } P$  arithmetic.

piece of data using her private key and sends it to the CA, who can verify that the owner really possesses the keys (it's typically more complicated than that).

Why is the second stage needed? Why not just depend on verifying the user identity? The reason second stage is needed is to prevent a rogue node, say Eve, from taking Alice's public-key and getting it certified in her name. If Eve can get Alice's public-key certified in her name, she could easily impersonate Alice. For example, when Alice sends a message signed with her private key, Eve can simply replace Alice's certificate with hers. Since the public-key in the certificate is same as that of Alice's public-key, the signature will remain valid. However, since the identity bound to that certificate is that of Eve, the authenticating host, Bob for example, will believe that the message came from Eve. (Note that this has very severe security implications.) So when Alice says, "deposit 100.00 in my account," Bob will deposit that amount in Eve's name! Unfortunately, this second stage of certification process is not always performed by all Certification Authorities.

Once a host has obtained the certificate, authentication is easy (See Figure 8(a)). If Alice wants to prove that  $T_A$  came from her, all she needs to do is to sign  $T_A$  with her private-key, and send the certificate along with  $T_A$  to Bob (i.e., Alice sends  $\{T_A, SIG(T_A), CERT(Alice)\}$  to Bob, where  $SIG(x)$  is the signature computed on  $x$  and  $CERT(y)$  is the certificate assigned to the user  $Y$ ). Remember that to compute  $SIG(T_A)$ , Alice computes the hash of  $T_A$ , and encrypts it with her private-key. For details about digital signature, refer to PKCS 1 [17] standard.

When Bob receives  $\{T_A, SIG(T_A), CERT(Alice)\}$ , he first gets the Public-Key of the Certification Authority to verify that  $CERT(Alice)$  really came from a CA that Bob can trust. If the certificate is valid, Bob extracts the public-key from the certificate and verifies that  $SIG(T_A)$  is valid. Since only Alice could have generated the  $SIG$  (because only she knows the private key), Bob can trust Alice. Note that it's important for Bob to verify the authenticity of Alice's Certificate. If Mallory can forge Alice's signature, she can do MITM attack once again. For a detailed overview of PKI trust model, refer to [14].

It might seem that Authenticated Diffie-Hellman exchange is secure. Alas! it's not. The problem is that the signature is calculated on  $T_A$ , so anyone, say Mallory, who has a certificate that Bob trusts could take  $T_A$ , sign it with her Private Keys and send  $\{T_A, SIG(T_A), CERT(Mal)\}$  to Bob (please see Figure 8(b)). Bob will authenticate the Certificate and the signature and find it okay since he trusts Mallory's. When Bob responds, Mallory could let the message go through. Alice will now validate Bob's signature and find it okay. But now there is a problem: *Alice thinks she is talking to Bob (which is right), but Bob thinks that he is talking to Mallory (which is not right)*. Again, as in the previous example, when Alice now says: "Deposit 100.00 in my account," Bob will gladly deposit that amount in Mallory's name! So much for all the trouble of getting a certificate. This attack is known as identity misbinding attack.

While Alice and Bob are authenticating each other correctly, the reason identity misbinding attack exists is because the Certification Authority gives out certificates to a lot more entities than just Alice and Bob. Furthermore many of the certified entities are trusted by both Alice and Bob. Therefore, to prevent identity misbinding attack, the two end points not only need to verify the certificate, but also need to indicate to the other side who they think they are verifying.

Based on this analysis, it's not hard to solve the identity misbinding attack. For example, in the simplest case, Bob could just sign the identity of who he thinks he is talking to as shown in Figure 8(c). In this case, when Bob signs the concatenation of who he thinks he is talking to, he will include Mallory's name. When Alice receives this message, she will find out that Bob has been tricked by Mallory, and she could tear-down the security association (please see Figure 8(d)).

While the scheme described above works in theory, it has an inherent problem: Bob needs to know Alice's identity before he can include it in the signature calculation. Therefore, this scheme cannot provide identity protection! To provide identity protection, there needs to be two components in the key exchange. First, the session key should be bound to the identities of the end points without divulging the identities themselves, and second, to verify that the identities are trusted by a third party. To bind the identity of end points to the session key, one can just compute the HMAC of identity with the session key. This way the identity doesn't get exposed. Second, to verify the identities, each side must compute the signature using their certificates. Note that these certificates could themselves be encrypted with the session key and thereby protect the identity that is available in the certificate.

When a node receives the HMAC and the encrypted certificate, it first decrypts the certificate and extracts the user's identity. Then, the node computes the HMAC using the identity present in the certificate (and the session key), and

verifies that the HMAC is valid. Then, the system verifies whether the signature (computed using the user's private key) is authentic. There are several possible variations of this theme. See SIGMA [11] protocol for details. (SIGMA is the mother of all IKE protocols. Serious readers must read it!)

To summarize, to provide a secure key-exchange exchange with identity protection, one needs two different features in the key-exchange:

- A mechanism to bind the session key to the identity of the end points, typically by computing the HMAC of identity under the session key. This step only prevents identity misbinding, but doesn't provide MITM protection.
- A mechanism to prevent MITM attacks by using a trusted third party. Furthermore, in case where certificates are used as the trust token, the certificates must be encrypted by the session key to provide identity protection.

Please note that the goal of the above two items are complementary. Proof of knowledge of session key provides guarantees that there is no identity misbinding. The certificates and the signatures on the other hand are needed to prevent MITM attacks.

## 9 IKEv1

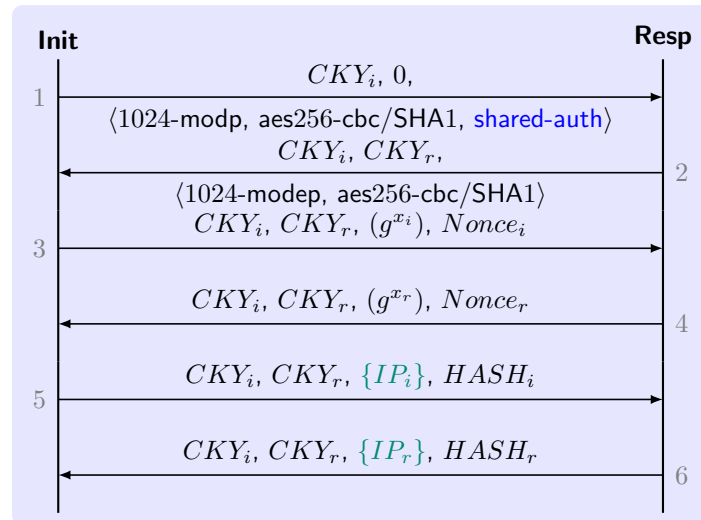
IKEv1 [1] defines a two phase process for key negotiation. The Phase-1 of key exchange is used to create the `ISAKMP_SA`. This security association is used to establish a secure channel over which the Phase-2 — called the `IPSEC_SA`— key-exchange takes place. Phase-1 establishes a bi-directional Security Association which could later be used by any one of the two sides to initiate a Phase-2 exchange. All messages in Phase-2 are protected under the auspices of `ISAKMP_SA`.

Phase-1 itself is split into two different parts: The *Main Mode* and the *Aggressive Mode*. The Main Mode provides Identity Protection while the Aggressive Mode does not.

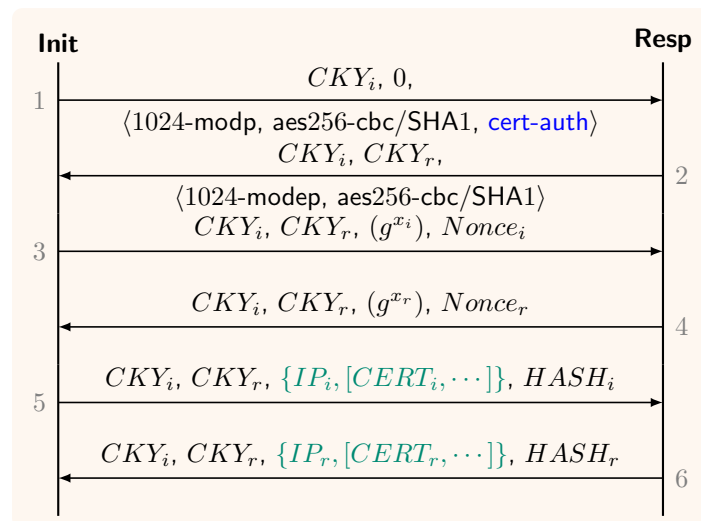
Phase-2 defines just one mode, called *Quick Mode*. The Quick Mode is used to establish keys for the actual IPsec data encryption and authentication. In addition to Quick Mode, Phase-2 can also have *Group Mode* which is used to exchange specific DH groups information. RFC2409 does not describe quick mode as phase-2, although there is no harm in considering it as a Phase-2 exchange. The idea behind Group Mode is to protect the generator and the prime to provide further security guarantees. The assumption is that by protecting  $g$  and  $P$ , the attacker has a much harder task of breaking the Diffie-Hellman that was protected by `ISAKMP_SA`. However, this reasoning is somewhat circular and flawed. If the attacker can break Diffie-Hellman, he can break Phase-1 and find the keys that will give her the  $g$  and  $P$  of the group mode exchange. If the attacker cannot break Diffie-Hellman, then there is no need to protect  $g$  and  $P$ . Therefore, I will not describe quick-mode.

As noted earlier, both Phase-1 and Phase-2 run as applications, typically on Port 500. Because applications can access the IP address and port-numbers of the source (typically returned by the `recvfrom` system call on Unix systems), most IKE messages (which are UDP payload) don't need to carry IP address in the payload. Note that if hosts performing the IKE exchange are also the ones doing the IPsec processing (as is typically the case in transport mode), then IP address as Identity Protection will not work (it will still work in tunnel mode where IKE will be run by gateways on behalf of someone who is behind the gateway). In some cases the hash of IP address is used in the IKE payload either for ID protection or for NAT/Firewall detection.

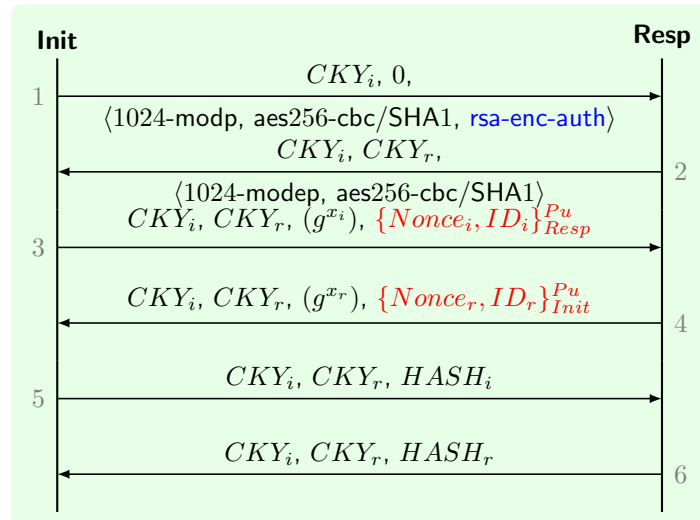
All IKE messages are based on the *ISAKMP* packet format. See RFC2408 [12] for the details. Each *ISAKMP* payload has two fields  $CKY_i$  and  $CKY_r$  which identifies individual security associations. The ordering of  $CKY_i$  and  $CKY_r$  remains the same throughout all the message exchange (*i.e.*, it doesn't get flipped like an IP address). In addition, the *ISAKMP* header includes an exchange type which defines how the messages are ordered. The third most important field is the Flags, which indicates whether the payload following *ISAKMP* header is encrypted, integrity protected, or request for something else. If the encrypted bit is set, the entire payload following the header is encrypted (of course *ISAKMP* header remains un-encrypted.)



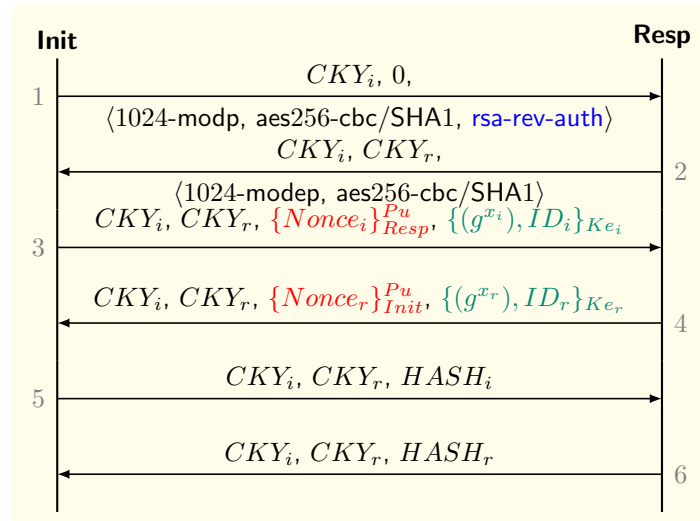
(a) Authentication using Pre-shared Key (Passwords).



(b) Authentication using Digital Signature (Certificates).



(c) Authentication using Public Key Encryption.



(d) Authentication using Revised Public Key Encryption.

Figure 9: Main Mode Key Exchange with Different Ways of Host Authentication. Encrypted messages are combined together within  $\{ \cdot \}$ . Text in **PineGreen** is encrypted on the wire using a Symmetric Key (if no key is specified in the subscript it means the keys were derived from  $SKEYID_e$ ). Text in **Red** is encrypted using Asymmetric Keys — The super script  $Pu$  or  $Pr$  indicates whether the encryption is based on Public-key or Private-key; the subscript indicates whose keys are being used. Text in **Blue** is used to high-light important messages. Numbers within  $( \cdot )$  indicate  $\text{mod } P$  arithmetic.

## 9.1 Main Mode

The Main mode is designed for Identity Protection and is based on SIGMA [11]. It allows four different ways of host authentication as shown in Figure 9. Regardless of the authentication mechanism, there are three pairs of message exchange. For each of these messages, it's the responsibility of the Initiator to ensure reliable communication. Therefore, all retransmissions are done by the Initiator.

The first message contains the client's "Anti Clogging Token"  $CKY_i$ , commonly known as a "cookie," and 0 for the responder's cookie. Presence of 0 in the responder's cookie field indicates the start of a new key-exchange. Until the Responder receives  $CKY_r$  in messages 3, the responder doesn't create any state to protect against memory or computation exhaustion attack (since denial of service attacks have been studied extensively in the recent years, I will not elaborate more on this). In addition to the cookies, message 1 and 2 in all these key-exchange contain a list of Security Association proposal (this is shown within  $\langle \cdot \rangle$  in Figure 9). The SA proposal includes information about the Diffie-Hellman Group, the Encryption and the Authentication Algorithm to be used during key-exchange, and the method used to authenticate the end-points (*i.e.*, whether to authenticate based on passwords, certificates, or public-keys). Note that typically, an Initiator will advertise a list of proposals from which the Responder can choose.

The 3<sup>rd</sup> and 4<sup>th</sup> messages include the Diffie-Hellman token ( $g^{x_i}$ ) and ( $g^{x_r}$ ), which is calculated based on the Diffie-Hellman Group selected during messages 1 and 2. In addition, the messages 3 and 4 also include a random number called *nonce*, which is used to generate entropy for deriving the keys. By the end of message 4, both sides could derive the session key based on the Diffie-Hellman exchange.

The session key, called  $SKEYID$  is derived based on the authentication method. Based on the session key, a list of new keys are calculated as described below:

$$KEY_d = \text{prf}(SKEYID, (g^{x_y}) \| CKY_i \| CKY_r \| 0)$$

where  $\text{prf}(\cdot)$  is a pseudo random function, which is typically HMAC and  $\|$  represents concatenation of strings encoded in 8-bit bytes.  $KEY_d$  (called  $SKEYID_d$  in RFC2409) is an intermediary key that is used to generate keys for encryption and authentication. The authentication key  $KEY_a$  (called  $SKEYID_a$  in RFC2409) is define as

$$KEY_a = \text{prf}(SKEYID, KEY_d \| (g^{x_y}) \| CKY_i \| CKY_r \| 1)$$

and the encryption key  $KEY_e$  (called  $SKEYID_e$ ) is derived as follows:

$$KEY_e = \text{prf}(SKEYID, KEY_a \| (g^{x_y}) \| CKY_i \| CKY_r \| 2)$$

A natural question to ask is why are the keys derived in such a strange way. The main reason is that to generate good keys, one needs to have enough entropy in the input stream.

Finally, IKEv1 defines four different ways to authenticate hosts. Following is the description of these authentication methods:

**Authentication Using Pre-shared keys (passwords):** Authentication using passwords is the most widely used method. In order to protect against password guessing attacks, passwords are never used alone to derive  $SKEYID$  (*i.e.*, unlike on your computer where your password and salt remains the same for authentication, IKE uses  $Nonce_i$  and  $Nonce_r$  to derive a session key). The  $SKEYID$  itself is calculated as:

$$SKEYID = \text{prf}(\text{password}, Nonce_i \| Nonce_r)$$

Identity Protection in case of password based authentication have a small caveat. Since  $HASH_i$  and  $HASH_r$  are dependent upon  $SKEYID$ , which is derived from *password*, the two end points must know the *password* before they can verify  $HASH_i$  and  $HASH_r$ . However, the  $HASH$  values must be verified before the encrypted part of the payload that contains  $ID_i$  and  $ID_r$  is processed. Therefore, the only way to find *passwords* is by using IP address as the Identity. Although this is not a problem in tunnel-mode, it completely defeats the purpose in transport mode!

**Authentication Using Certificates:** When authentication is based on certificates, the *SKEYID* is calculated as:

$$SKEYID = \text{prf}(Nonce_i || Nonce_r, (g^{xy}))$$

When using certificates, it's important to verify the validity of the certificates. Verifying a certificate means two things: First, the Certification Authority's signature is valid and second, verifying that the Certificate has not been revoked. Experience gained from TLS shows that while many web browsers verify the signature of the Certification Authority, they do not check for Certificate Revocation List — partly because there are no certification revocation lists easily available.

**Authentication Using Public Keys:** Authentication using Public-Key Encryption assumes that the public-keys of the two sides are stored on the two IKE participants through some out-of-band mechanism. For example, if Alice wants to talk to Bob, then Alice will put her public-key on Bob's machine and Bob will do the same for Alice. Unlike a digital signature, which requires a trusted third party, Public-Key authentication requires mutual trust — which is convenient when the number of users in the system is small (for example, in a lab).

The first method of Authentication using Public Key Encryption, encrypts the Identity and Nonce in messages 3 and 4. Since  $ID_i$  and  $ID_r$  are encrypted using the public-key of the each side, even an active attacker cannot find out the Identity of the two end points. If the two end points have multiple Public Keys then the *HASH* of the public key used for encryption is used exchange in clear in messages 3 and 4 (for example, if each user has her own public key as in case of *ssh*, then including this *HASH* helps find out right key to be used. Note, however, that there is no exchange for *HASH* algorithm to be used, so if a receiver receives a random *HASH*, it has to do the guess work to find out which private key belongs to the given public key. Typically, this is not a problem, if the public keys stored on the end points also included the fingerprints of the keys.)

**Authentication Using Revised Public Keys:** One problem with the previous method of using public keys is that both Identity and Nonce need to go through the exponentiation process to generate the encrypted payload. This could require a lot of computation power on the end-points. To alleviate this, the revised version of public key based authentication only encrypts the nonce using public-key, but derives a symmetric key to encrypt  $ID_i$ ,  $ID_r$ ,  $g^{x_i}$  and  $g^{x_r}$ . The symmetric keys  $Ke_i$  and  $Ke_r$  are derived as:

$$Ke_i = \text{prf}(Nonce_i, CKY_i)$$

$$Ke_r = \text{prf}(Nonce_r, CKY_r)$$

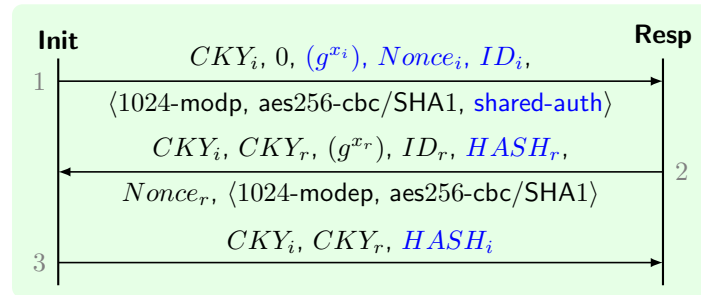
Note that since  $Ke_i$  and  $Ke_r$  are derived from the nonces, which is transmitted on the wire, such a design is dangerous. This, however, is not a problem as breaking the public key will only divulge the users identity: The keying material for the session is still derived from  $g^{xy}$ , which is just as safe as before. Such design practices should however be discouraged.

Finally, the 5<sup>th</sup> and 6<sup>th</sup> messages use the authentication payload which is shown as  $HASH_i$  and  $HASH_r$ . In case of authentication using certificates, the the hashes are further converted into signatures. The hashes are computed as shown below:

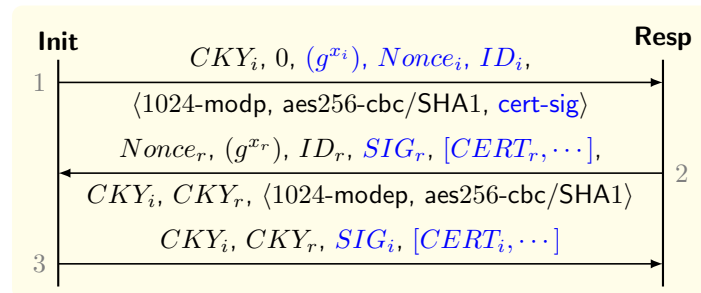
$$HASH_i = \text{prf}(SKEYID, g^{x_i} || g^{x_r} || CKY_i || CKY_r || SA_i || ID_i)$$

$$HASH_r = \text{prf}(SKEYID, g^{x_r} || g^{x_i} || CKY_r || CKY_i || SA_i || ID_r)$$

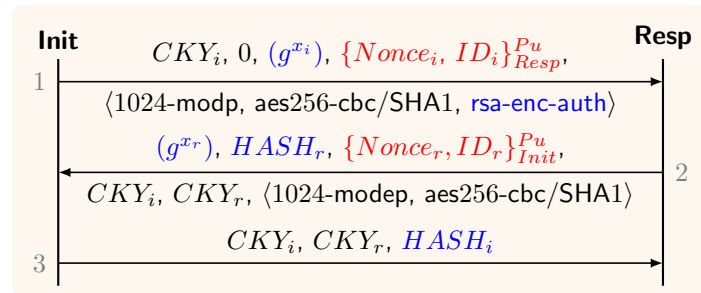
Note that in addition to  $ID_i$ ,  $ID_r$  and the Diffie-Hellman tokens,  $HASH_i$  and  $HASH_r$  also includes the security association payload (shown as  $SA_i$ ) that was negotiated in message 1 and 2. The main reason for including Security Association payload is to prevent denial of service attacks where an active attacker degrades the strength of encryption/authentication algorithm.



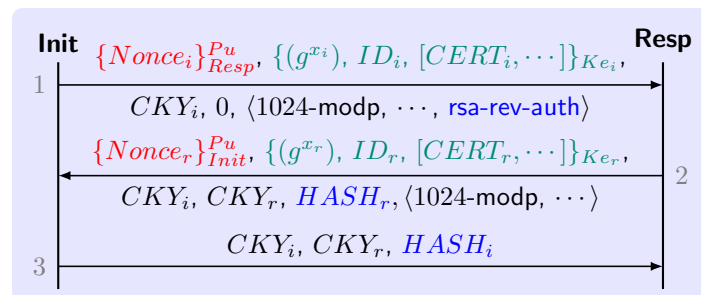
(a) Authentication using Preshared Key (Passwords).



(b) Authentication using Digital Signature (Certificates).



(c) Authentication using Public Key Encryption.



(d) Authentication using Revised Public Key Encryption.

Figure 10: Aggressive Mode Key Exchange. Encrypted messages are combined together within  $\{\cdot\}$ . Text in PineGreen is encrypted on the wire using a Symmetric Key (if no key is specified in the subscript it means the keys were derived from  $SKEYID_e$ ). Text in Red is encrypted using asymmetric keys. The super script  $Pu$  or  $Pr$  indicates whether the encryption is based on Public-key or Private-key; the subscript indicates whose keys are being used. Text in Blue is used to high-light important messages. Numbers within  $(\cdot)$  indicate mod  $P$  arithmetic.

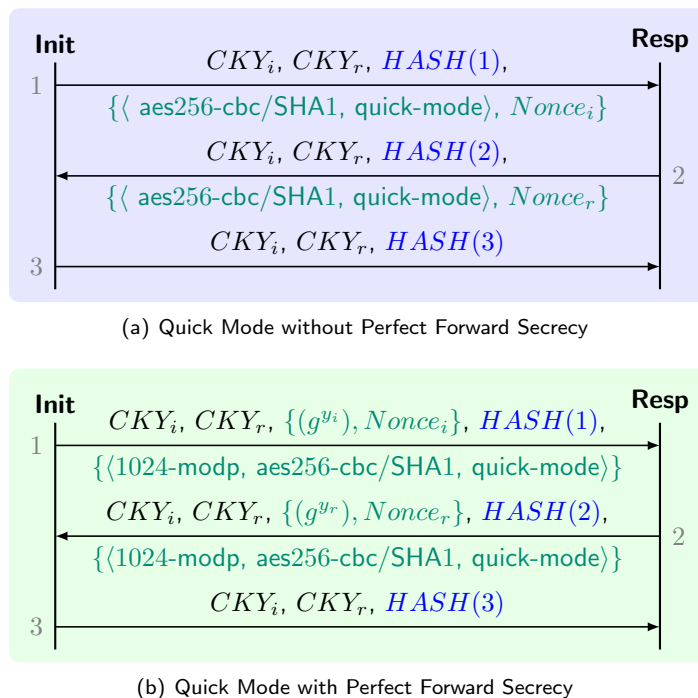


Figure 11: Quick Mode Key Exchange in IKEv1. (See previous Figures on the use of color.)

## 9.2 Aggressive Mode

The aggressive mode of key-exchange is designed to reduce the number of round trips needed to establish `ISAKMP_SA`. As in case of main mode, the aggressive mode key exchange follows the SIGMA protocol, and has four modes of authentication. However, unlike the main mode, the aggressive mode does not have identity protection in case of authentication using certificates and pre-shared keys.

See Figure 10 about the details of the message exchange. The keying material for aggressive mode is derived exactly as in case of main mode.

## 9.3 Quick Mode

Quick Mode is not a complete exchange in itself because it is bound to phase 1 exchange, but it is used as part of the SA negotiation process to derive keying material and negotiate shared policy for non-ISAKMP SAs [1]. All message exchange during Quick Mode is encrypted and integrity protected under the auspices of `ISAKMP_SA` (except of course  $CKY_i$  and  $CKY_r$  since they identify the `ISAKMP_SA`). There are two different key exchanges defined for Quick Mode. The first exchange, shown in Figure 11(a) does not provide perfect forward secrecy, while the one shown in Figure 11(b) does. Recall that PFS means that even if the long-term keys (e.g., the certificates) of the two principals are compromised, the message exchange that was performed before the key-compromise happened cannot still be broken. Regardless of which scheme is used, the end result of the key exchange is a set of new session keys that can be used by IPsec to encrypt and integrity protect different messages. See RFC2409 for details.

## 9.4 IKEv1 Problems

One obvious and hard to escape problem with IKEv1 is the complexity of the protocol. First, there is no need to have a quick mode and a main mode: Most transactions on the Internet are ephemeral; it's hard to find a connection that will last so long that users perform two quick-mode exchange within the same main-mode transaction. Second,

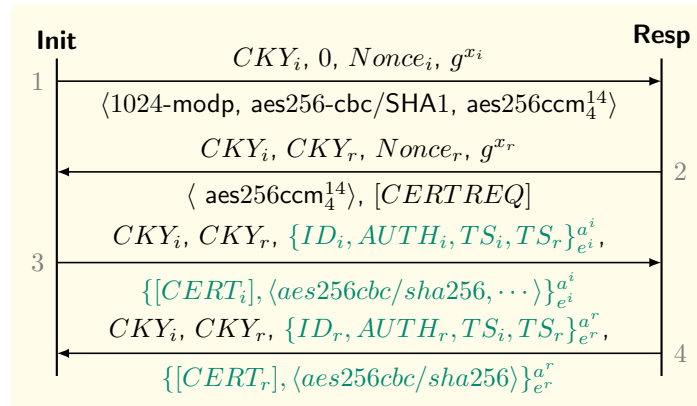


Figure 12: IKEv2 Key Exchange. Elements within  $\{\cdot\}_e^a$  are encrypted using key  $e$  and integrity protected using key  $a$ . Presence of  $[\cdot]$  implies optional elements. Elements within  $\langle \cdot \rangle$  represent a set of Security Association algorithm that the initiator and responder agree to before using authentication or encryption. The  $AUTH$  payload authenticates messages 1 and 2 and are defined based on the authentication algorithm used. (See previous figures on the use of color.)

each authentication method in IKEv1 is essentially a different protocol. This means a considerable overhead for software implementation (remember the goal of specifying these protocols is to ensure that someone can implement them in software; the goal is not to just write an RFC). Finally, the protocol is poorly specified. For example, it's hard to find out whether the  $HASH_i$  and  $HASH_r$  tokens in Figure 9 are encrypted or not (I myself don't know the answer!). There are several other occurrences where one can interpret the protocol in two different ways. This is clearly a inter-operability nightmare and could also lead to unforeseen security vulnerabilities. For example, if a naive implementation might not understand the implications of order of encryption and authentication and could implement protocol which might be insecure (in the presence of certain encryption algorithm). For more details on the limitations of IKEv1 see the excellent paper by Radia Perlman and Charlie Kaufman [15].

## 10 IKEv2

As described in Section 8, both IKEv1 and IKEv2 [2] are based on the SIGMA protocol. So while the underlying cryptographic philosophy (*i.e.*, use HMAC for identity misbinding prevention and Signature for MITM protection) of IKEv2 is similar to that of IKEv1 IKEv2 exchange looks much simpler than IKEv1. In order to simplify the number of key-exchange protocols that IKEv1 has, IKEv2 takes a more abstract approach. All messages in IKEv2 perform a single well known security operation. As long as the underlying primitive of each step is secure, the protocol is guaranteed to be secure. This is a welcome change from IKEv1 since we don't need to analyze eight different key exchange protocols and *their interaction with each other*. Note that it's not just enough to analyze the protocol as it is defined, because attackers could try to mix one kind of key exchange with another kind and try to fool the system.

Figure 12 describes IKEv2 key exchange protocol. The first two messages in IKEv2 are known as `IKE_SA_INIT`. During the `IKE_SA_INIT` exchange, the Initiator and Responder exchange their Diffie-Hellman Tokens, a pair of Nonce (in addition to the Cookies), and the security association algorithms to be used *during messages 3 and 4 exchange*. Note that unlike IKEv1, the initiator starts directly with the Diffie-Hellman token, and expects the responder to process it. This has two security implications: First, the Initiator must somehow guess the Diffie-Hellman group (*i.e.*, the modulus  $g$  and the prime  $P$ ) of the responder and second, the responder must somehow protect itself if it's under denial of service attack. To address these two issues, IKEv2 allows the Responder to ignore the first message and request the Initiator to send back message 1 again, but the response must include the non-zero cookie,  $CKY_r$ , that the responder sent during message 2. Furthermore, in message 2, the Responder may request a different Diffie-

Hellman group (as a part of security association algorithm negotiation). This step expands the number of key-exchange messages from four to six. This is an excellent design because under normal conditions, hosts are not under denial of service attack and they typically use well known Diffie Hellman groups. Wasting a round trip on trying to protect from these “corner” cases would have been suboptimal. Note, however, that such a design might have some unintended consequence. For example, in order to minimize the round trip time during key exchange, all hosts might strive to use just one single “most popular” Diffie-Hellman group to maximize their chance of having an additional round trip. If this “popular group” group turns out to be vulnerable to attacks, it could lead to very poor security on a majority of hosts. As usual, diversity is an excellent survival tool, and it applies even to these cases.

At the end of message 2, both the Initiator and Responder will have enough information to derive the Diffie-Hellman key ( $g^{x_i \cdot x_r}$ ). Therefore, messages 3 and 4 (called `IKE_AUTH`) are always encrypted and integrity protected<sup>3</sup>. The keys used on both the sides are always different in each direction, *i.e.*, the Initiator generates a pair of keys  $\{e_i, a_i\}$ , where  $e_i$  is used for encryption, and  $a_i$  is used for authentication. Similarly, the Responder generates a pair of keys  $\{e_r, a_r\} \neq \{e_i, a_i\}$ . Note that both the parties need all the four keys to decrypt and verify the message from the other side.

Just like `IKE_SA_INIT`, `IKE_AUTH` is very well designed: Each message in `IKE_AUTH` is integrity protected. This is equivalent to computing the HMAC of the Identity as described in SIGMA [11] protocol (note that the identity  $ID_i$  and  $ID_r$  are encrypted and the entire message is integrity protected). Therefore, the protocol exchange as it is safe against key-misbinding attack. However, it’s not yet safe against the MITM attack.

The goal of `AUTH` payload in message 3 is used to prevent the MITM attack. The `AUTH` payload is calculated by performing the authentication operation on messages 1 and 2. Recall, that in order to prevent MITM attack, both the sides must include  $g^{x_i}$  and  $g^{x_r}$  while calculating the auth payload, otherwise, an attacker could change either of these token and cause Denial of service attack. Furthermore, in order to prevent an attacker from changing the strength of the encryption/authentication algorithm, the entire payload in message 1 and 2 is used to compute the `AUTH` payload.

The exact method used to compute the `AUTH` payload is based on the authentication algorithm to be used. For example, if the authentication algorithm is digital certificates, then a signature over message 1 and 2 is used. Note that by separating the `AUTH` payload from the default key exchange algorithm, it’s possible for each side to authenticate the other side using entirely different methods. For example, the Responder could authenticate itself using a digital certificate, while the Initiator could use pre-shared keys (passwords). This is similar to what we typically do when we buy products on-line using TLS. First, our browser authenticates the web server by it’s digital certificate, and then we authenticate ourselves to the web site by using the password we used to register with the web site.

Unlike IKEv1, IKEv2 allows each side to agree on one set of keys that could be used by ESP/AH. Therefore, the `IKE_AUTH` payloads contain the security association algorithms to be used for IPsec processing. In addition, the message also contains traffic selector which could allow a remote host to get an IP address that the security gateway would accept for IPsec processing. Without the traffic selector, it’s often hard for a remote client to know what IP address and port numbers it should use the remote VPN network. For example, consider the case where each host in an enterprise network gets an IP address through DHCP. When a client connects through the VPN gateway, it will not have enough information as to what IP address it should use for the tunnel end-point. For the details, about traffic selector, see [2].

In addition to the main key exchange, IKEv2 also provides a way to create additional security associations for IPsec processing. These exchanges are known as `CHILD_SA` and are similar to the `IKE_AUTH` exchange. Refer to [2] for details.

---

<sup>3</sup>Since  $CKY_i$  and  $CKY_r$  are used to identify the security association, they are not encrypted although they are used for integrity protection.

## References

- [1] D. Harkins and D. Carrel, “The Internet Key Exchange (IKE),” RFC 2409 (Proposed Standard), Nov. 1998, updated by RFC 4109. [Online]. Available: <http://www.ietf.org/rfc/rfc2409.txt>
- [2] C. Kaufman, “Internet Key Exchange (IKEv2) Protocol,” Sep. 2004, URL might change without notice. [Online]. Available: <http://www.ietf.org/internet-drafts/draft-ietf-ipsec-ikev2-17.txt>
- [3] C. Kaufman, R. Perlman, and M. Speciner, *Network Security: Private Communication in a Public World*, 2nd ed. Prentice Hall PTR, Apr. 2002.
- [4] S. Kent, “IP Authentication Header,” Internet draft; work in progress, Mar. 2005, expires: Sep. 2005. [Online]. Available: <http://www.ietf.org/internet-drafts/draft-ietf-ipsec-rfc2402bis-11.txt>
- [5] —, “IP Encapsulating Security Payload (ESP),” Internet draft; work in progress, Mar. 2005, expires: Sep. 2005. [Online]. Available: <http://www.ietf.org/internet-drafts/draft-ietf-ipsec-esp-v3-10.txt>
- [6] S. Kent and R. Atkinson, “IP Authentication Header,” RFC 2402 (Proposed Standard), Nov. 1998. [Online]. Available: <http://www.ietf.org/rfc/rfc2402.txt>
- [7] —, “IP Encapsulating Security Payload (ESP),” RFC 2406 (Proposed Standard), Nov. 1998. [Online]. Available: <http://www.ietf.org/rfc/rfc2406.txt>
- [8] —, “Security Architecture for the Internet Protocol,” RFC 2401 (Proposed Standard), Nov. 1998, updated by RFC 3168. [Online]. Available: <http://www.ietf.org/rfc/rfc2401.txt>
- [9] S. Kent and K. Seo, “Security Architecture for the Internet Protocol,” Mar. 2005, URL might change without notice. [Online]. Available: <http://www.ietf.org/internet-drafts/draft-ietf-ipsec-rfc2401bis-06.txt>
- [10] T. Kivinen and M. Kojo, “More Modular Exponential (MODP) Diffie-Hellman groups for Internet Key Exchange (IKE),” RFC 3526 (Proposed Standard), May 2003. [Online]. Available: <http://www.ietf.org/rfc/rfc3526.txt>
- [11] H. Krawczyk, “SIGMA: The ‘SIGn and MAC’ approach to authenticated Diffie-Hellman and its use in IKE protocols,” in *Crypto ’03: Advances in Cryptology*, vol. 2729, 2003, pp. 400–425. [Online]. Available: <http://www.ee.technion.ac.il/~hugo/sigma.ps>
- [12] D. Maughan, M. Schertler, M. Schneider, and J. Turner, “Internet security association and key management protocol (ISAKMP),” Internet Engineering Task Force, RFC 2408, Nov. 1998. [Online]. Available: <ftp://ftp.isi.edu/in-notes/rfc2408.txt>
- [13] J. Mirkovic and P. Reiher, “A taxonomy of DDoS attack and DDoS defense mechanisms,” *SIGCOMM Computer Communication Review*, vol. 34, no. 2, pp. 39–53, 2004.
- [14] R. Perlman, “An overview of PKI trust models,” in *IEEE Network*, vol. 13, no. 6, Nov. 1999, pp. 38–43.
- [15] R. Perlman and C. Kaufman, “Key Exchange in IPsec: Analysis of IKE,” *IEEE Internet Computing*, vol. 4, no. 6, pp. 50–56, 2000.
- [16] RSA Laboratories Technical Note, “PKCS 6: Extended-Certificate Syntax Standard,” Nov. 1993. [Online]. Available: <ftp://ftp.rsasecurity.com/pub/pkcs/ps/pkcs-6.ps>
- [17] —, “PKCS 1 v2.1: RSA Cryptography Standard,” Jun. 2002. [Online]. Available: <ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-1/pkcs-1v2-1.pdf>