A Cryptographic Evaluation of IPsec

Niels Ferguson* and Bruce Schneier**

Counterpane Internet Security, Inc.,
3031 Tisch Way, Suite 100PE, San Jose, CA 95128
http://www.counterpane.com

1 Introduction

In February 1999, we performed an evaluation of IPsec based on the November 1998 RFCs for IPsec [KA98c, KA98a, MG98a, MG98b, MD98, KA98b, Pip98, MSST98, HC98, GK98, TDG98, PA98]. Our evaluation focused primarily on the cryptographic properties of IPsec. We concentrated less on the integration aspects of IPsec, as neither of us is intimately familiar with typical IP implementations,

IPsec was a great disappointment to us. Given the quality of the people that worked on it and the time that was spent on it, we expected a much better result. We are not alone in this opinion; from various discussions with the people involved, we learned that virtually nobody is satisfied with the process or the result. The development of IPsec seems to have been burdened by the committee process that it was forced to use, and it shows in the results.

Even with all the serious critisisms that we have on IPsec, it is probably the best IP security protocol available at the moment. We have looked at other, functionally similar, protocols in the past (including PPTP [SM98, SM99]) in much the same manner as we have looked at IPsec. None of these protocols come anywhere near their target, but the others manage to miss the mark by a wider margin than IPsec. This difference is less significant from a security point of view; there are no points for getting security nearly right. From a marketing point of view, this is important. IPsec is the current "best practice," no matter how badly that reflects on our ability to create a good security standard.

Our main criticism of IPsec is its complexity. IPsec contains too many options and too much flexibility; there are often several ways of doing the same or similar things. This is a typical committee effect. Committees are notorious for adding features, options, and additional flexibility to satisfy various factions within the committee. As we all know, this additional complexity and bloat is seriously detrimental to a normal (functional) standard. However, it has a devastating effect on a security standard.

It is instructive to compare this to the approach taken by NIST for the development of AES [NIST97a, NIST97b]. Instead of a committee, NIST organized a contest. Several small groups each created their own proposal, and the process is

* niels@counterpane.com
** schneier@counterpane.com

limited to picking one of them. At the time of writing there has been one stage of elimination, and any one of the five remaining candidates will make a much better standard than any committee could ever have made.[1]

Lesson 1 *Cryptographic protocols should not be developed by a committee.*

We stress that our analysis is in no way a full security review. We have only looked at the most obvious aspects of the system. There are many areas that have hardly been reviewed at all, and we have not included all of our comments in this review. Fixing the problems that we list will improve IPsec, but will not result in a secure system.

2 General Comments

Before we go into the details of IPsec we give some general comments on the entire system.

2.1 Complexity

We start by introducing a new rule of thumb, which we call the "Complexity Trap":

The Complexity Trap: Security's worst enemy is complexity.

This might seem an odd statement, especially in the light of the many simple systems that exhibit critical security failures. It is true nonetheless. Simple failures are simple to avoid, and often simple to fix. The problem in these cases is not a lack of knowledge of how to do it right, but a refusal (or inability) to apply this knowledge. Complexity, however, is a different beast; we do not really know how to handle it. Complex systems exhibit more failures as well as more complex failures. These failures are harder to fix because the systems are more complex, and before you know it the system has become unmanageable.

Designing any software system is always a matter of weighing and reconciling different requirements: functionality, efficiency, political acceptability, security, backward compatibility, deadlines, flexibility, ease of use, and many more. The unspoken requirement is often simplicity. If the system gets too complex, it becomes too difficult and too expensive to make and maintain. Because fulfilling

[1] Imagine the AES process in committee form. RC6 is the most elegant cipher, so we start with that. It already uses multiplications and data-dependent rotations. We add four decorrelation modules from DFC to get provable security, add an outer mixing layer (like MARS) made from Rijndael-like rounds, add key-dependent S-boxes (Twofish), increase the number of rounds to 32 (Serpent), and include one of the CAST-256 S-boxes somewhere. We then make a large number of arbitrary changes until everybody is equally unhappy. The end result is a hideous cipher that combines clever ideas from many of the original proposals and which most likely contains serious weaknesses because nobody has taken the trouble to really analyze the final result.

more of the other requirements usually involves a more complex design, many systems end up with a design that is as complex as the designers and implementers can reasonably handle. (Other systems end up with a design that is too complex to handle, and the project fails accordingly.)

Virtually all software is developed using a try-and-fix methodology. Small pieces are implemented, tested, fixed, and tested again. Several of these small pieces are combined into a larger module, and this module is tested, fixed, and tested again. The end result is software that more or less functions as expected, although we are all familiar with the high frequency of functional failures of software systems.

This process of making fairly complex systems and implementing them with a try-and-fix methodology has a devastating effect on security. The central reason is that you cannot easily test for security; security is not a functional aspect of the system. Therefore, security bugs are not detected and fixed during the development process in the same way that functional bugs are. Suppose a reasonable-sized program is developed without any testing at all during development and quality control. We feel confident in stating that the result will be a completely useless program; most likely it will not perform any of the desired functions correctly. Yet this is exactly what we get from the try-and-fix methodology with respect to security .

The only reasonable way to "test" the security of a system is to perform security reviews on it. A security review is a manual process; it is very expensive in terms of time and effort. And just as functional testing cannot prove the absence of bugs, a security review cannot show that the product is in fact secure. The more complex the system is, the harder a security evaluation becomes. A more complex system will have more security-related errors in the specification, design, and implementation. We claim that the number of errors and difficulty of the evaluation are not linear functions of the complexity, but in fact grow much faster.

For the sake of simplicity, let us assume the system has n different options, each with two possible choices. (We use n as the measure of the complexity; this seems reasonable, as the length of the system specification and the implementation are proportional to n.) Then there are $n(n-1)/2 = O(n^2)$ different pairs of options that could interact in unexpected ways, and 2^n different configurations altogether. Each possible interaction can lead to a security weakness, and the number of possible complex interactions that involve several options is huge. We therefore expect that the number of actual security weaknesses grows very rapidly with increasing complexity.

The increased number of possible interactions creates more work during the security evaluation. For a system with a moderate number of options, checking all the two-option interactions becomes a huge amount of work. Checking every possible configuration is effectively impossible. Thus the difficulty of performing security evaluations also grows very rapidly with increasing complexity. The combination of additional (potential) weaknesses and a more difficult security analysis unavoidably results in insecure systems.

In actual systems, the situation is not quite so bad; there are often options that are "orthogonal" in that they have no relation or interaction with each other. This occurs, for example, if the options are on different layers in the communication system, and the layers are separated by a well-defined interface that does not "show" the options on either side. For this very reason, such a separation of a system into relatively independent modules with clearly defined interfaces is a hallmark of good design. Good modularization can dramatically reduce the effective complexity of a system without the need to eliminate important features. Options within a single module can of course still have interactions that need to be analyzed, so the number of options per module should be minimized. Modularization works well when used properly, but most actual systems still include cross-dependencies where options in different modules do affect each other.

A more complex system loses on all fronts. It contains more weaknesses to start with, it is much harder to analyze, and it is much harder to implement without introducing security-critical errors in the implementation.

This increase in the number of security weaknesses interacts destructively with the weakest-link property of security: the security of the overall system is limited by the security of its weakest link. Any single weakness can destroy the security of the entire system.

Complexity not only makes it virtually impossible to create a secure system, it also makes the system extremely hard to manage. The people running the actual system typically do not have a thorough understanding of the system and the security issues involved. Configuration options should therefore be kept to a minimum, and the options should provide a very simple model to the user. Complex combinations of options are very likely to be configured erroneously, which results in a loss of security. The stories in [Kah67, Wri87, And93b] illustrate how management of complex systems is often the weakest link.

We therefore repeat: security's worst enemy is complexity. Security systems should be cut to the bone and made as simple as possible. There is no substitute for simplicity.

Complexity of IPsec In our opinion, IPsec is too complex to be secure. The design obviously tries to support many different situations with different options. We feel very strongly that the resulting system is well beyond the level of complexity that can be analysed or properly implemented with current methodologies. Thus, no IPsec system will achieve the goal of providing a high level of security.

2.2 The IPsec Documentation

The first thing one notices when looking at IPsec is that the documentation is very hard to understand. There is no overview or introduction, the reader has to assemble all the pieces and build an overview himself. This is a highly unsatisfactorily state of affairs; after all, the documentation is meant to convey information to the readers. We do not believe it is reasonable to expect anyone to learn IPsec from the IPsec documentation.

Various parts of the IPsec documentation are very hard to read. For example, the ISAKMP specifications [MSST98] contain numerous errors, essential explanations are missing, and the document contradicts itself in various places.

Stated Goal The documentation does not specify what IPsec is supposed to achieve. Without stated goals there is no standard to analyze against. Although some of the goals are more or less obvious, we were often left to deduce the goals of the designers from the design itself, and then check whether the design fulfilled these goals. This is of course highly undesirable, and any discussion of the results is likely to deteriorate into a discussion of what IPsec is supposed to achieve.

This lack of specification also makes it very difficult to use IPsec. A designer who wishes to use IPsec as part of a system has no way of determining what functionality the IPsec system provides. Saying that IPsec provides packet-level security is like saying that a car will transport you from A to B. Both are true given the right circumstances, but in both situations there are a large number of additional prerequisites and restrictions. A designer that tries to use IPsec without knowing all the prerequisites and the resulting functionality is virtually certain to create a system that does not achieve its security goals.

This problem is already evident. IPsec provides IP-level security, and is thus essentially a VPN protocol. Yet we hear about people trying to use it for application-level security, such as authenticating the user when she tries to get her email. IPsec authenticates packets as originating from someone who knows a particular key, yet many seem to think it authenticates the originating IP address as that is what they can filter on in their firewall. Such misuses of IPsec can only lead to problems.

Rationale None of the IPsec documentation provides any rationale for any of the choices that were made. Although this is somewhat less important than a clear statement of the goals, we nevertheless consider it crucial information. If a reviewer is to comment on the design (and RFCs are, after all, Requests For Comments), he should be told what each option was intended to achieve. Without any rationale, the reviewer is left to guess at it, and then review the design based on the guessed-at rationale.

Lesson 2 *The documentation of a system should include introductory material, an overview for first-time readers, stated goals, rationale, etc.*

If the reason for not including such extra material is the desire to ensure that there are no contradictions, then the usual standardization practice of marking some part of the documentation as normative and other parts as informative can be used.

3 Bulk Data Handling

We will now discuss the various parts of IPsec in more detail. For this, we assume that the reader is familiar with the details of IPsec.

The core of IPsec consists of the functions that provide authentication and confidentiality services to IP packets [KA98c, KA98a, MG98a, MG98b, MD98, KA98b, PA98, GK98, TDG98]. These can, for example, be used to create a VPN over an untrusted network, or to secure packets between any pair of computers on a local network.

3.1 Complexity

IPsec has two modes of operation: transport mode and tunnel mode. There are two protocols: AH and ESP. AH provides authentication, ESP provides authentication, encryption, or both. This creates a lot of extra complexity: two machines that wish to authenticate a packet can use a total of four different modes: transport/AH, tunnel/AH, transport/ESP with NULL encryption, and tunnel/ESP with NULL encryption. The differences between these options, both in functionality and performance, are minor. The documentation also makes it clear that under some circumstances it is envisioned to use two protocols: AH for the authentication and ESP for the encryption.

Modes As far as we can determine, the functionality of tunnel mode is a superset of the functionality of transport mode. (From a network point of view, one can view tunnel mode as a special case of transport mode, but from a security point of view this is not the case.) The only advantage that we can see to transport mode is that it results in a somewhat smaller bandwidth overhead. However, the tunnel mode could be extended in a straightforward way with a specialized header-compression scheme that we will explain shortly. This would achieve virtually the same performance as transport mode without introducing an entirely new mode. We therefore recommend that transport mode be eliminated.

Recommendation 1 *Eliminate transport mode.*

Without any documented rationale, we do not know why IPsec has two modes. In our opinion it would require a very compelling argument to introduce a second major mode of operation. The extra cost of a second mode (in terms of added complexity and resulting loss of security) is huge, and it certainly should not be introduced without clearly documented reasons.

Eliminating transport mode also eliminates the need to separate the machines on the network into the two categories of hosts and security gateways. The main distinction seems to be that security gateways may not use transport mode; without transport mode the distinction is no longer necessary.

Protocols The functionality provided by the two protocols overlaps somewhat. AH provides authentication of the payload and the packet header, while ESP provides authentication and confidentiality of the payload.

In transport mode, AH provides a stronger authentication than ESP can provide, as it also authenticates the IP header fields. One of the standard modes of operation would seem to be to use both AH and ESP in transport mode. In tunnel mode, ESP provides the same level of authentication (as the payload includes the original IP header), and AH is typically not combined with ESP [KA98c, section 4.5]. (Implementations are not required to support nested tunnels that would allow ESP and AH to both be used in tunnel mode.)

One can question why the IP header fields are being authenticated at all. The authentication of the payload proves that it came from someone who knows the proper authentication key. That by itself should provide adequate information. The IP header fields are only used to get the data to the recipient, and should not affect the interpretation of the packet. There might be a very good reason why the IP header fields need to be authenticated, but until somebody provides that reason the rationale remains unclear to us.

The AH protocol [KA98a] authenticates the IP headers of the lower layers. This is a clear violation of the modularization of the protocol stack. It creates all kind of problems, as some header fields change in transit. As a result, the AH protocol needs to be aware of all data formats used at lower layers so that these mutable fields can be avoided. This is a very ugly construction, and one that will create more problems when future extensions to the IP protocol are made that create new fields that the AH protocol is not aware of. Also, as some header fields are not authenticated, the receiving application still cannot rely on the entire packet. To fully understand the authentication provided by AH, an application needs to take into account the same complex IP header parsing rules that AH uses. The complex definition of the functionality that AH provides can easily lead to security-relevant errors.

The tunnel/ESP authentication avoids this problem, but uses more bandwidth. The extra bandwidth requirement can be reduced by a simple specialized compression scheme: for some suitably chosen set of IP header fields X, a single bit in the ESP header indicates whether the X fields in the inner IP header are identical to the corresponding fields in the outer header.[2] The fields in question are then removed to reduce the payload size. This compression should be applied after computing the authentication but before any encryption. The authentication is thus still computed on the entire original packet. The receiver reconstitutes the original packet using the outer header fields, and verifies the authentication. A suitable choice of the set of header fields X allows tunnel/ESP to achieve virtually the same low message expansion as transport/AH.

We conclude that eliminating transport mode allows the elimination of the AH protocol as well, without loss of functionality. We therefore recommend that the AH protocol be eliminated.

[2] A trivial generalization is to have several flag bits, each controlling a set of IP header fields.

Recommendation 2 *Eliminate the AH protocol.*

ESP The ESP protocol allows the payload to be encrypted without being authenticated. In virtually all cases, encryption without authentication is not useful. The only situation in which it makes sense not to use authentication in the ESP protocol is when the authentication is provided by a subsequent application of the AH protocol (as is done in transport mode because ESP authentication in transport mode is not strong enough). Without the transport mode or AH protocol to worry about, ESP should always provide its own authentication. We recommend that ESP authentication always be used, and only encryption be made optional.

Recommendation 3 *Modify ESP to always provide authentication; only encryption should be optional.*

This also avoids misconfigurations by system administrators. There will be many administrators who know that they need "encryption" to make their network secure, but will not know exactly what cryptographic services they require. They will be quite likely to configure ESP for only encryption, believing that it provides security.

Fragmentation IPsec exhibits some unfortunate interactions with the fragmentation system of the IP protocol [KA98c, appendix B]. This is a complex area, but a typical IPsec implementation has to perform specialized processing to facilitate the proper behavior of higher-level protocols in relation to fragmentation. Strictly speaking, fragmentation is part of the communication layer below the IPsec layer, and in an ideal world it would be transparent to IPsec. Compatibility requirements with existing protocols (such as TCP) force IPsec to explicitly handle fragmentation issues, which adds significantly to the overall complexity.[3] As far as we can see, there does not seem to be an elegant solution to this problem.

Conclusion The bulk data handling is overly complex. The number of major modes of operation can be drastically reduced without significant loss of functionality. We recommend that only ESP/tunnel mode be retained, and that the ESP protocol be modified to always provide authentication.

3.2 Order of Operations

When both encryption and authentication are provided, IPsec performs the encryption first, and authenticates the ciphertext. In our opinion, this is the wrong

[3] The real problem is of course that TCP interacts directly with the fragmentation which is on a different layer, but it is a bit late to change that.

order. Going by the "Horton principle" [WS96], the protocol should authenticate what was meant, not what was said. The "meaning" of the ciphertext still depends on the decryption key used. Authentication should thus be applied to the plaintext (as it is in SSL [FKK96]), and not to the ciphertext.

Encrypting first and authenticating second does not always lead to a direct security problem. In the case of the ESP protocol, the encryption key and authentication key are part of a single ESP key in the SA. A successful authentication shows that the packet was sent by someone who knew the authentication key. The recipient trusts the sender to encrypt that packet with the other half of the ESP key, so that the decrypted data is in fact the same as the original data that was sent. The exact argument why this is secure gets to be very complicated, and requires special assumptions about the key agreement protocol. For example, suppose an attacker can manipulate the key agreement protocol used to set up the SA in such a way that the two parties get an agreement on the authentication key but a disagreement on the encryption key. When this is done, the data transmitted will be authenticated successfully, but decryption takes place with a different key than encryption, and all the plaintext data is still garbled. All in all it is much simpler and more robust to authenticate the plaintext.

An Attack on IPsec Suppose two hosts have a manually keyed transport-mode AH-protocol Security Association (SA), which we will call $\mathrm{SA}_{\mathrm{AH}}$. Due to the manual keying, the AH protocol does not provide any replay protection. These two hosts now negotiate a transport-mode encryption-only ESP SA (which we will call $\mathrm{SA}_{\mathrm{ESP1}}$) and use this to send information using both $\mathrm{SA}_{\mathrm{ESP1}}$ and $\mathrm{SA}_{\mathrm{AH}}$. The application can expect to get confidentiality and authentication on this channel, but no replay protection. When the immediate interaction is finished, $\mathrm{SA}_{\mathrm{ESP1}}$ is deleted. A few hours later, the two hosts again negotiate a transport-mode encryption-only ESP SA ($\mathrm{SA}_{\mathrm{ESP2}}$), and the receiver chooses the same SPI value for $\mathrm{SA}_{\mathrm{ESP2}}$ as was used for $\mathrm{SA}_{\mathrm{ESP1}}$. Again, data is transmitted using both $\mathrm{SA}_{\mathrm{ESP2}}$ and $\mathrm{SA}_{\mathrm{AH}}$. The attacker now introduces one of the packets from the first exchange. This packet was encrypted using $\mathrm{SA}_{\mathrm{ESP1}}$ and authenticated using $\mathrm{SA}_{\mathrm{AH}}$. The receiver checks the authentication and finds it valid. (As replay protection is not enabled, the sequence number field is ignored.) The receiver then proceeds to decrypt the packet using $\mathrm{SA}_{\mathrm{ESP2}}$, which presumably has a different decryption key than $\mathrm{SA}_{\mathrm{ESP1}}$. The end result is that the receiver accepts the packet as valid, decrypts it with the wrong key, and presents the garbled data to the application. Clearly, the authentication property has been violated.

This may seem like a somewhat contrived example, but it points to a serious flaw in the concept of authentication. The above-mentioned problem can be solved easily (our recommendations remove the transport mode and the AH protocol and thereby the entire scenario). However, this does not fix the underlying problem. Authenticating the ciphertext, or even the plaintext, is not enough; to authenticate the meaning of the packet, the authentication has to cover every bit of data that is used in the interpretation of the packet. See [WS96] for a more detailed discussion of this principle.

Lesson 3 *Authenticate not just the message, but everything that is used to determine the meaning of the message.*

Fast Discarding of Fake Packets Authenticating the ciphertext allows the recipient to discard packets with erroneous authentication more quickly, without the overhead of the decryption. This helps the computer cope with denial-of-service attacks in which a large number of fake packets eat up a lot of CPU time. We question whether this would be the preferred mode of attack against a TCP/IP-enabled computer.

If this is an important consideration then the authentication of the ciphertext can be retained, but only if the decryption key (and any other data relevant to the interpretation) is also included in the authentication. This would be possible to do within the ESP protocol, but it becomes very ugly if the AH protocol has to dig into the ESP protocol data structures to authenticate the decryption key too.

Recommendation 4 *Modify the ESP protocol to ensure that it authenticates all data used in the deciphering of the payload.*

From a modularization point of view it is not a good idea to rely on the key negotiation protocols to provide a strong linkage between the authentication key and the encryption key. Thus the ESP protocol should take care of its own security considerations.

Conclusion The ordering of encryption and authentication in IPsec is dangerous. Authentication must be applied to all data that is used to determine the meaning of a packet. Authenticating the ciphertext without also authenticating the decryption key to be used is wrong.

3.3 Security Associations

A Security Association (SA) is a simplex "connection' that affords security services to the traffic carried by it [KA98c, section 4]. The two computers on either side of the SA store the mode, protocol, algorithms, and keys used in the SA. Each SA is used only in one direction; for bi-directional communications two SAs are required. Each SA implements a single mode and protocol; if two protocols (such as AH and ESP) are to be applied to a single packet, two SAs are required.

Most of our aforementioned comments also affect the SA system; the use of two modes and two protocols make the SA system more complex than necessary.

There are very few (if any) situations in which a computer sends an IP packet to a host, but no reply is ever sent. There are also very few situations in which the traffic in one direction needs to be secured, but the traffic in the other direction does not need to be secured. It therefore seems that in virtually all practical situations, SAs occur in pairs to allow bi-directional secured communications. In fact, the IKE protocol negotiates SAs in pairs.

This would suggest that it is more logical to make an SA a bi-directional "connection" between two machines. This would halve the number of SAs in the overall system. It would also avoid asymmetric security configurations, which, although somewhat useful in real-time traffic scenarios, we think are undesirable.

3.4 Security Policies

The security policies are stored in the SPD (Security Policy Database). For every packet that is to be sent out, the SPD is checked to find how the packet is to be processed. The SPD can specify three actions: discard the packet, let the packet bypass IPsec processing, or apply IPsec processing. In the last case, the SPD also specifies which SAs should be used (if suitable SAs have already been set up) or specifies with what parameters new SAs should be set up to be used.

The SPD seems to be a very flexible control mechanism that allows a very fine-grained control over the security processing of each packet. Packets are classified according to a large number of selectors, and the SPD can match some or all selectors to determine the appropriate action. Depending on the SPD, this can result in either all traffic between two computers being carried on a single SA, or a separate SA being used for each application, or even each TCP connection. Such a very fine granularity has disadvantages. There is a significantly increased overhead in setting up the required SAs, and more traffic analysis information is made available to the attacker. We do not see any need for such a fine-grained control. The SPD should specify whether a packet should be discarded, bypass IPsec processing, be authenticated, or be authenticated and encrypted. Whether several packets are combined on the same SA is not important. The same holds for the exact choice of cryptographic algorithm: any good algorithm will do.

Asking users and administrators to determine which packets are to be secured and which are to bypass IPsec processing is probably already asking too much. A significant fraction of all installations will have settings that create security holes. Asking them to specify the details of the cryptographic algorithms to be used and the parameters that these algorithms take almost guarantees that the average installation will contain configuration weaknesses. We feel that the very most we can ask of the users and administrators is to determine which packets require what type of processing. Ideally a user should specify, for example, that a packet requires authentication and encryption, and leave it at that. The implementation should ensure that *all* algorithms that it uses provide adequate security for all situations. Good cryptographic algorithms are available and cheap; there is no reason to use weak or bad algorithms.[4]

[4] One could argue that the configuration should allow for two levels of encryption: "export-quality" and "full-quality" encryption. This will not work in practice. The quality level of encryption required for the SMTP port depends on the actual contents of the e-mail, and cannot be determined by the data available to the policy matcher.

It would be nice if the same high-level approach could be done in relation to the choice of SA end-points. As there currently does not seem to be a reliable automatic method of detecting IPsec-enabled security gateways, we do not see a practical alternative to manual configuration of these parameters. It is questionable whether automatic detection of IPsec-enabled gateways is possible at all. Without some initial knowledge of the other side, any detection and negotiation algorithm can be subverted by an active attacker.

3.5 Other Comments

This section contains some of our minor comments on the bulk data handling of IPsec.

1. [KA98c, section 5.2.1, point 3] suggests that an implementation can find the matching SPD entry for a packet using back-pointers from the SAD entries. In general this will not work correctly. Suppose the SPD contains two rules: the first one outlaws all packets to port X, and the second one allows all incoming packets that have been authenticated. An SA is set up for this second rule. The sender now sends a packet on this SA addressed to port X. This packet should be refused as it matches the first SPD rule. However, the backpointer from the SA points to the second rule in the SPD, which allows the packet. This shows that back-pointers from the SA do not always point to the appropriate rule, and that this is not a proper method of finding the relevant SPD entry.
2. The handling of ICMP messages as described in [KA98c, section6] is unclear to us. It states that an ICMP message generated by a router must not be forwarded over a transport-mode SA, but transport-mode SAs can only occur in hosts. By definition, hosts do not forward packets, and a router never has access to a transport-mode SA.
   The text further suggests that unauthenticated ICMP messages should be disregarded. This creates problems. Let us envision two machines that are geographically far apart and have a tunnel-mode SA set up. There are probably a dozen routers between these two machines that forward the packets. None of these routers knows about the existence of the SA. Any ICMP messages relating to the packets that are sent will be unauthenticated and unencrypted. Simply discarding these ICMP messages results in a loss of IP functionality. This problem is mentioned, but the text claims this is due to the routers not implementing IPsec. Even if the routers implement IPsec, they still cannot send authenticated ICMP messages concerning the tunnel unless they themselves set up an SA with the tunnel end-point for the purpose of sending the ICMP packet. The tunnel end-point in turn wants to be sure the source is a real router. This requires a generic public-key infrastructure, which does not exist.
   As far as we understand this problem, this is a fundamental compatibility problem with the existing IP protocol that does not have a good solution.

3. Various algorithm specifications require the implementation to reject known weak keys. For example, the DES-CBC encryption algorithm specifications [MD98] requires that DES weak keys be rejected. It is questionable whether this actually increases security. It might very well be that the extra code that this requires creates more security problems due to bugs than are solved by rejecting weak keys.
   Weak keys are not really a problem in most situations. For DES, the easiest way for the attacker to detect the weak keys is to try them all. This is equivalent to a partial keyspace search, and can be done on any small set of keys. Weak-key rejection is only required for algorithms where detecting the weak key class by the weak cipher properties is significantly less work than trying all the weak keys in question (e.g. [WFS99]).

   Recommendation 5 *Remove the weak-key elimination requirement. Encryption algorithms that have large classes of weak keys that introduce security weaknesses should simply not be used.*

4. The only mandatory encryption algorithm in ESP is DES-CBC. Due to the very limited key length of DES [BDR+96], this cannot be considered to be very secure. We strongly urge that this algorithm not be standardized but be replaced by a stronger alternative. The most obvious candidate is triple-DES. Blowfish [Sch94] could be used as an interim high-speed solution in software. The upcoming AES standard [NIST97a, NIST97b] will presumably gain quick acceptance and probably become the default encryption method for most systems.
5. The insistence on randomly selected IV values in [MD98] seems to be overkill. It is true that a counter would provide known low Hamming-weight input differentials to the block cipher, but all reasonable block ciphers are secure enough against this type of attack. Furthermore, chosen-plaintext attacks also allow low-weight input differentials, and are quite likely in an IPsec system. The use of a random generator results in an increased risk of an implementation error that will lead to low-entropy or constant IV values; such an error does reduce security and would typically not be found during testing.
6. Use of a block cipher with a 64-bit block size should in general be limited to at most 2^{32} block encryptions per key. This is due to the birthday paradox. After 2^{32} blocks we can expect one collision.[5] In CBC mode, two equal ciphertexts give the attacker the XOR of two blocks of the plaintext. The specifications for the DES-CBC encryption algorithm [MD98] should mention this, and require that any SA using such an algorithm limit the total amount of data encrypted by a single key to a suitable value.
7. The preferred mode for using a block cipher in ESP seems to be CBC mode [PA98]. This is probably the most widely used cipher mode, but it has some disadvantages. As mentioned earlier, a collision gives direct information about the relation of two plaintext blocks. Furthermore, in hardware

[5] To get a 10^{-6} probability of a collision, it should be limited to about 2^{22} blocks.

implementations, each of the encryptions has to be done in turn. This gives a limited parallelism, which hinders high-speed hardware implementations.

Although not used very often, the counter mode seems to be preferable. The ciphertext of block i is formed as $C_i = P_i \oplus E_K(i)$, where i is the block number that needs to be sent at the start of the packet.[6] After more than 2^{32} blocks, counter mode also reveals some information about the plaintext, but this is less than what occurs in CBC. The big advantage of counter mode is that hardware implementations can parallelize the encryption and decryption process, thus achieving a much higher throughput and lower latency.

8. [PA98, section 2.3] states that Blowfish has weak keys, but that the likelihood of generating one is very small. We disagree with these statements. The likelihood of getting two equal 32-bit values in any one 256-entry S-box is about $\binom{256}{2} \cdot 2^{-32} \approx 2^{-17}$. This is an event that will certainly occur in practice. However, the Blowfish weak keys only lead to detectable weaknesses in reduced-round versions of the cipher. There are no known weak keys for the full Blowfish cipher.
9. In [PA98, section 2.5], it is suggested to negotiate the number of rounds of a cipher. We consider this to be a very bad idea. The number of rounds is integral to the cipher specifications and should not be changed at will. Even for ciphers that are specified with a variable number of rounds, the determination of the number of rounds should not be left up to the individual system administrators. The IPsec standard should specify the number of rounds for those ciphers.

   Negotiating the number of rounds opens up other avenues of attack. Suppose the negotiation protocol can be subverted so that one side used r rounds and the other side $r+1$ rounds. An attacker can now try to get one side to encrypt a message, and the other side to decrypt a message. If the resulting plaintext is returned, the attacker now has a block after 1 round of processing, which for many ciphers is easy to attack.
10. [PA98, section 2.5] proposes the use of RC5 [Riv95]. We urge caution in the use of this cipher. It uses some new ideas that have not been fully analyzed or understood by the cryptographic community. The original RC5 as proposed (with 12 rounds) was broken [BK98], and in response to that the recommended number of rounds was increased to 16. We feel that further research into the use of data-dependent rotations is required before RC5 is used in fielded systems.

3.6 Conclusion

The IPsec bulk data handling can, and should, be simplified significantly. We have identified various other problems and weaknesses that should be addressed.

[6] If replay protection is always in use, then the starting i-value could be formed as 2^{32} times the sequence number. This saves eight bytes per packet.

4 ISAKMP

ISAKMP [MSST98] defines the overall structure of the protocols used to establish SAs and to perform other key management functions. ISAKMP supports several Domains Of Interpretation (DOI), of which the IPsec-DOI is one. ISAKMP does not specify the whole protocol, but provides building blocks for the various DOIs and key-exchange protocols.

4.1 Comments on the Documentation

The description of ISAKMP is not very clear. We had great difficulty understanding the intentions of the much of the description. We list a few of our comments on the documentation.

1. The document often contains repeated instances of the same text, or very similar texts. This makes the text unnecessarily long, and hard to read. For example, in section 5.2 the list of checking actions and possible logging and notifications could be described much more succinctly.
2. [MSST98, section 3.11] defines the Hash Payload, which is used to verify the integrity and authentication of the data in the ISAKMP message. The data is described as being the result of a hash function applied to the message; this provides no integrity or authentication. Presumably this payload contains a MAC, in which case the name "Hash Payload" (as well as the "Hash Data" field name) is a misnomer. In general, IPsec documentation seems to confuse hashes and MACs.
3. [MSST98, section 5.1] specifies the generic processing of sending an ISAKMP message. The description method used to specify the algorithm is ambiguous, and probably even wrong. It is not clear when exactly step 4 is entered. Strictly speaking, in step 2 the message is resent and the counter is decremented. Let us suppose it reaches 0. Step 3 is now executed immediately, which logs the RETRY LIMIT REACHED event in the log before the other side has had a chance to respond to the last resend. Assuming step 4 is executed immediately after step 3, the last resend of the packet will never be answered, and is in vain. This is clearly not the intention of the designers. The description of the resend mechanism should be clarified.
   Presumably the resend mechanism is to be disabled as soon as a valid reply is received.
4. Data attributes are only present in a Transform payload. The IPsec DOI defines several data attributes that apply to the various levels. Some attributes apply to the entire SA. An SA can contain several proposals, and a proposal can contain several transforms. Should the SA-wide attributes be in the first transform, in each of the transforms, or elsewhere? How should the various placings of attributes be interpreted?
5. It is interesting to note that [MSST98, section 1] claims that splitting the functionality into the ISAKMP, DOI, and key exchange protocol makes the

security analysis more complex. We feel that a modularization along well-chosen boundaries could simplify the analysis significantly. Unfortunately, the ISAKMP, DOI and key exchange protocol documents still contain numerous cross-dependencies, which indeed do make the security analysis more complex.

4.2 Header Types

The definition of the ISAKMP headers gives rise to various questions. The standard header for each payload contains the type of the *next* payload in the message [MSST98, section 3.2]. This construction seems to complicate both the generation and parsing of messages. As the ISAKMP header already contains the overall length, the parser already knows whether there is another payload in the message. It would seem simpler to have each payload contain its own type in the standardized payload-header, instead of the type of the next payload.

[MSST98, section 3.4] does not make it clear that the Security Association Payload uses a system of sub-payloads, which confused us for a while. Another oddity is the fact that there is no field in the SA header to indicate the type of the first sub-payload. Each sub-payload specifies the type of the *next* sub-payload, but there is no corresponding field for the very first sub-payload. Presumably the receiver has to know that an SA payload will always have a proposal payload as the first sub-payload. This gives rise to the question why the proposal payload has a "next header" field at all. Using the same type of information (allowed sub-payload structures), the receiver can already deduce what the value of this field should be. There are also other inconsistencies. The SA header does not specify how many proposals are included as sub-payloads, but the proposal header does specify how many transform payloads are included as sub-payloads.

The system of sub-payloads should be documented more clearly. We would suggest that the "next header" field be changed into a "this header" field. That would certainly make a lot of things simpler.

The confusion over the sub-payloads also leads to a second confusion. A Certificate Request payload may not be inserted between two Proposal payloads–proposal payloads are just subpayloads inside an SA header–although this is not made explicit. If a Certificate Request payload was inserted before the first proposal, the recipient would have no way of identifying that payload as a Certificate Request payload, and would interpret it as a Proposal. The fact that the Certificate Request Payload section specifies that it must be accepted at any point during the exchange only adds to the confusion.

4.3 Security Comments

We found several cases in which the security of the constructions is questionable. As the ISAKMP is a very general structure document, it is often unclear whether this will result in weaknesses in actual specific cases in which ISAKMP is used. Again we stress that we did not perform a full analysis of all the ISAKMP protocols; we expect that there are further undiscovered weaknesses.

1. ISAKMP allows great latitude to the participants in the exact messages that are sent and the exact processing that is done. This makes it extremely hard to give any reasonable statement about the security properties achieved by the protocols. The authors do not feel they have a good overview of all the different variations that are allowed by ISAKMP. This makes a good security analysis extremely hard to do.
2. [MSST98, section 5.1] specifies the resend functionality. What is not clear is how the receiver responds to multiple copies of the same message. Ideally, the implementation should verify that all copies it receives are identical, and resend identical copies of the first reply that it sent.

   A less careful implementation might re-process the incoming message. This opens up various modes of attack. For example, in the identity protection exchange an attacker could resend the fourth message to the initiator, and collect authentications on many chosen messages by varying the nonce. (We are ignoring the encryption for the moment; note that the encryption might be weak or nonexistent.) Even more threatening, by changing the KE payload the attacker might be able to use subtle interactions to find information about the key.

   An attack along the following lines might be possible. An attacker eavesdrops on an identity protection exchange, and then resends the fourth message with modified KE payloads. Let us assume that a DH key agreement protocol modulo p is used with a subgroup of size $q|(p-1)$, and that $p-1$ has several other small divisors. (Even though this specific choice of DH group might seem unfortunate, it could be used in a practical and secure key exchange algorithm.) The attacker can now recover the DH secret of the initiator by repeatedly sending KE payloads that contain elements of a low order and checking which of the limited set of possible keys is being used in the encryption. This reveals the initiator's DH secret modulo the order of the element sent in the KE payload. Combining several of these tries reveals the entire DH secret. Once the DH secret of the initiator is known, the attacker resends the original fourth message. This returns both parties to the same state they had before the attacker interfered. Any subsequent exchange continues as normal, but the attacker has the key, which clearly violates the intention of the protocol.

   This shows that it is important to specify how each party should react to a repeat of a message in an exchange. We have not analyzed other protocols for similar weaknesses, but it seems clear that any solution that reprocesses a protocol step can introduce additional weaknesses. Therefore, the retransmission system and handling of duplicate messages should avoid reprocessing messages.

   Rejecting copies of a message that are not identical to the first copy received does not allow for recovery from a bogus first message. There is little security risk in not providing for this recovery. The exchange will have to be abandoned and a new one started. In general, there is no method of ensuring the completion of an exchange in the presence of an active attacker. It is always possible for the attacker to flip some bits in each message, in which case any exchange will fail.

3. [MSST98, section 4.6] defines the authentication-only exchange that provides authentication of the other party but does not exchange keys. There is no direct use for this protocol. At the end of the exchange both parties know who they were talking to, but by itself that does not provide much information. We assume this protocol is meant to be used to authenticate the other party when a secure channel has already been established without authentication. This does not work, as the other side can perform a man-in-the-middle attack against the authentication-only protocol. This is in fact a general property of the four exchanges defined in ISAKMP. All of them can be completed by a man in the middle. However, the man in the middle does not learn the key that is established by the exchange. As the authentication-only protocol does not establish a key, and does not link the authentication to an existing key, none of the participants in the protocol is much wiser at the end.
   What makes this protocol dangerous is that some implementers might think it actually provides some kind of security service. Any attempted use of the authentication-only exchange will most likely result in a security weakness of some sort.
4. The Identity Protection Exchange of [MSST98, section 4.5] is vulnerable to an active attack. The Initiator sends its identity before having authenticated the other party. An active attacker can pretend to be the responder, and collect the Initiator's identity before abandoning the protocol. The Initiator has little choice but to try again to negotiate the SA. If the attacker does not interfere this will complete, but the attacker has retrieved the IDii value.
   Note that this type of weakness can be avoided by the use of public-key encryption. The initiator can send his own identity encrypted with the public key of the responder. For this to work, the initiator has to retrieve the responder's public key in some way. A central public-key directory can provide this service, without revealing more information to an attacker than that the initiator is retrieving somebody's key.

4.4 Functional Comments

As part of our analysis we also generated various comments on the functionality of ISAKMP. Here is a choice selection:

1. The Certificate Request Payload is used to request a certificate from the other party. It allows the requestor to specify a single CA that it will accept. The responder must send "its certificate based on the value contained in the payload." In many practical systems, a single responder might have several certificates from the same CA. It is unclear whether the responder should send only one suitable certificate, or all suitable certificates.
   One common case is not handled at all. There does not seem to be a way for one of the parties to ask for a certificate issued by any one of a list of CAs. This could be very useful. There are currently many competing CAs

for X.509v3 certificates. A specific machine might very well trust a subset of them. In that case it would accept a certificate signed by any one of a list of CAs. In ISAKMP, sending a list of CAs is interpreted as asking for all certificates from these SAs (see [MSST98, section 3.10]).

2. [MSST98, section 4.5] defines the Identity Protection Exchange. The nonces in this protocol could also be sent in the first two messages, making this protocol more like the base exchange. Furthermore, the exchange can be shortened by two messages.

   The order of the third and fourth message is not important. If the order of these two messages is swapped, then each of them can be merged with an adjacent message and sent in a single packet. This would save a full round-trip time in the negotiations without loss of functionality.

3. [MSST98, section 4.7] defines the aggressive exchange. It states that the initial SA payload can only contain one proposal and one transform. This does not seem to be a necessary restriction. First of all, two proposal payloads with the same proposal number constitute a single proposal. If the concern is that the Initiator must know the final transforms to be used while sending the first message, then the requirement that only one proposal number be used is enough.

   As the KE payload is not dependent on the way in which the final keys are actually being used, we do not see a reason why the single-proposal restriction is in place.

   The restriction to a single transform seems redundant. We see no problems with a single proposal that contains multiple transforms.

4. [MSST98, section 5.5] forces the SPI to be generated pseudorandomly. This is a needless restriction on the sender, and should be removed. Some implementations might wish to use the SPI directly as an index into a table, in which case it cannot be generated pseudorandomly. [KA98c, appendix A] states that the SPI has only local significance, and that the recipient may choose various ways to interpret the SPI. This would seem to imply that the SPI does not have to be chosen randomly.

5. Various features, such as the cookies, are included to prevent denial-of-service attacks against the participants. In ISAKMP, they do not even do that. As stated in [MSST98, section 1.7.1], denial-of-service attacks cannot be prevented completely. We are not convinced that it is a good idea to increase the complexity of the protocol to provide a partial solution to this problem. This would require an analysis of the complexity of the various forms of IP-related denial-of-service attacks against computers. Partial countermeasures only make sense if they make attacks harder to perform. Protecting against difficult attacks makes no sense if there is no protection against the easy forms of attack. This remains an area for further study.

5 IKE

IKE is the default key-exchange protocol for ISAKMP and, at the moment, the only one [HC98]. IKE is built on top of ISAKMP and performs the actual establishment of both ISAKMP SAs and IPsec SAs.

5.1 Primitive Functions

IKE supports the negotiation of various primitive functions for use in the protocols. Among these are the hash function and the pseudorandom function (PRF) to be used. No requirements are given for the hash function and PRF. There is no mention of the properties that IKE assumes the hash function and PRF have.

Hash Function The most widely used definition of a hash function states that it is collision-resistant. Thus, it should be impossible to find two messages m_1 and m_2 such that $H(m_1) = H(m_2)$ where H is the hash function. In [And93a], Ross Anderson showed that this is often not enough, and that many protocols break for certain collision-resistant hash functions. He gave one simple example. Let m' be the first n bits of the message m, and let m'' be the rest. Define $H'(m'|m'') := m'|H(m'')$ for any collision-resistant hash function H. It is clear that H' is as collision-resistant as H is. However, many protocols silently assume that the hash function is more or less a random mapping, and can fail when this is not the case [And93a]. For example, the HMAC construction fails miserably with H' as hash function, as it reveals the key in the output. This shows that it is important to define which properties are required from the hash function. Applying HMAC to just any (collision-resistant) hash function is dangerous and should be avoided.

Pseudorandom Function Currently no special PRFs are defined; the default mode of using the negotiated hash function in the HMAC construction is used instead. Following the notational convention of [HC98], we will call the first argument of the PRF the key and the second argument the message. There are no requirements given for a new PRF; the description calls for a "keyed pseudorandom function."

We take this to mean the following: for a fixed key, the mapping of messages to the output is a pseudorandom function. The PRF should be indistinguishable from a function where the key uniformly selects a random element of the set of all functions mapping messages into suitable output values.

Note that this definition does not preclude a PRF that is invertible for a known key. We define a PRF called NPRF (for Nasty PRF) as follows: for a short message the PRF is defined as

$$\mathrm{prf}(k, m) = E(k, 0|m)$$

while for long messages it is defined as

$$\text{prf}(k, m) = E(k, 1|H(m))$$

Here, H is a suitable hash function, E is a block cipher encryption taking a key and a plaintext block as arguments, and a message is considered "long" if it is at least as long as the block size of the block cipher. For short messages, this PRF is invertible if the key is known. If the block cipher is good and the block size is large enough (e.g., 1024 bits), this function is indistinguishable from a random function, as detecting the fact that it is a permutation and not a random function requires too much work.

IKE uses the PRF sometimes with known key input. It is clearly not the designer's intention to have an invertible function in this case. To avoid this type of problems, IKE should specify what properties it expects from its PRF.

The default construction of the PRF is to use HMAC and the hash function. As shown above, some hash functions result in very bad PRFs when used with the HMAC construction. This can break any protocol using the PRF. For example, the main mode exchange with public-key encryption fails when using HMAC with H' defined above as the HASH_I reveals SKEYID, allowing a fake responder to impersonate the real responder.

Lesson 4 *The properties required of each of the primitive functions used in the system should be clearly documented.*

This can be seen as specifying the interface between the two parts of the definition. It allows each of the parts to be analyzed separately. Without such clear specifications, it is much harder to analyze the security of the system. Furthermore, there is a serious danger that some future extension will introduce a new primitive which lacks one of the properties that was silently assumed by the designers and evaluators of the original version.

5.2 Derivation of SKEYID Data

Observe the computation of SKEYID [HC98, page 10] in the three different authentication cases. For signature authentication, the key input of the PRF is public, and the message input is secret. This is clearly an abuse of the PRF. (If the NPRF is used, revealing SKEYID also reveals g^{xy}.) Furthermore, the concatenation of the two nonces can be up to 512 bytes long, while HMAC-SHA1 takes at most 64 bytes of key. The public-key encryption case on the next line resolves this by inserting an extra hash function application.

The three formulas for computing SKEYID seem somewhat ad hoc. It is not clear what properties are wanted, and which are achieved. We are unsure why the cookies used in the public-key encryption case, and not in the pre-shared key case.

One idea seems to be to use every value at most once as input. This principle is not followed in the derivation of the SKEYID_d, SKEYID_a, and SKEYID_e, where the same values are repeatedly used in the derivation. In the signature authentication case, g^{xy} affects both the key and the message field of the PRF application used to compute SKEYID_d.

As far as we are able to see, these properties do not lead to a direct weakness, as PRFs are very forgiving (in the ideal case). The signature authentication case is saved by the fact that SKEYID itself is never revealed, but only used as key input to other PRF applications.

Recommendation 6 *Fix the derivation formulas for SKEYID and associated values.*

5.3 The HASH Authentication Values

IKE uses so-called HASH values for the authentication of both parties. These are in fact Message Authentication Code (MAC) values, and not hash values at all. It would be much clearer if standard terminology were to be used.

A Reflection Attack The two formulas for computing HASH_I and HASH_R are symmetrical with regard to swapping the initiator and the responder. This opens up the possibility of a reflection attack. In main mode with pre-shared keys, a fraudulent responder can claim the same identity as the initiator and still pass the authentication phase. The responder does this by choosing CKY-R as CKY-I and g^{x_r} as g^{x_i}, and using the initiator's identity. In this case HASH_R is equal to HASH_I, so the responder can just send back the value that the initiator sent.[7]

The related attack for a fraudulent initiator in aggressive mode does not work, as the initiator has to commit to g^{x_i} before the responder chooses g^{x_r} in a random manner.

The same attack seems to apply to the signature authenticated main mode exchange. The responder can replay all the initiator's values and the last message, and pass the authentication.

A Proposal Attack The HASH computations do not include the SA reply by the responder. This implies that an attacker can manipulate this payload without adverse effects within the protocol. Suppose the initiator sends a list of many proposals in order of preference. The least preferred proposal only provides marginal security (e.g., 40 bits or so). The attacker can now modify the responder's SA to select this weak mode, and let the rest of the exchange complete as usual. The initiator will now start using the newly negotiated SA, which is considerably weaker than it should be.

Suppose this is done in an aggressive mode negotiation for phase 1. The resulting ISAKMP SA is weak. As soon as the initiator starts to use the weak

[7] Actually, the fraudulent responder cannot read the fifth message it gets from the initiator, as it is encrypted. This does not matter, as the message that he wants to send in reply has the same format, so the encrypted fifth message can be sent back as the sixth message.

keys, the attacker can do a brute-force search for the keys. Once found, the attacker has recovered the ISAKMP SA keys and can now negotiate full-strength IPsec SAs with the initiator while pretending to be the responder. This is a clear violation of the intention of the protocol.

Another subtlety is that changing the responder's SA might change the mode being used. An attacker can thus have the responder performing the protocol in one mode, and the initiator the protocol in another mode. We have not investigated the possible interactions, but this is clearly undesirable. The responder's SA should be included in the HASH computations.

Conclusion The definition of the HASH values is unacceptably weak.

Recommendation 7 *Fix the definition of the HASH value so that it provides proper authentication.*

5.4 Parsing a String of Bytes

The PRFs, signature functions, and hash functions are each applied to strings of bytes. By themselves, these strings of bytes do not specify how they are formatted. This opens up a lot of freedom for an attacker. Let us suppose that there are two places in the protocol where a PRF function is applied with the same key but with different message formats. If these formats could generate strings of the same length, then it is in principle possible to confuse the two PRF computations. An attacker can take the result of the first PRF computation (which he might receive in the first part of the protocol) and use this value in the second PRF computation. Whether or not this leads to an actual weakness depends on too many factors, but it is clearly an undesirable property. Instead of authenticating the intended message, the PRF is authenticating a string of bytes whose interpretation is not fully defined. This violates the "Horton principle" [WS96].

Even if the set and order of the fields is known, the parsing of some of the messages depends on outside information. For example the HASH_I computation has a message that consists of six concatenated fields. The length of each field has to be derived from some context, which is open to manipulation by an attacker.

There are too many ways in which this type of attack can be attempted to analyze them all. We strongly recommend that the input to every hash function, PRF function, and signature be extended with a field that uniquely identifies the function within the system. (e.g. protocol ID, message number, version number, etc.) Furthermore, each message that is used as an input to a hash, PRF, or signature algorithm should be parseable into its constituent fields without any outside information. This can be achieved by using a Tag-Length-Value (TLV) encoding scheme along the lines of ASN.1.[8]

[8] We would not advocate using ASN.1 itself, as it is very complex.

Lesson 5 *The protocol must ensure that each string that is authenticated by a signature or a MAC is uniquely marked as to its function within the system, and that each string can be parsed into its constituent fields without any contextual information.*

5.5 Efficiency

As we noted earlier, the ISAKMP identity protection exchange can be shortened to four messages. The Main mode exchanges of IKE are implementations of the identity protection exchange of ISAKMP. The main mode exchange with signature authentication or pre-shared keys authentication can be shortened in the same way.

The main modes that use public-key encryption for authentication cannot be shortened in exactly the same way, as the responder would need to know the identity of the initiator instead of the other way around. This would change the properties of the protocol. However, it is possible to shorten the exchange to five messages, by swapping the order of the last two and merging the two adjacent messages that the responder sends into a single packet.

There might be a reason why IKE does not use the shorter versions of the protocols. If that is the case, this reason should be documented.

5.6 Comments Relating to Security

We give a selection of our security-related comments on IKE.

1. [HC98, page 11] describes how the choice of signature algorithm affects the choice of PRF algorithm used to compute the HASH values. This is an ugly construction, and should be removed. It can even affect security. Suppose the signature method uses a weird hash function like the one in section 5.1; this is perfectly suitable for a signature system, as the hash function is collision-resistant. IKE would use this hash function in HMAC mode, which is not at all secure. The overall effect might very well be to reveal SKEYID to an attacker.
   This is not as far-fetched as it seems. The RSA system allows message bits to be recovered from a signature. A system that wants to take advantage of this property could very well use a definition similar to the one we gave in section 5.1, as this produces exactly the type of value that is suitable for an RSA signature with (partial) message recovery.
2. The method used to derive the key material for the newly negotiated SA in the phase 2 quick mode contains a serious weakness. The KEYMAT value depends on SKEYID_d, the protocol, the SPI, and both nonces. The SPI values are only 32-bit, and might very well be chosen from a much smaller set of index values. This implies that it is quite likely that both the initiator and the responder will choose the same SPI value. In that case, the keying material for the two SAs (one in each direction) will be the same, which in turn may allows all kind of splicing attacks. This weakness exists whether PFS was used or not.

Recommendation 8 *Change the KEYMAT derivation in phase 2 to avoid generating the same keys for the two negotiated SAs.*

3. In [HC98, section 5.6], the New Group Mode is specified. The two HASH values used for authentication are computed in exactly the same manner. If the SA in the first message contains only a single proposal, then the SA reply will be identical, and HASH(2) will be identical to HASH(1). As a result, HASH(2) does not really provide any authentication.
   If authentication is needed, the definitions of the HASH values should be changed. If authentication is not needed, the HASH values should be removed.
4. We mentioned earlier that there is an active attack against the Identity Protection Exchange protocol of ISAKMP. In this attack the attacker pretends to be the responder, and collects the initiator's identity before abandoning the protocol execution.
   This attack works against the main mode exchanges with signature authentication and with pre-shared key authentication. It does not work against the public-key encryption authenticated versions.

5.7 General Comments

This section contains some of our other comments on IKE.

1. [HC98, section 5] defines encoding of group elements in the KE payload. This is different from other group element encodings in other parts of the protocol. Group element encodings should be consistent.
2. [HC98, section 5.3] shows a revised mode of using public-key encryption for authentication. From the properties given, this mode is superior to the mode of section 5.2. As there does not seem to be a reason to retain the modes of section 5.2, those should be removed.
3. [HC98, appendix B] defines the methods used to stretch the output length of the PRF, or to stretch other keying material. These methods are strong enough (given a good enough PRF), but can be improved. Even with a theoretically perfect PRF, the current definition does not generate uniformly distributed outputs.
   In the example given on page 37, if BLOCK1-8 is equal to BLOCK9-16, then it must also be equal to BLOCK17-24. This implies that about 2^{-64} of all possible SKEYID values cannot be generated by this construction. The same criticism applies to the stretching of encryption keys for ciphers that need a longer key. This is shown on pages 15, 19, and 38. This construction cannot generate all possible sequences of K1, K2, and K3.
   These stretching methods can easily be fixed by including a counter in the message input of the PRF. This has already been done in deriving the SKEYID_d, SKEYID_a, and SKEYID_e from SKEYID. The overall system would be simplified if a single stretching function were used for all situations.

4. The Quick mode negotiation protocol could be simplified significantly by using the ESP transform from IPsec for both encryption and authentication. The current solution uses a separate CBC encryption mode, and ad hoc message authentication based on specific HASH values. The protocols would be much easier to analyze if the ISAKMP SA were used to set up a confidential and authenticated tunnel (without replay protection), and the quick mode were built on top of that. As every ISAKMP implementation must also support IPsec, this would also simplify any implementation.

6 Conclusions

We are of two minds about IPsec. On the one hand, IPsec is far better than any IP security protocol that has come before: Microsoft PPTP, L2TP, etc. On the other hand, we do not believe that it will ever result in a secure operational system. It is far too complex, and the complexity has lead to a large number of ambiguities, contradictions, inefficiencies, and weaknesses. It has been very hard work to perform any kind of security analysis; we do not feel that we fully understand the system, let alone have fully analyzed it.

We have found serious security weaknesses in all major components of IPsec. As always in security, there is no prize for getting 90% right; you have to get everything right. IPsec falls well short of that target, and will require some major changes before it can possibly provide a good level of security.

What worries us more than the weaknesses we have identified is the complexity of the system. In our opinion, current evaluation methods cannot handle systems of such a high complexity, and current implementation methods are not capable of creating a secure implementation of a system as complex as this. We feel that the only way to salvage IPsec is to rigorously reduce the complexity by eliminating options and improving the modularization. This will require a major overhaul.

Finally, it has become clear that most of the blame for this state of affairs lies not with the people that worked on IPsec, but with the process used to develop it. Committee designs are bad enough when all you try to do is to get something to work. The IPsec experience has demonstrated that a committee design process is wholly incapable of creating a useful design for a security system. In our opinion, there is a fundamental conflict between the committee process and the property of security systems being only as strong as their weakest link. Therefore, we think that continuing the existing process and fixing IPsec based on various comments is bound to fail. The NIST-run process for selecting the new AES might serve as an example of an alternate process for developing and standardizing a security system.

We strongly discourage the use of IPsec in its current form for protection of any kind of valuable information, and hope that future iterations of the design will be improved. However, we even more strongly discourage any current alternatives, and recommend IPsec when the alternative is an insecure network. Such are the realities of the world.

References

[And93a] R. Anderson, "The Classification of Hash Functions," *Proceedings of the Fourth IMA Conference on Cryptography and Coding*, pp. 83–93, 1993. Available from `http://www.cl.cam.ac.uk/ftp/users/rja14/hash.ps.Z`.

[And93b] R. Anderson, "Why Cryptosystems Fail," *Proceedings of the 1st ACM Conference on Computer Communications Security '93*, ACM Press, Nov 1993, pp. 215–227.

[BDR+96] M. Blaze, W. Diffie, R. Rivest, B. Schneier, T. Shimomura, E. Thompson, and M. Weiner, "Minimal Key Lengths for Symmetric Ciphers to Provide Adequate Commercial Security," Jan 1996.

[BK98] A. Biryukov and E. Kushilevitz, "Improved Cryptanalysis of RC5," *Advances in Cryptology — EUROCRYPT '98 Proceedings*, Springer-Verlag, 1998, pp. 85–99.

[FKK96] A. O. Freier, P. Karlton, and P. Kocher, "The SSL Protocol, Version 3.0," Internet draft, Transport Layer Security Working Group, November 18, 1996. Available from `http://home.netscape.com/eng/ssl3/`.

[GK98] R. Glenn and S. Kent, "The NULL Encryption Algorithm and its Use with IPsec," RFC 2410, Nov 1998.

[HC98] D. Harkins and D. Carrel, "The Internet Key Exchange (IKE)," RFC 2409, Nov 1998.

[Kah67] David Kahn, *The Codebreakers, The Story of Secret Writing*, Macmillan Publishing Co., New York, 1967.

[KA98a] S. Kent and R. Atkinson, "IP Authentication Header, RFC 2402, Nov 1998.

[KA98b] S. Kent and R. Atkinson, "IP Encapsulating Security Payload (ESP)," RFC 2406, Nov 1998.

[KA98c] S. Kent and R. Atkinson, "Security Architecture for the Internet Protocol," RFC 2401, Nov 1998.

[MD98] C. Madson and N. Doraswamy, "The ESP DES-CBC Cipher Algorithm with Explicit IV," RFC 2405, Nov 1998.

[MG98a] C. Madson and R. Glenn, "The Use of HMAC-MD5-96 Within ESP and AH," RFC 2403, Nov 1998.

[MG98b] C. Madson and R. Glenn, "The Use of HMAC-SHA-1-96 Within ESP and AH," RFC 2404, Nov 1998.

[MSST98] D. Maughan, M. Schertler, M. Schneider, and J. Turner, "Internet Security Association and Key Management Protocol (ISAKMP)," RFC 2408, Nov 1998.

[NIST97a] National Institute of Standards and Technology, "Announcing Development of a Federal Information Standard for Advanced Encryption Standard," *Federal Register*, v. 62, n. 1, 2 Jan 1997, pp. 93–94.

[NIST97b] National Institute of Standards and Technology, "Announcing Request for Candidate Algorithm Nominations for the Advanced Encryption Standard (AES)," *Federal Register*, v. 62, n. 117, 12 Sep 1997, pp. 48051–48058.

[PA98] R. Pereira and R. Adams, "The ESP CBC-mode Cipher Algorithms," RFC 2452, Nov 1998.

[Pip98] D. Piper, "The Internet IP Security Domain of Interpretation for ISAKMP," RFC 2407, Nov 1998.

[Riv95] R.L. Rivest, "The RC5 Encryption Algorithm," *Fast Software Encryption, 2nd International Workshop Proceedings*, Springer-Verlag, 1995, pp. 86–96.

[RRS+98] R. Rivest, M. Robshaw, R. Sidney, and Y.L. Yin, "The RC6 Block Cipher," NIST AES Proposal, Jun 98.

[Sch94] B. Schneier, "Description of a New Variable-Length Key, 64-Bit Block Cipher (Blowfish)," *Fast Software Encryption, Cambridge Security Workshop Proceedings*, Springer-Verlag, 1994, pp. 191–204.

[SKW+98] B. Schneier, J. Kelsey, D. Whiting, D. Wagner, C. Hall, and N. Ferguson, "Twofish: A 128-Bit Block Cipher," NIST AES Proposal, Jun 98.

[SKW+99a] B. Schneier, J. Kelsey, D. Whiting, D. Wagner, C. Hall, and N. Ferguson, *The Twofish Encryption Algorithm: A 128-bit Block Cipher*, John Wiley & Sons, 1999.

[SM98] B. Schneier and Mudge, "Cryptanalysis of Microsofts Point-to-Point Tunneling Protocol (PPTP)," *Proceedings of the 5th ACM Conference on Communications and Computer Security*, ACM Press, 1998, pp. 132–141. Available from `http://www.counterpane.com/pptp.html`.

[SM99] B. Schneier and Mudge, "Cryptanalysis of Microsoft's PPTP Authentication Extensions (MS-CHAPv2)," *CQRE Proceedings*, Springer-Verlag, 1999, to appear. Available from `http://www.counterpane.com/pptp.html`. ((check URL))

[TDG98] R. Thayer, N. Doraswamy, and R. Glenn, "IP Security Document Roadmap," RFC 2411, Nov 1998.

[WFS99] D. Wagner, N Ferguson, and B. Schneier, "Cryptanalysis of FROG, " *Proc. 2nd AES Candidate Conference*, National Institute of Standards and Technologies, 1999, pp. 175–181.

[WS96] D. Wagner and B. Schneier, "Analysis of the SSL 3.0 Protocol, *The Second USENIX Workshop on Electronic Commerce Proceedings*, USENIX Press, 1996, pages 29–40. USENIX Press, 1996. Revised version available from `http://www.counterpane.com`.

[Wri87] P. Wright, *Spycatcher*, Viking Penguin Inc., 1987.