# Analysis of the IPSec Key Exchange Standard

Radia Perlman, Sun Microsystems Laboratories
Charlie Kaufman, Iris Associates

## 1 Abstract

This paper describes the purpose, history, and analysis of IKE [RFC2409], the current standard for key exchange for the IPSec protocol. We discuss some issues with the rest of IPSec, such as what services it can offer without changing the applications, and whether the AH header is necessary. Then we discuss the various protocols of IKE, and make suggestions for improvement and simplification.

## 2 Background

IPSec is an IETF standard for real-time communication security. In such a protocol, Alice initiates communication with a target, Bob. Each side authenticates itself to the other based on some key that the other side associates with it, either a shared secret key between the two parties, or a public key. Then they establish secret session keys (4 keys, one for integrity protection, and one for encryption, for each direction).

The other major real-time communication protocol is SSL [R01], standardized with minor changes by the IETF as TLS. IPSec is said to operate at "layer 3" whereas SSL operates at "layer 4". We discuss what this means, and the implications of these choices, in section 2.2.

### 2.1 ESP vs AH

There are several pieces to IPSec. One is the IPSec data packet encodings of which there are two: AH (authentication header), which provides integrity protection, and ESP (encapsulating security payload) that provides encryption and optional integrity protection. Many people argue [FS99] that AH is unnecessary, given that ESP can provide integrity protection. The integrity protection provided by ESP and AH are not identical, however. Both provide integrity protection of everything beyond the IP header, but AH provides integrity protection for some of the fields inside the IP header as well.

It is unclear why it is necessary to protect the IP header. If it were necessary, this could be provided by ESP in "tunnel mode" (where a new IP header with ESP is prepended to the original packet, and the entire original packet including IP header is considered payload, and therefore cryptographically protected by ESP). Intermediate routers can not enforce AH's integrity protection, because they do not know the session key for the Alice-Bob security association, so AH can at best be used by Bob to check that the IP header was received as launched by Alice. Perhaps an attacker could change the QOS fields, so that the packet would have gotten preferential or discriminatory treatment unintended by Alice, but Bob would hardly wish to discard a packet from Alice if the contents were determined cryptographically to be properly received, just because it travelled by a different path, or according to different handling, than Alice intended.

The one function that AH offers that ESP does not provide is that with AH, routers and firewalls know the packet is not encrypted, and can therefore make decisions based on fields in the layer 4 header, such as the ports. (Note: even if ESP is using null encryption, there is no way for a router to be able to know this conclusively on a packet-by-packet basis.) This "feature" of having routers and firewalls look at the TCP ports can only be used with unencrypted IP traffic, and many security advocates argue that IPSec should always be encrypting the traffic. Information such as TCP ports does divulge some information that should be hidden, even though routers have become accustomed to using that information for services like differential queuing. Firewalls also base decisions on the port fields, but a malicious user can disguise any traffic to fit the firewall's policy database (e.g., if the firewall allows HTTP, then run all protocols on top of HTTP), so leaving the ports unencrypted for the benefit of firewalls is also of marginal benefit.

The majority of our paper will focus on IKE, the part of IPSec that does mutual authentication and establishes session keys.

### 2.2 Layer 3 vs Layer 4

The goal of SSL was to deploy something totally at the user level, without changing the operating systems, whereas the goal of IPSec was to deploy something within the OS and not require changes to the applications. Since everything from TCP down is generally implemented in the OS, SSL is implemented as a process that calls TCP. That is why it is said to be at the "Transport Layer" (layer 4 in the OSI Reference Model). IPSec is implemented in layer 3, which means it considers everything above layer 3 as data, including the TCP header. The philosophy behind IPSec is that if only the OS needed to change, then by deploying an

IPSec-enhanced OS all the applications would automatically benefit from IPSec's encryption and integrity protection services.

There is a problem in operating above TCP. Since TCP will not be participating in the cryptography, it will have no way of noticing if malicious data is inserted into the packet stream. TCP will acknowledge such data and send it up to SSL, which will discard it because the integrity check will indicate the data is bogus, but there is no way for SSL to tell TCP to accept the real data at this point. When the real data arrives, it will look to TCP like duplicate data, since it will have the same sequence numbers as the bogus data, so TCP will discard it. So in theory, IPSec's approach of cryptographically protecting each packet independently is a better approach.

However, if only the operating system changes, and the applications and the API to the applications do not change, then the power of IPSec cannot be fully utilized. The API just tells the application what IP address is on a particular connection. It can't inform the application of which user has been authenticated. That means that even if users have public keys and certificates, and IPSec authenticates them, there is no way for it to inform the application. Most likely after IPSec establishes an encrypted tunnel, the user will have to type a name and password to authenticate to the application. So it is important that eventually the APIs and applications change so that IPSec can inform the application of something more than the IP address of the tunnel endpoint, but until they do, IPSec accomplishes the following:

- It encrypts traffic between the two nodes.
- As with firewalls, IPSec can access a policy database that specifies which IP addresses are allowed to talk to which other IP addresses.
- Some applications do authentication based on IP addresses, and the IP address from which information is received is passed up to the application. With IPSec, this form of authentication becomes much more secure because one of the types of endpoint identifiers IPSec can authenticate is an IP address, in which case the application would be justified in trusting the IP address asserted by the lower layer as the source.

## 3. Overview of IKE

IKE is incredibly complex, not because there is any intrinsic reason why authentication and session key establishment should be complex, but due to unfortunate politics and the inevitable result of years of work by a large committee. Because it is so complex, and because the documentation is so difficult to decipher, IKE has not gotten significant review.

The IKE exchange consists of two phases. We argue that the second phase is unnecessary. The phase 1 exchange is based on identities such as names, and secrets such as public key pairs, or pre-shared secrets between the two identities. The phase 1 exchange happens once, and then allows subsequent setup of multiple phase 2 connections between the same pair of identities. The phase 2 exchange relies on the session key established in phase 1 to do mutual authentication and establish a phase 2 session key used to protect all the data in the phase 2 security association.

It would certainly be simpler and cheaper to just set up a security association in a single exchange, and do away with the phases, but the theory is that the although the phase 1 exchange is necessarily expensive (if based on public keys), the phase 2 exchanges can then be much simpler and less expensive because they can use the session key created out of the phase 1 exchange. This reasoning only makes sense if there *will* be multiple phase 2 setups inside the same phase 1 exchange.

Why would there be multiple phase 2-type connections between the same pair of nodes? Here are the arguments in favor of having two phases:

1. It is a good idea to change keys periodically. You can do key rollover of a phase 2 connection by doing another phase 2 connection setup, which would be cheaper than restarting the phase 1 connection setup.
2. You can set up multiple connections with different security properties, such as integrity-only, encryption with a short (insecure, snooper-friendly) key, or encryption with a long key.
3. You can set up multiple connections between two nodes because the connections are application-to-application, and you'd like each application to use its own key, perhaps so that the IPSEC layer can give the key to the application.

We argue against each of these points:

1. If you want perfect forward secrecy when you do a key rollover, then the phase 2 exchange is not significantly cheaper than doing another phase 1 exchange. If you are simply rekeying, either to limit the amount of data encrypted with a single key, or to prevent replay after the sequence number wraps around, then a protocol designed specifically for rekeying would be simpler and less expensive than the IKE phase 2 exchange.
2. It would be logical to use the strongest protection needed by any of the traffic for *all* the traffic rather than having separate security associations in order to give weaker protection to some traffic. There might be some legal or performance reasons to want to use different protection for different forms

of traffic, but we claim that this should be a relatively rare case that we should not be optimizing for. A cleaner method of doing this would be to have completely different security associations rather than multiple security associations loosely linked together with the same phase 1 security association.

3. This case (wanting to have each application have a separate key) seems like a rare case, and setting up a totally unrelated security association for each application would suffice. In some cases, different applications use different identities to authenticate. In that case they would need to have separate Phase 1 security associations anyway.

In this paper we concentrate on the properties of the variants of Phase 1 IKE. Other than arguably being unnecessary, we do not find any problems with security or functionality with Phase 2 IKE.

## 4 Overview of Phase I IKE

There are two "modes" of IKE exchange. "Aggressive mode" accomplishes mutual authentication and session key establishment in 3 messages. "Main mode" uses 6 messages, and has additional functionality, such as the ability to hide endpoint identifiers from eavesdroppers, and negotiate cryptographic parameters.

Also, there are three types of keys upon which a phase 1 IKE exchange might be based: pre-shared secret key, public encryption key, or public signature key. The originally specified protocols based on public encryption keys were replaced with more efficient protocols. The original ones separately encrypted each field with the other side's public key, instead of using the well-known technique of encrypting a randomly chosen secret key with the other side's public key, and encrypting all the rest of the fields with that secret key. Apparently a long enough time elapsed before anyone noticed this that they felt they needed to keep the old-style protocol in the specification, for backward compatibility with implementations that might have been deployed during this time.

This means there are 8 variants of the Phase 1 of IKE! That is because there are 4 types of keys (old-style public encryption key, new-style public encryption key, public signature key, and pre-shared secret key), and for each type of key, a main mode protocol and an aggressive mode protocol. The variants have surprisingly different characteristics.

In main mode there are 3 pairs of messages. In the first pair Alice sends a "cookie" (see section 4.2) and requested cryptographic algorithms, and Bob responds with his cookie value, and the cryptographic algorithms he will agree to. Message 3 and 4 consist of a Diffie-Hellman exchange. Messages 5 and 6 are encrypted with the Diffie-Hellman value agreed upon in messages 3 and 4, and here each side reveals its identity and proves it knows the relevant secret (e.g., private signature key or pre-shared secret key).

In aggressive mode there are only 3 messages. The first two messages consist of a Diffie-Hellman exchange to establish a session key, and in the 2nd and 3rd messages each side proves they know both the Diffie-Hellman value and their secret.

### 4.1 Key Types

We argue one simplification that can be made to IKE is to eliminate the variants based on public encryption keys. It's fairly obvious why in some situations the variant of pre-shared secret keys makes sense. Secret keys are higher performance. But why the two variants on public key?

There are several reasons we can think of for the signature-key-only variant:

- Each side knows its own signature key, but may not know the other side's encryption key until the other side sends a certificate.
- If Alice's encryption key was escrowed, and her signature key was not, then using the signature keys offers more assurance that you're talking to Alice rather than the escrow agent.
- In some scenarios people would not be allowed to have encryption keys, but it is very unlikely that anyone who would have an encryption key would not also have a signature key.

But there are no plausible reasons we can come up with that would require variants based on encryption keys. So one way of significantly simplifying IKE is to eliminate the encryption public key variants.

### 4.2 Cookies

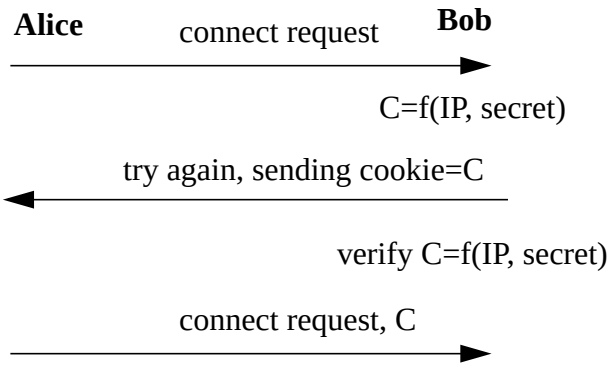
Stateless cookies were originally proposed in Photuris [K94] as a way of defending against denial of service attacks. The server, Bob, has finite memory and computation capacity. In order to prevent an attacker initiating connections from random IP addresses, and using up all of the state Bob needs in order to keep track of connections in progress, Bob will not keep any state or do any significant computation unless the connect request is accompanied by a number, known as a "cookie", that consists of some function of the IP address from which the connection is made and a secret known to Bob. In order to connect to Bob, Alice first makes an initial request, and is given a cookie. After telling Alice the cookie value, Bob does not need to remember anything about the connect request. When Alice contacts Bob again with a valid cookie, Bob will

be able to verify, based on Alice's IP address, that Alice's cookie value is the one Bob would have given Alice. Once he knows that Alice can receive from the IP address she claims to be coming from, he is willing to devote state and significant computation to the remainder of the authentication.

Cookies do not protect against an attacker, Trudy, launching packets from IP addresses at which she *can* receive responses. But in some forms of denial of service attacks the attacker chooses random IP addresses as the source, both to make it harder to catch them, and to make it harder to filter out these attacking messages.

So cookies are of some benefit. If computation were the only problem, and Bob had sufficient state to keep track of the maximum number of connect requests that could possibly arrive within the time window before he is allowed to give up and delete the state for the uncompleted connection, it would not be necessary for the cookie to be stateless. But memory *is* a resource at Bob that can be swamped during a denial of service attack, so it is desirable for Bob not to need to keep any state until he receives a valid cookie.

OAKLEY [O98] allowed the cookies to be optional. If Bob was not being attacked and therefore had sufficient resources, he could accept connection requests without cookies. A round trip delay and two messages could be saved. In Photuris the cookie (and the extra two messages) was always required. The idea behind the OAKLEY stateless cookies is:



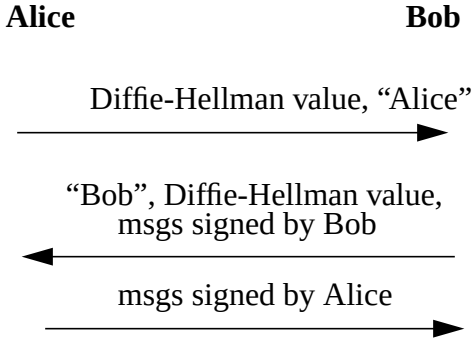

Surprisingly, although IKE was designed years after Photuris, and it has fields in the messages named "cookies", none of the IKE variants allows Bob to be stateless. This was pointed out in [S99]. In the "main mode" variants the cookie protects Bob from being forced to do a significant amount of computation. However, IKE requires Bob to keep state from the first message, before he knows whether the other side would be able to return a cookie. It would be straightforward to add two messages to IKE to allow for a stateless cookie. However, we claim that stateless cookies can be implemented in IKE main mode without additional messages by repeating in message 3 the information in message 1. Furthermore, it might be nice, in aggressive mode, to allow cookies to be optional, turned on only by the server when it is experiencing a potential denial of service attack, using the OAKLEY technique.

### 4.3 Hiding Endpoint Identities

One of the main intentions of main mode was the ability to hide the endpoint identifiers. Although it's easy to hide the identifier from a passive attacker, with some key types it is difficult to design a protocol to prevent an active attacker from learning the identity of one end or the other.

If it is impossible to hide one side's identity from an active attacker, we argue it would be better for the protocol to hide the initiator's identity rather than the responder's (because the responder is likely to be at a fixed IP address so that it can be easily found while the initiator may roam and arrive from a different IP address each day). Keeping that in mind, we'll summarize how well the IKE variants do at hiding endpoint identifiers.

In all of the aggressive mode variants, both endpoint identities are exposed, as would be expected. Surprisingly, however, we noticed that the signature key variant of aggressive mode could have easily been modified, with no technical disadvantages, to hide both endpoint identifiers from an eavesdropper, and the initiator's identity even from an active attacker! The relevant portion of that protocol is:



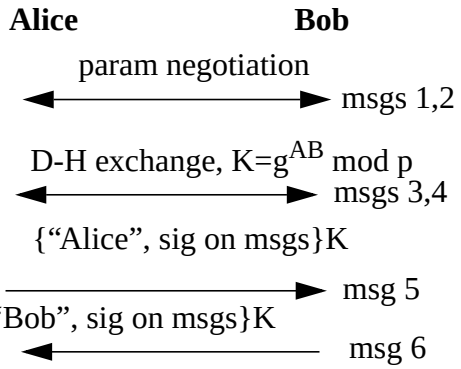

The endpoint identifiers could have been hidden by removing them from messages 1 and 2 and including them, encrypted with the Diffie-Hellman shared value, in messages 2 (Bob's identifier) and 3 (Alice's identifier).

In the next sections we discuss how the main mode protocols hide endpoint identifiers

#### 4.3.1 Public Signature Keys

In the public signature key main mode, Bob's identity is hidden even from an active attacker, but Alice's identity is exposed to an active attacker impersonating

Bob's address to Alice. The relevant part of the IKE protocol is the following:




An active attacker impersonating Bob's address to Alice will negotiate a Diffie-Hellman key with Alice and discover her identity in msg 5. The active attacker will not be able to complete the protocol since it will not be able to generate Bob's signature in msg 6.

The protocol could be modified to hide Alice's identity instead of Bob's from an active attacker. This would be done by moving the information from msg 6 into msg 4. This even completes the protocol in one fewer message. And as we said earlier, it is probably in practice more important to hide Alice's identity than Bob's.

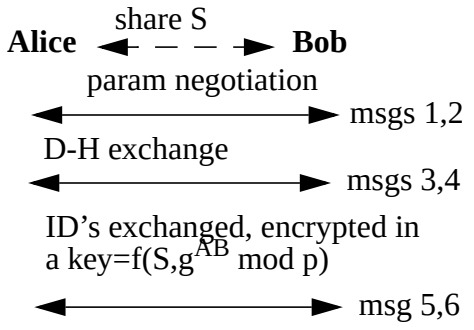
### 4.3.2 Public Encryption Keys

In this variant both sides' identities are protected even against an active attacker. Although the protocol is much more complex, the main idea is that the identities (as well as the Diffie-Hellman values in the Diffie-Hellman exchange) are transmitted encrypted with the other side's public key, so they will be hidden from anyone that doesn't know the other side's private key.

We offer no optimizations to the public encryption key variants of IKE other than suggesting their removal.

### 4.3.3 Pre-Shared Key

In this variant, both endpoints' identities are revealed, even to an eavesdropper! The relevant part of the protocol is the following:




Since the endpoint identifiers are exchanged encrypted, it would seem as though both endpoint identifiers would be hidden. However, Bob has no idea who he is talking to after message 4, and the key with which messages 5 and 6 are encrypted is a function of the pre-shared key between Alice and Bob. So Bob can't decrypt message 5, which reveals Alice's identity, unless he already knows, based on messages 1-4, who he is talking to!

The IKE spec recognizes this property of the protocol, and specifies that in this mode the endpoint identifiers have to be the IP addresses! In which case, there's no reason to include them in messages 5 and 6 since Bob (and an eavesdropper) already knows them!

Main mode with pre-shared keys is the only required protocol. One of the reasons you'd want to use IPSec is in the scenario in which Alice, an employee travelling with her laptop, connects into the corporate network from across the Internet. IPSec with pre-shared keys would seem a logical choice for implementing this scenario. However the protocol as designed is completely useless for this scenario since by definition Alice's IP address will be unpredictable if she's attaching to the Internet from different locations.

It would be easy to fix the protocol. The fix is to encrypt messages 5 and 6 with a key which is a function of the shared Diffie-Hellman value, and not also a function of the pre-shared key. Proof of knowledge of the pre-shared key is already done inside messages 5 and 6. In this way an active attacker who is acting as a man-in-the middle in the Diffie-Hellman exchange would be able to discover the endpoint identifiers, but an eavesdropper would not. And more importantly than whether the endpoint identifiers are hidden, it allows use of true endpoint identifiers, such as the employee's name, rather than IP addresses. This change would make this mode useful in the scenario (road warrior) in which it would be most valuable.

## 5. Negotiating Security Parameters

IKE allows the two sides to negotiate which encryption, hash, integrity protection, and Diffie-Hellman parameters they will use. Alice makes a proposal of a set of algorithms and Bob chooses. Bob does not get to choose 1 from column A, 1 from column B, 1 from column C, and 1 from column D, so to speak. Instead Alice transmits a set of complete proposals. While this is more powerful in the sense that it can express the case where Alice can only support certain combinations of algorithms, it greatly expands the encoding in the common case where Alice is capable of using the algorithms in any combination. For instance, if Alice can support 3 of each type of algorithm, and would be happy with any

combination, she'd have to specify 81 ($3^4$) sets of choices to Bob in order to tell Bob all the combinations she can support! Each choice takes 20 bytes to specify--4 bytes for a header and 4 bytes for each of encryption, hash, authentication, and Diffie-Hellman.

## 6. Additional Functionality

Most of this paper dealt with simplifications we suggest for IKE. But in this section we propose some additional functionality that might be useful.

### 6.1 Unidirectional Authentication

In some cases only one side has a cryptographic identity. For example, a common use case for SSL is where the server has a certificate and the user does not. In this case SSL creates an encrypted tunnel. The client side knows it is talking to the server, but the server does not know who it is talking to. If the server needs to authenticate the user, the application typically asks for a name and password. The one-way authentication is vital in this case because the user has to know he is sending his password to the correct server, and the protocol also ensures that the password will be encrypted when transmitted. In some cases security is useful even if it is only one-way. For instance, a server might be disseminating public information, and the client would like to know that it is receiving this information from a reliable source, but the server does not need to authenticate the client.

Since this is a useful case in SSL, it would be desirable to allow for unidirectional authentication within IPSec. None of the IKE protocols allow this.

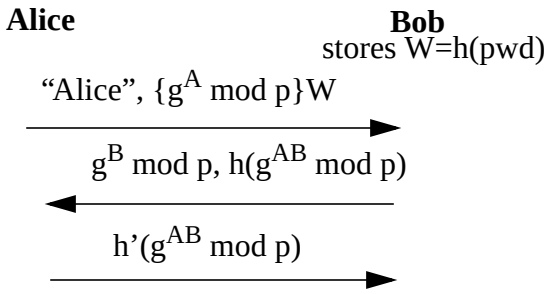
### 6.2 Weak Pre-shared Secret Key

The IKE protocol for pre-shared secrets depends on the secret being cryptographically strong. If the secret were weak, say because it was a function of a password, an active attacker (someone impersonating one side to the other) could obtain information with which to do an off-line dictionary attack. The relevant portion of the IKE protocols is that first the two sides generate a Diffie-Hellman key, and then one side sends the other something which is encrypted with a function of the Diffie-Hellman key and the shared secret. If someone were impersonating the side that receives this quantity, they know the Diffie-Hellman value, so the encryption key is a function of a known quantity (the Diffie-Hellman value) and the weak secret. They can test a dictionary full of values and recognize when they have guessed the user's secret.

The variant we suggest at the end of section 4.3.3 improves on the IKE pre-shared secret protocol by allowing identities other than IP addresses to be authenticated, but it is still vulnerable to dictionary attack by an active attacker, in the case where the secret is a weak secret. Our variant first establishes an anonymous Diffie-Hellman value, and then sends the identity, and some proof of knowledge of the pre-shared secret, encrypted with the Diffie-Hellman value. Whichever side receives this proof first will be able to do a dictionary attack and verify when they've guessed the user secret.

There is a family of protocols [BM92], [BM94], [Jab96], [Jab97], [Wu98], [KP01], in which a weak secret, such as one derived from a password, can be used in a cryptographic exchange in a way that is invulnerable to dictionary attack, either by an eavesdropper or someone impersonating either side. The first such protocol, EKE, worked by encrypting a Diffie-Hellman exchange with a hash of the weak secret, and then authenticating based on the strong secret created by the Diffie-Hellman exchange.

The ability to use a weak secret such as a password in a secure way is very powerful in the case where it is a user being authenticated. The current IKE pre-shared secret protocol could be replaced with one of these protocols at no loss in security or performance. For instance, a 3-message protocol based on EKE would look like:




The user types her name and password at the client machine, so that it can compute W. Alice sends her name, and her Diffie-Hellman value encrypted with W. Bob responds with his Diffie-Hellman value, and a hash of the Diffie-Hellman key, which could only agree with the one computed by Alice if Alice used the same W as Bob has stored. In the third message, Alice authenticates by sending a different hash of the Diffie-Hellman key.

This protocol does not hide Alice's identity from a passive attacker. Hiding Alice's identity could be accomplished by adding two additional messages at the beginning, where a separate Diffie-Hellman exchange is done, and the remaining 3 messages encrypted with that initially established Diffie-Hellman key.

## 7. Summary

The main points covered in the paper are:

- By operating below layer 3, IPSec avoids the problem of an active attacker fatally disrupting a session by injecting a single rogue packet. Solutions such as SSL, which operate above TCP, are vulnerable to this threat.
- We agree with many in the security community that AH is unnecessary,
- Although IPSec can be deployed without changes to applications, the power of IPSec cannot be exploited until the API is changed to inform the application of the endpoint identifier, and the application is modified to use the information in the modified API.
- IKE is far too complex, and the specifications are so difficult to understand that it has not gotten a thorough review, and many of the properties we point out were not known.
- IKE's second phase should be removed.
- The public encryption key variants of IKE should be removed.
- Modify IKE to allow stateless cookies (section 4.2).
- The encoding should be changed to allow negotiating sets of independent choices or cryptographic parameters, to avoid exponential explosion.
- In some modes it is only possible to hide one endpoint's identity. It is better to hide the initiator's identity. In some modes IKE only hides the responder's identity.
- The only mandated IKE key type, pre-shared secret keys, forces the endpoint identifiers to be the IP addresses in the packet. This makes this mode useless for the road warrior case. We describe a variant that allows arbitrary endpoint identifiers (such as names), and keeps them hidden from passive attackers, at no cost in security or performance.
- We describe how to hide both endpoint identifiers in the aggressive-mode signature key variant, and the initiator's identity will be hidden even from an active attacker.
- We describe how to shorten the main-mode signature key variant and improve it by hiding the initiator's identity rather than the responder's identity from an active attacker.
- We recommend adding a unidirectional authentication capability.
- We recommend replacing IKE's secret key variants with one of the strong password variants.

## 8. Bibliography


1. [BM92] S. Bellovin and M. Merritt, "Encrypted Key Exchange: Password-based protocols secure against dictionary attacks", Proceedings of the IEEE Symposium on Research in Security and Privacy, May 1992.
2. [BM94] S. Bellovin and M. Merritt, "Augmented Encrypted Key Exchange: a Password-Based Protocol Secure Against Dictionary Attacks and Password File Compromise", ATT Labs Technical Report, 1994.
3. [FS99] Ferguson, Niels, and Schneier, Bruce, "A Cryptographic Evaluation of IPSec", http://www.counterpane.com, April 1999.
4. [Jab96] D. Jablon, "Strong password-only authenticated key exchange", ACM Computer Communications Review, October 1996.
5. [Jab97] D. Jablon, "Extended Password Protocols Immune to Dictionary Attack", Proceedings of the WETICE `97 Enterprise Security Workshop, June 1997.
6. [K94] Karn, Phil, "The Photuris Key Management Protocol", Internet Draft draft-karn-photuris-00.txt, December 1994.
7. [KP01] Kaufman, Charlie, and Perlman, Radia, "PDM: A New Strong Password-Based Protocol", Usenix Security Conference, 2001.
8. [O98] Orman, Hilarie, "The OAKLEY Key Determination Protocol", RFC 2412, Nov 1998.
9. [PK00] Perlman, R. and Kaufman, C. "Key Exchange in IPSec: Analysis of IKE", IEEE Ineternet Computing, Nov/Dec 2000.
10. [R01] Rescorla, Eric, SSL and TLS: Designing and Building Secure Systems, Addison Wesley, 2001.
11. [RFC2402] Kent, Steve, and Atkinson, Ran, "IP Authentication Header", RFC 2402, Nov 1998.
12. [RFC2406] Kent, Steve, and Atkinson, Ran, "IP Encapsulating Security Payload (ESP)," RFC 2406, Nov 1998.
13. [RFC2409] Harkins,D., and Carrel, D., "The Internet Key Exchange (IKE)", RFC 2409, Nov 1998.
14. [S99] Simpson, W. A., "IKE/ISAKMP Considered Dangerous", Internet Draft, draft-simpson-danger-isakmp-00.txt, April 1999.
15. [Wu98] T. Wu, "The Secure Remote Password Protocol", ISOC NDSS Symposium, 1998.