packet to B. Show the relevant fields of the IP header(s) as given to A's IPsec layer, as transmitted by A, as transmitted by F1, and as received by B. This page is intentionally left blank 403 16 IPSEC: IKE Progress might have been all right once, but it has gone on too long. —Ogden Nash The previous chapter covered how IPsec works once the security association (SA) is set up, the session key established, and so on. This chapter covers IKE (Internet Key Exchange). IKE is a protocol for doing mutual authentication and establishing a shared secret key to create an IPsec SA. IKE took many years to come out of IETF. The original contenders were Photuris (RFC 2522) and SKIP (http://skip.incog.com/inet-95.ps). Either of these protocols would have been just fine in practice. But due to committee politics, neither one was chosen and instead IKE/ISAKMP emerged, almost a decade after work began, with a protocol so complex and specification so incomprehensible that nobody had the patience to understand what was being decided upon, and so nobody had objections. The result had lots of ambiguities and flaws, though the world did manage to have interoperable implementations and a fair amount of deployment because of the power of public domain reference implementations and interoperability testing workshops. There are two ways to design a system. One is to make it so simple there are obviously no deficiencies. The other is to make it so complex there are no obvious deficiencies. —C. A. R. Hoare The specification of IKE is in three pieces; ISAKMP (Internet Security Association and Key Management Protocol, RFC 2408), IKE (RFC 2409) and the DOI (Domain of Interpretation, RFC 2407). As of the writing of this book, the IETF WG realizes the problems with IKE, especially the documentation of IKE, and will almost certainly replace it. In this chapter we describe the current IKE in detail because there is no other place in which it is described in a readable way (especially no place that also critiques it). Even if the IETF makes a decision to replace it with something else, it is likely that implementations will persist for quite some time. And, there is a lot to learn from analyzing its idiosyncracies. The intention of IKE is to do mutual authentication using some sort of long term key (preshared secret key, public signature-only key, or public encryption key), and establish a session key. In addition to variants necessitated by having different types of keys, there are variants depending 404 IPSEC: IKE 16.1 on what features you want (e.g., hiding endpoint identifiers, or the ability to negotiate cryptographic algorithms rather than having them chosen by the initiator), trading off those features against extra messages. Each of the variants has subtle differences in security properties, which we explain. We also describe some of the alternatives/precursors to IKE. 16.1 PHOTURIS Photuris was one of the two main candidates for this piece of IPsec (the other being SKIP). Photuris was basically a signed Diffie-Hellman exchange, with identity hiding by first doing an anonymous Diffie-Hellman, and using an initial stateless cookie (see §14.5.1 Cookies). Alice transmits CA, which Photuris calls a cookie, but it's not for the same purpose as Bob's stateless cookie CB. CA is just a way for Alice to keep connection attempts separate, in case she is initiating multiple simultaneous connections to Bob. Messages 3 and 4 consist of the Diffie-Hellman exchange, and the Alice CA Bob message 1 CA, CB, crypto offered message 2 CA, CB, ga mod p, crypto selected message 3 CA, CB, gb mod p message 4 CA, CB, gab mod p {"Alice", signature on previous messages} message 5 CA, CB, gab mod p {"Bob", signature on previous messages} message 6 Protocol 16-1. Photuris 16.2 SKIP 405 resulting Diffie-Hellman key is used to encrypt the identities in messages 5 and 6. In addition to the identities, the signatures on the previous messages are sent in message 5 and 6. This is somewhat simplified. There's also crypto parameter negotiation, and choosing of SPI values for each direction. (SPI identifies the SA; see §15.1.1 Security Associations). CB is for denial of service protection. It is desirable for Bob to be stateless until message 3 (when he knows that Alice can return a valid cookie). The only way he can do this is to reuse his Diffie-Hellman secret number b for many connections. But if he always uses the same b, perfect forward secrecy will be lost. Therefore

he should change his b periodically. 16.2 SKIP SKIP (Simple Key-Management for Internet Protocols) has some interesting ideas, and was at one point widely deployed. SKIP uses long term Diffie-Hellman public keys (e.g., ga mod p). Assuming Alice knows Bob's public key (gb mod p), and her own private key (a), then she can compute gab mod p, the shared secret between herself and Bob, thus establishing a session key in zero messages! If they don't already know each other's public keys, they would have to send each other certificates, or retrieve certificates from a directory, in which case it wouldn't be zero messages, of course. It's bad cryptographic practice to use a key for encrypting a lot of data. So SKIP doesn't use the shared key X = gab mod p for encrypting the data. It only uses it for encrypting the data encryption key. Each packet of data is encrypted with some key S, and has a SKIP header that contains X{S}. You could use the same S for many packets, and would be able to tell efficiently (without needing to decrypt X{S}) that the key was still S because the header would start with the same value of X{S}. Or you could change the key on every packet. The SKIP designers really liked this feature because it somewhat got around the 40-bit key limit imposed at that time by the U.S. government. Although S was only 40 bits, if a conversation involves breaking 1000 Ss, then an attacker had 250 amount of work to do to decrypt an entire conversation. Later SKIP was modified in order to meet some of the objections of the IPsec working group. Perfect forward secrecy was added, which meant periodically doing more Diffie-Hellman exchanges, and the data packet format used AH and ESP, which added more complexity since SKIP's idea of including X{S} in each packet was not all that well-suited to being encoded with AH and ESP. Although widely deployed before IKE, once IKE was standardized most deployments migrated to IPsec. 406 IPSEC: IKE 16.3 16.3 HISTORY OF IKE While SKIP and Photuris proponents were fighting with each other, ISAKMP (Internet Security Association and Key Management Protocol, RFC 2408) emerged, a gift to the IETF from the NSA. ISAKMP wasn't a protocol but was a framework in which fields could be exchanged to create a protocol. It is specified as running over UDP rather than TCP, and is woefully underspecified on issues such as what happens when messages get lost. Given how thrilled IETF usually is at anything having to do with the NSA, and given how ISAKMP didn't actually do anything, it was astonishing that the IETF embraced it and decided that IPsec would somehow have to operate within the ISAKMP framework. Adopting ISAKMP gave an excuse not to adopt either SKIP or Photuris, because once the requirement of working with ISAKMP was assumed, neither protocol met that requirement. Another document was written, supposedly as a protocol that would work within the ISAKMP framework, and that was called OAKLEY (RFC 2412). Another proposal was SKEME [KRAW96]. Then IKE (RFC 2409) was written, crediting ideas from OAKLEY and SKEME, and using ISAKMP syntax. But even at that it was incomplete, and there's another document, The Internet IP Security Domain of Interpretation for ISAKMP (RFC 2407) that defines a lot of the fields. ISAKMP's idea of a domain of interpretation (DOI) is that a DOI specifies a particular use of ISAKMP, and the intention is that for each DOI value there would be a specification that would define what all the parameters mean for that DOI value. So you need RFCs 2407, 2408, and 2409 in order to know how to implement IKE. The distinction between IKE and ISAKMP is very confusing. Probably the best way to think of it is that IKE is a profiling (i.e., defining fields, choosing options) of ISAKMP, but it isn't that straightforward. The impression is that the IKE authors attempted to make their document self-contained but ran out of energy and deferred to the ISAKMP spec for encodings. The terms tend to be used inconsistently, adding to the confusion. For instance, a direct quote from the IKE spec (RFC 2409): While Oakley defines "modes", ISAKMP defines "phases". The relationship between the two is very straightforward and IKE presents different exchanges as modes which operate in one of two phases. —RFC 2409 Imagine trying to read 150 pages of this (80 for ISAKMP, 30 for DOI document, and 40 for

IKE), and you'll see why ISAKMP/IKE never got much scrutiny, though miraculously, people were able to implement it and even interoperate. In the next sections we'll describe the protocols conceptually, together with an analysis of them and suggestions for improvement. 16.4 IKE PHASES 407 16.4 IKE PHASES IKE defines two phases. Phase 1 does mutual authentication and establishes session keys. It is based on identities such as names, and secrets such as public key pairs, or pre-shared secrets between the two entities. Then using the keys established in phase 1, multiple phase-2 SAs between the same pair of entities can be established. The phase-1 exchange is known as the ISAKMP SA, or sometimes it is referred to as the IKE SA. An ESP or AH SA would be established through phase 2. Why not just establish an ESP or AH SA in a single exchange and not bother with a separate phase 2? It would certainly be simpler and cheaper to just set up an SA in a single exchange, and do away with the phases, but the theory is that although the phase-1 exchange is necessarily expensive (if based on public keys), the phase-2 exchanges can then be 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. Here are some arguments people give for the two phases: • The ISAKMP designers assumed ISAKMP would be used by more than just IPsec, and in addition to setting up IPsec (e.g., AH, ESP) SAs, it might be used to establish SAs for other protocols. The IETF even assigned values for DOI for some routing protocols such as RIP and OSPF, but never wound up designing exchanges for them. And indeed since those protocols run on top of IP, they didn't need their own protocol. They could use IPsec. • Some people advocate setting up different SAs for different traffic flows (conversations). In that case, a firewall-to-firewall link might require many SAs, one perhaps for each source/ destination/port pair. Even if the SA is end-to-end, there might be multiple processes on one machine talking to processes on the other. The concern is that there might be security weaknesses if different flows used the same key. Indeed such a weakness was discovered if the SA used encryption only (no integrity protection). Imagine two machines M1 and M2 with an SA over which traffic for many source/destination pairs flows. The source/destination pairs might be applications on M1 and M2, or M1 and M2 might be firewalls forwarding traffic from one portion of the net to machines on another portion of the net. Suppose there are conversations between A and B, and between C and D that go through the M1-M2 SA. If C wants to decrypt a packet sent by A to B, then it can record an encrypted packet between A and B, and between C and D, and splice the first encrypted part (the part that contains the source and destination) from the C-D packet onto the A-B encrypted packet, and forward the packet to M2. M2 will decrypt the spliced packet, deliver the decrypted data from the A-B packet to D (because the initial portion of the packet was spliced from a packet that indicated M2 should forward the packet to D). 408 IPSEC: IKE 16.5 This is not an issue if integrity protection is used, because the spliced packet would not pass the integrity check, and there is no excuse for ever using encryption without integrity protection. But because of this bug with multiplexing flows over an encryption-only SA, some people think it is safer not to multiplex flows over an SA. Without multiplexing flows, there might then be many SAs between the same pair of machines. • Key rollover is cheaper using phase 2 rather than restarting the phase-1 connection setup. • 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. We disagree with this since it would seem logical to use the strongest protection needed by any of the traffic for all the traffic rather than having separate SAs 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 this should be a relatively rare case not worth optimizing for. A cleaner method of doing this would be to have completely different SAs rather

than multiple SAs loosely linked together with the same phase-1 SA. 16.5 PHASE 1 IKE 16.5.1 Aggressive Mode and Main Mode There are two types of phase-1 exchanges, called modes. Aggressive mode accomplishes mutual authentication and session key establishment in three messages. Main mode uses six messages, and has additional functionality, such as the ability to hide endpoint identifiers from eavesdroppers and additional flexibility in negotiating cryptographic algorithms. In main mode there are six messages. In the first pair of messages, Alice sends a cookie and requested cryptographic algorithms, and Bob responds with his cookie and the cryptographic algorithms he will agree to. Messages 3 and 4 are a Diffie-Hellman exchange. Messages 5 and 6 are encrypted with the Diffie-Hellman value agreed upon in messages 3 and 4. In messages 5 and 6, 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 (Protocol 16-2), there are only three messages. The first two messages include a Diffie-Hellman exchange to establish a session key, and in the second and third messages each side proves they know both the Diffie-Hellman value and their secret. In main mode (Protocol 16-3), Alice starts by giving all the cryptographic algorithms she supports, in order of preference, and Bob responds by making a choice. In aggressive mode, Alice can also propose cryptographic algorithms, but since she has to send a Diffie-Hellman number she 16.5.1 PHASE 1 IKE 409 has to specify a unique flavor of Diffie-Hellman (e.g. g and p) and hope Bob supports it. Otherwise, Bob will refuse the connection, and not even tell her what he would have supported. Note that there is no way for Bob to cryptographically protect a message complaining about Alice's cryptographic choices, since Alice and Bob haven't established a session key yet, so Alice can't be sure a refusal message is coming from Bob. Although the specification doesn't say what Alice should do if Alice's aggressive mode message is refused, Alice should attempt to reconnect Alice ga mod p, "Alice", crypto proposal Bob message 1 gb mod p, crypto choice, proof I'm Bob message 2 proof I'm Alice message 3 Protocol 16-2. General idea for all IKE phase-1 protocols, Aggressive Mode parameter negotiation Alice crypto suites I support Bob message 1 crypto suite I choose message 2 Diffie-Hellman exchange ga mod p message 3 gb mod p message 4 send IDs and authenticate, encrypted gab mod p {"Alice", proof I'm Alice} message 5 gab mod p {"Bob", proof I'm Bob} message 6 Protocol 16-3. General idea for all IKE phase-1 protocols, Main Mode 410 IPSEC: IKE 16.5.2 with main mode, rather than retry aggressive mode with a weaker cryptographic choice (see Homework Problem 1). 16.5.2 Key Types There are three types of keys upon which a phase 1 IKE exchange might be based: pre-shared secret key, public encryption key (a public key pair whose usage is restricted to encryption and decryption), or public signature key (a public key pair whose usage is restricted to signing and signature verification). The originally specified protocols based on public encryption keys were replaced with more efficient protocols. The original public key encryption variants 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 sufficiently long 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 authentication methods (original public key encryption, revised public key encryption, public key signature, and pre-shared secret key encryption), and for each authentication method, a main mode protocol and an aggressive mode protocol. What are the arguments for supporting all these authentication methods? Variants based on pre-shared secrets might make sense because secret keys are higher performance and might be easier to configure. But why do we need three variants of public key authentication? Well, the original

public key encryption method is only there for backward compatibility. But why have both public key encryption and public key signature methods? There are several reasons for the signature-key variant: • Each side definitely starts out knowing its own signature key, but may not know the other side's encryption key until the other side sends a certificate, and that would require an extra message. • If Alice's encryption key was escrowed, and her signature key was not, then using the signature key 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. The IKE specification gives two properties that it claims the public key encryption variants have that the public key signature variants do not provide. One is plausible deniability that the conversation took place, since someone with knowledge of both public keys could construct a complete 16.5.3 PHASE 1 IKE 411 conversation (Homework Problem 3). The other is that the way the protocol is designed you'd have to break both Diffie-Hellman and RSA in order to break the protocol. Both arguments are the kind of far-fetched properties that only a theorist could get excited about. In practice neither of them matters, certainly not enough to cause the world to implement twice as many protocols and force the user to make a choice of which exotic cryptographic properties she wants for a particular conversation. But the first argument (plausible deniability that Alice talked to Bob) isn't even true. With IKE's signature variant, Alice signs a hash before she even knows who she's talking to, so the record of her IKE session can't be used to prove she had a conversation with Bob. All it can possibly prove is that she tried to use the public signature key variant of IKE to talk to somebody. As for the argument about needing to break both Diffie-Hellman and RSA, this is an extremely implausible advantage, especially when applied to a real-time communication standard (as opposed to a standard for encrypted stored data). Another point in favor of the signature key variant is that the public key encryption variants are operationally unusable in many situations, since they require Alice to know Bob's public key before she begins the exchange. There is one interesting property one might gain from using public key encryption rather than public key signatures. With public key signatures or pre-shared keys, one side has to reveal its identity to the other first. If it's the responder that reveals his identity first, then anyone can initiate an IPsec connection to Bob's IP address to find out who is there. If it's the initiator that reveals her identity first, then an active attacker might be impersonating Bob's IP address to see who might be connecting. But with public key encryption, it is possible to have both sides reveal their identity only to whom they intend to authenticate themselves, by encrypting their identity and any other identifying information (such as their certificate) with the other side's public key. But this can only be done if at least one side already knows the public encryption key of the other side. One can come up with arguments for all the key types, but is it worth specifying, implementing, and presumably asking users to decide between all these variants? I1 once explained to me2 how to make my2 house less cluttered. Pick up each item, one at a time, and ask my2self, "Would I2 pay $1.00 for this if I2 saw it at a garage sale?" This is the question the world needs to ask, for each of the variants. 16.5.3 Proof of Identity The proof of identity (transmitted in messages 2 and 3 of aggressive mode, and 5 and 6 of main mode), proves the sender knows the key associated with the identity (the pre-shared secret key, the private encryption key, or the private signature key), and it also serves as integrity protection on the previous messages. In IKE the proof of identity is different for each key type, and each consists of some hash of the key associated with the identity, the Diffie-Hellman values, the nonces, the crypto- 412 IPSEC: IKE 16.5.4 graphic choices Alice offered, and the cookies. It would have been much more straightforward to just use a hash of the previous messages in their entirety, and as it turns out, at least one of the fields left out of IKE's hash (vendor ID payload, and Bob's accepted cipher suite) could possibly

be exploited by an attacker. This was pointed out by Tero Kivinen and is something that will be most likely be fixed in a later version of IKE. One example of a field that should have been protected is the cryptographic suite Bob chooses. Theoretically, if one of the choices Alice is willing to accept is so weak that it can be broken in real time, then a man-in-the-middle, Trudy, could replace Bob's choice of a good crypto suite with that weak suite. Then, before Alice times out the connection, Trudy could break the cryptography and impersonate Bob for the remainder of the session. This is admittedly far-fetched, but given that a more straightforward protocol is more secure, there's really no excuse. Leaving fields such as the private use fields out of the integrity check might be more of an issue. By definition, we have no idea what they might be used for and whether Trudy can cause some problem by modifying them. 16.5.4 Cookie Issues Photuris had the ability for Bob to remain stateless until he knew that the initiator was able to return his cookie. Like Photuris, IKE has Alice and Bob each transmit a cookie in messages 1 and 2. However, amidst all the complexity and over-engineering of the cryptographic functions, apparently nobody noticed until too late that IKE no longer had the ability for Bob to remain stateless. For instance, he has to remember the set of cryptographic proposals Alice requests in message 1 because they are included in the hashes used in the proof of identity. If Alice were to repeat the necessary information from message 1 in message 3, Bob could be stateless until receipt of message 3. With stateless cookies, Bob would send the same cookie to the same IP address for some time (until Bob changes his secret), because Bob needs to be able to reconstruct, from the IP address alone, what cookie value he would have sent. ISAKMP requires the cookies to be unique for each connection, so even if IKE took our suggestion from the previous paragraph and had Alice repeat the information from message 1 in message 3, the ISAKMP specification would forbid Bob from choosing stateless cookies. Another problem with the design of the ISAKMP/IKE cookies is duplicate connection identifiers. The IKE exchange is identified by the pair of cookies ⟨initiator cookie, responder cookie⟩. There is nothing to prevent Alice from being the initiator in one exchange and the responder in another and having the same cookie pair established. If the order in which events happens is that Alice initiates a connection, choosing X as the initiator cookie, and then someone initiates a connection to Alice, and chooses X, to which Alice chooses Y for the responder cookie, then if the responder in the exchange that Alice initiated also happened to choose Y there would be two connections with the same connection identifier, and no way for Alice to have prevented it. This is 16.5.5 PHASE 1 IKE 413 unlikely to happen (unless attackers are observing Alice's packets and doing it on purpose) since cookies are large (8 octets) and ISAKMP requires they be randomly chosen. But the randomness requirement which is needed to prevent duplicate connection identifiers is another reason why Bob can't choose cookies designed for stateless operation. A more straightforward design, which also eliminates this flaw, would be to have each side choose an SPI for identifying traffic going towards it, as is done in ESP and AH. 16.5.5 Negotiating Cryptographic Parameters IKE allows the two sides to negotiate which encryption, hash, authentication method, and DiffieHellman parameters they will use. Alice proposes acceptable suites 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 some of the algorithms in any combination. For instance, if Alice can support 3 of each of 4 types of algorithm, and would be happy with any combination, she'd have to specify 81 (34) suite choices to Bob in order to tell Bob all the combinations she can support! Each choice must specify each of encryption, hash, authentication, and Diffie-Hellman group. A much simpler scheme is to predefine a few suites,

and only allow a choice of one of the suites. This is what is done in SSL/TLS. But if the flexibility is really desired for independently mixing-and-matching cryptographic algorithms, it would be better if IKE had allowed Alice to make a set of proposals, where each set is independent, but within a set there are allowed to be choices. For instance, proposal 1 might be to use any of three algorithms for encryption, either of 2 algorithms for hash, a specific authentication method, and any of 6 Diffie-Hellman groups. Proposal 2 might be to use any of some other (possibly overlapping) set of encryption algorithms, any of a set of hash algorithms, etc. But back to IKE as specified. In the first message of phase 1 IKE, Alice makes a set of proposals for cryptographic algorithms, and Bob chooses. Main mode vs. aggressive mode is not negotiated. Alice just decides which to do, and specifies which she chose. If she chose aggressive mode, then all proposed suites have to have the same Diffie-Hellman group, and it has to be the one that she uses in message 1. Examples of algorithms to be negotiated are: • encryption algorithm (e.g., DES, 3DES, IDEA) • hash algorithm (e.g., MD5, SHA) 414 IPSEC: IKE 16.5.6 • authentication method (e.g., pre-shared keys, RSA public key signature, DSS, RSA public key encryption with the old protocol, RSA public key encryption with the new improved protocol) • Diffie-Hellman group (e.g., modular exponentiation with a particular g and p, elliptic curve with a particular set of parameters) In all of the main mode exchanges, all cryptographic algorithms can be negotiated (encryption algorithm, hash algorithm, authentication method, Diffie-Hellman group). Alice makes a proposal and Bob chooses. IKE specifies at least one algorithm of each category as MUST implement. The MUST-implements in IKE are encryption method=DES, hash=MD5 and hash=SHA, authentication method=pre-shared key, Diffie-Hellman group=modular exponentiation with a canned g and p). Optionally within a choice Alice can specify a lifetime beyond which the SA should not be used. IKE suggests that, when the lifetime is close to being exceeded, the SA should be closed and, if needed, a new SA established. The lifetime is specified as a quantity of data and/or a duration, and is considered exceeded when either limit is exceeded. If only one type of lifetime (duration or data quantity) is specified, the default for the other is assumed (duration=8 hours, data quantity=infinite). In aggressive mode, there is no way to negotiate the group for the Diffie-Hellman exchange since Alice has already chosen one and included the Diffie-Hellman value in the message. However, the Diffie-Hellman choice still has to appear in the menu and is necessary information to Bob so that he can interpret what kind of Diffie-Hellman value Alice has sent to him. Although none of the aggressive mode variants allow negotiation of the Diffie-Hellman group, some of the other cryptographic algorithms can be negotiated. The only ones that cannot be negotiated are those that Alice has to use in message 1. For instance, in both public key encryption variants, Alice may send a hash of Bob's certificate (allowing Bob to identify which of his keys is needed to decrypt the information Alice is sending him), so Alice must choose the hash algorithm. And in the aggressive mode using the revised public key encryption algorithm, Alice additionally uses secret key encryption, so she must choose the secret key encryption algorithm. She's not allowed to propose anything different once she's used something. 16.5.6 Session Keys IKE Phase 1 establishes 2 session keys: an integrity key and an encryption key for the purpose of integrity-protecting and encrypting the last of the phase 1 IKE messages and all phase 2 IKE messages. The keys are hashes of the Diffie-Hellman values used in the exchange, the nonces, the cookies, and in the case of a pre-shared secret, that secret. It also creates a keying material seed to be used to mix into the information in the phase 2 exchanges to create unique keys for the phase 2 16.5.6 PHASE 1 IKE 415 SAs. It's a little surprising that IKE doesn't establish 4 session keys (integrity and encryption for each direction), since cryptographers generally recommend using different keys in the two directions to avoid reflection attacks. And indeed IKE is vulnerable to reflection attacks, though most likely the reflection attacks can only cause

denial of service, e.g., closing SAs. IKE uses the term “prf” (for “pseudo random function”) for the kind of function you’d use as an integrity check that takes two arguments, a key and the data, and outputs a hash. Examples of such a function are DES CBC residue (see §4.3 Generating MACs), or HMAC (see §5.7 HMAC). When there are several items that are essentially hashed together, in order to fit the form factor for a prf function, the items are concatenated together in order to make it look like exactly two inputs to the prf function. IKE needs to calculate various types of keys (integrity, encryption for the IKE SA, and keys for IPsec SAs established with phase 2). First IKE hashes the information from the IKE exchange (e.g., the nonces, the cookies, the Diffie-Hellman values) to get a quantity mysteriously known as SKEYID. It is a bad name because it isn’t an ID of anything. A better name would be something like KEYSEED. Equally baffling, SKEYID is produced by hashing together different information depending on which key type was used. This was mostly due to taking the functions from the SKEME paper, which had different protocols, and sometimes had reasons for not being able to use the same SKEYID (such as to allow SKEYID to be computed before the Diffie-Hellman shared value). But these reasons, for the most part, were no longer valid for the IKE protocols. IKE could therefore have been much simpler. For readability, we’ll leave out “mod p” and assume the reader assumes exponentiation, such as gxy is intended to mean mod p. IKE defines SKEYID as: • in the case of signature public keys, prf(nonces, gxy) • in the case encryption public keys, prf(hash(nonces), cookies) • in the case of pre-shared secret key, prf(pre-shared secret key, nonces) The double hash in the case of encryption public keys probably looks strange to you. Again, there was no reason for it, nor was there any reason why the cookies had to be there, other than copying it from the SKEME paper (where they weren’t necessary either). Next, the IKE paper defines what it calls SKEYID_d, which is the secret bits used to create the other keys. It is defined as prf(SKEYID, (gxy, cookies, 0)). (Remember, IKE wants the hash to take two arguments so the last three arguments are considered to be concatenated to form one argument.) The integrity protection key is called SKEYID_a, (where “a” is for “authentication”). SKEYID_a is defined as prf(SKEYID, (SKEYID_d, (gxy, cookies, 1)). The encryption key is called SKEYID_e, and is defined as prf(SKEYID, (SKEYID_a, gxy, cookies, 2)). The proof of identity for Alice is prf(SKEYID, (gx , gy , cookies, Alice’s initial crypto-parameters proposal, Alice’s identity)). 416 IPSEC: IKE 16.5.7 The proof of identity for Bob is prf(SKEYID, (gy , gx , cookies, Alice’s initial crypto-parameters proposal, Bob’s identity)). 16.5.7 Message IDs IKE messages contain a 32-bit “message ID” which ISAKMP specifies should be randomly chosen. The IKE message ID serves the purpose that would be served with sequence numbers in most protocols, and does it less well, since in order to recognize a replay, you would have to remember all message IDs you’ve ever seen. With a sequence number, it’s much easier to recognize messages you’ve seen already. For instance, if your window size is one and the last one you processed had message n, then anything other than n+1 will be rejected as a replay or out of window. Given that the header is the same in both directions (initiator cookie, responder cookie), and the session keys are the same in both directions, not only can an attacker replay messages to the same recipient, but the attacker can also reflect messages back to the sender. The reflection problem could have been solved by using different keys in the different directions, or by reversing the order of the cookies, so that the recipient’s cookie value appears first in the message (thus treating it like IPsec’s SPIs). And using sequence numbers for the message IDs is simpler and solves the replay problem. 16.5.8 Phase 2/Quick Mode Once an IKE SA is set up between Alice and Bob, either Alice or Bob can initiate an IPsec SA through the phase 2 “quick mode” exchange (i.e., the initiator of a phase 2 SA does not have to be the same party that initiated the phase 1 SA). The quick mode exchange establishes an ESP and/or AH SA, which involves negotiating crypto parameters, optionally doing a Diffie-Hellman exchange (if

perfect forward secrecy is desired), and negotiating what traffic will be sent on the SA. 16.5.9 Traffic Selectors IPsec allows each side of a phase 2 SA to restrict the traffic sent on that SA, by IP address, protocol type (the field in the IP header that indicates UDP, TCP, etc.), and/or TCP/UDP port. This is done by having the phase 2 initiator give a proposal for what IPsec calls a "traffic selector", which is an IP address or (address, mask) pair, a port or all ports allowed, a protocol or all protocols allowed. The other side can either accept it exactly as specified, or refuse. Note that there is an asymmetry here. If the initiator of a phase 2 SA requests a larger set of addresses than the other side is configured to want, the connection will be refused, with no hint as 16.5.10 PHASE 1 IKE 417 to what traffic selector would have been acceptable. But if the other side (the one with the more restrictive configuration) initiated the SA, then it would work. Why does IKE include this feature of specifying the type of traffic that will go over the SA? Perhaps a firewall is configured to only allow certain types of traffic, in which case telling the other side your policy might detect misconfiguration and give some clue as to why traffic is not getting through. In the case where people would want many different SAs between the same pair of nodes, so that traffic from different flows are not multiplexed over a single SA, it is necessary for the two sides to agree on what traffic each SA should be used for. 16.5.10 The IKE Phase 1 Protocols We now describe all 8 Phase 1 IKE protocols. In the first message, Alice transmits her "cookie" value. After that, all messages start with the cookie pair (initiator cookie, responder cookie), and that pair serves as the IKE connection identifier. Note that in an IKE exchange between Alice and Bob, all messages start with the same cookie pair, in the same order. If Alice initiated the IKE connection, her cookie value always appears in the "initiator cookie" field. To reduce clutter, we won't write "(initiator cookie, responder cookie)" in the figures for the messages. Fields that are optional are indicated with square brackets ("[" and "]"). When the message is encrypted is it indicated by being enclosed in curly brackets. "{}". To reduce clutter, CP indicates crypto proposal, and CPA indicates crypto proposal accepted. 16.5.10.1 Public Signature Keys, Main Mode In this mode, the two parties have public keys capable of doing signatures. Both endpoint identifiers are hidden from an eavesdropper, but an active attacker can figure out the initiator's identity. The reason for including nonces in messages 3 and 4 is so that Alice and/or Bob can save themselves computation by using the same Diffie-Hellman private value for many exchanges. If they always use the same value, then there will not be perfect forward secrecy, so it's a good idea to change it periodically. Mysteriously (and for no good reason), in the public signature key variants, K is also a function of the cookies. If Bob instead appended the information from message 6 onto message 4 then the exchange would complete in 5 messages instead of 6. However, there is a disadvantage of doing that, since Alice and Bob can't be computing the Diffie-Hellman key in parallel. (See §14.8 Arranging for Parallel Computation and Homework Problem 4.) The proof of identity consists of a signature on the hash of all the information discussed in §16.5.3 Proof of Identity. 418 IPSEC: IKE 16.5.10.2 16.5.10.2 Public Signature Keys, Aggressive Mode Note that messages 2 and 3 are not encrypted, even though the same information is encrypted in the main mode public signature key variant. The identities could have been encrypted and have the exchange still be 3 messages (see Homework Problem 5). Alice CP Bob message 1 CPA message 2 ga mod p, nonceA message 3 gb mod p, nonceB message 4 compute K = f(gab mod p, nonceA, nonceB) K{"Alice", proof I'm Alice, [certificate]} message 5 K{"Bob", proof I'm Bob, [certificate]} message 6 Protocol 16-4. Public signature keys, main mode Alice CP, ga mod p, nonceA, "Alice" Bob message 1 CPA, gb mod p, nonceB, "Bob", proof I'm Bob, [certificate] message 2 proof I'm Alice, [certificate] message 3 Protocol 16-5. Public signature keys, aggressive mode 16.5.10.3 PHASE 1 IKE 419 16.5.10.3 Public Encryption Key, Main Mode, Original IKE specifies 4 different phase-1 protocols for public encryption keys, because the original protocols (main

mode and aggressive mode) were inefficient (separately encrypted multiple fields with public keys, requiring multiple private key operations). It's astonishing that they left the original protocols in the spec once they redesigned them. A problem with this variant is that in message 3 there are two fields separately encrypted with Bob's public key, so he needs to do two private key operations to decrypt it. Likewise Alice needs to do two private key operations to decrypt message 4. Another problem would occur if a nonce or a name were larger than the public key with which it is being encrypted. The spec could have defined some sort of CBC mode for encrypting something larger than a key, but it didn't. Note that with X.500 names it would not be far-fetched for a name to be very long. Alice and Bob prove they know their private keys because they are able to decrypt the nonce from the other side. They prove this both by knowing K and the hashes used in the proofs in messages 5 and 6, since they are all functions of (among other things) the nonces. Alice CP Bob message 1 CPA message 2 ga mod p, {nonceA}Bob, {"Alice"}Bob message 3 gb mod p, {nonceB}Alice, {"Bob"}Alice message 4 compute K = f(gab mod p, nonceA, nonceB) K{proof I'm Alice} message 5 K{proof I'm Bob} message 6 Protocol 16-6. Public Encryption Keys, main mode, original protocol 420 IPSEC: IKE 16.5.10.4 As we discussed in §16.5.2 Key Types, there's no way for either Alice or Bob to ask the other side to send them their certificate! If you don't already know the other side's public key, you can't use this protocol. And if neither side knows the other side's public key without their certificate, there is no way, even if Alice and Bob could request certificates in messages 1 and 2, for them to send their certificates without divulging their identity. There is an option of having Alice send, in message 3, a hash of Bob's certificate. The reasoning is that Bob might have multiple public keys, that Alice would know that he had multiple public keys, and that he wouldn't have lots of certificates so a hash of the certificate Alice happens to have for Bob's key would be recognized by Bob. 16.5.10.4 Public Encryption Key, Aggressive Mode, Original This protocol is almost the same as the main mode version except that messages 1 and 2 are removed (and crypto suites other than Diffie-Hellman group are negotiated in parallel with the other information in messages 1 and 2) and Bob provides his proof in message 2 rather than, as in main mode, doing it after Alice presents her proof. The proof consists of a hash of the nonce presented by the other side (which requires knowledge of the private key to decrypt), along with the Diffie-Hellman values and the cookie values. 16.5.10.5 Public Encryption Key, Main Mode, Revised The public encryption protocol was revised to require only a single private key operation on each side (rather than two in the original). This is done by encrypting with a secret key which is a function of the nonce, and the nonce is encrypted with the other side's public key. Thus the other side uses its private key to retrieve the nonce, but then decrypts the other fields with a secret key. The revised protocol allows Alice to optionally send Bob her certificate. It still has the problem that Alice needs to know Bob's public key. Alice CP, ga mod p, {nonceA}Bob, {"Alice"}Bob Bob message 1 CPA, gb mod p, {nonceB}Alice, {"Bob"}Alice, proof I'm Bob message 2 proof I'm Alice message 3 Protocol 16-7. Public Encryption Keys, aggressive mode, original protocol 16.5.10.6 PHASE 1 IKE 421 16.5.10.6 Public Encryption Key, Aggressive Mode, Revised 16.5.10.7 Shared Secret Key, Main Mode This is the one required protocol. And it is the most broken. One situation in which this protocol might be useful is in the "road warrior" case, where an employee's laptop is configured with a shared secret with the company's firewall. This would allow the employee to authenticate to the firewall and establish an encrypted tunnel. But the way this mode is designed requires the identities to be the IP addresses. This makes it useless in the road warrior case, because a road warrior's IP address is dependent on where she is that day. And if the identity has to be the IP address, why go to all the work of hiding it by using the 6-message main mode protocol? The problem with this protocol is that Alice sends her identity in message 5 encrypted with a key K which is a function of the shared

secret J. Bob can't decrypt message 5 in order to find out Alice CP Bob message 1 CPA message 2 KA = hash(nonceA, cookieA) {nonceA}Bob, KA{ga mod p}, KA{"Alice"}, [KA{Alice's cert}] message 3 KB = hash(nonceB, cookieB) {nonceB}Alice, KB{gb mod p}, KB{"Bob"} message 4 K = f(gab mod p, nonceA, nonceB, cookieA, cookieB) K{proof I'm Alice} message 5 K{proof I'm Bob} message 6 Protocol 16-8. Public Encryption Keys, main mode, revised protocol 422 IPSEC: IKE 16.5.10.8 who he's talking to unless he knows J, which means he needs to know who he's talking to. The working group noticed this, and rather than fixing the protocol (which wouldn't have been hard), they instead said that in this mode Alice's identity has to be her IP address! This makes it almost useless in practice, and it certainly doesn't hide identities. This protocol can be fixed to allow arbitrary identities by not making K a function of J. J is included in the hash which is the proof of identity, so it's not needed in the encryption key. Perhaps one could claim that there is an advantage of having K be a function of J, since it hides both identities from active attackers. But if Bob doesn't know in advance who will be connecting, or at least have a very small set of candidates, then you're stuck with using the IP address as the identity, which as we said, makes it almost useless, and certainly doesn't hide identities from anyone. And if Bob does know in advance who will be connecting, then there's no reason for Alice to send her identity at all. And certainly Alice knows who she's talking to, since she knows what J to use. So Bob doesn't need to tell Alice his identity, either. 16.5.10.8 Shared Secret Key, Aggressive Mode This protocol doesn't have the problem that the main mode shared secret protocol has, because the identities are not sent encrypted. Alice KA = hash(nonceA, cookieA) Bob message 1 CP, {nonceA}Bob, KA{ga mod p}, KA{"Alice"}, [KA{Alice's cert}] KB = hash(nonceB, cookieB) CPA, {nonceB}Alice, KB{gb mod p}, KB{"Bob"}, proof I'm Bob message 2 K = f(gab mod p, nonceA, nonceB, cookieA, cookieB) K{proof I'm Alice} message 3 Protocol 16-9. Public Encryption Keys, aggressive mode, revised protocol 16.5.10.8 PHASE 1 IKE 423 Alice shared secret J Bob CP message 1 CPA message 2 ga mod p, nonceA message 3 gb mod p, nonceB message 4 K = f(J, gab mod p, nonceA, nonceB, cookieA, cookieB) K{"Alice", proof I'm Alice} message 5 K{"Bob", proof I'm Bob} message 6 Protocol 16-10. Pre-shared secret, main mode Alice shared secret J Bob CP, ga mod p, nonceA, "Alice" message 1 CPA, gb mod p, nonceB, proof I'm Bob, "Bob" message 2 proof I'm Alice message 3 Protocol 16-11. Pre-shared secret, aggressive mode 424 IPSEC: IKE 16.6 16.6 PHASE-2 IKE: SETTING UP IPSEC SAS Phase-2 IKE is known as "Quick Mode". It is a 3-message protocol that negotiates parameters for the phase-2 SA, including cryptographic parameters and the SPI with which the phase-2 SA will be identified. The phase-2 exchange sends nonces and other information which get shuffled into the SKEYSEED computed in the IKE SA to compute integrity and encryption keys for the IPsec SA, It can optionally do a Diffie-Hellman exchange if it would like PFS for the IPsec SA (the IKE SA might be much longer lived than the IPsec SA, and its keying material might be stolen after the IPsec SA concludes). There is no way to negotiate Diffie-Hellman parameters in the phase-2 exchange. If her choice of Diffie-Hellman parameters is unacceptable, Bob will complain. One would think they might as well just use the Diffie-Hellman group chosen for phase 1, but phase 2 does have the initiator specify the group, so IKE allows phase 2 to use a different group. All messages in Quick Mode are encrypted with the Phase 1 SA's encryption key Kenc (which IKE calls SKEYID_e) and integrity protected with the Phase 1 SA's integrity key Kint (which IKE calls SKEYID_a). This exchange agrees upon parameters, and encryption and/or integrity keys for each direction, to be used by the created IPsec SA. Now we'll talk about the details. Each message starts with the following, unencrypted: • X, which is the pair of cookies generated in phase 1, and Alice phase-1 SA Bob X, Y, CP, traffic, SPIA, nonceA, [ga mod p] message 1 X, Y, CPA, traffic, SPIB, nonceB, [gb mod p] message 2 X, Y, ack message 3 Protocol 16-12. IKE Quick Mode 16.7 ISAKMP/IKE ENCODING 425 • Y, a 32-bit number chosen by the phase-2 initiator to

distinguish this phase-2 session setup from perhaps many phase-2 sessions simultaneously being set up within the same phase-1 session. There are two problems with this design. One is the replay problem (see §16.5.7 Message IDs). The other is that it is possible for Bob and Alice to each initiate a phase-2 exchange, and choose the same value for Y, which would create confusion. The IKE specification notes this possibility but remarks that it's "unlikely". But they could easily have reduced the probability of Y-value collisions from "unlikely" to zero by having a convention whereby the initiator of the phase 1 exchange chooses odd Y's and the responder of the phase 1 exchange chooses even Y's. The rest of each message (following X and Y) is encrypted, using SKEYID_e, and integrity protected using SKEYID_a. The IV for the first message is the final ciphertext block of the last message of phase 1 hashed with Y. The IVs for the other messages in the exchange consist of the final ciphertext block of the previous phase 2 message. The other fields in the phase 2 exchange are: • authenticator (an integrity check on the message using SKEYID_a) • proposed crypto parameters (message 1), and accepted crypto parameters (message 2) • nonces in messages 1 and 2 • optionally Diffie-Hellman values in messages 1 and 2 • optionally a description of the traffic to be sent (see §16..5.9 Traffic Selectors) 16.7 ISAKMP/IKE ENCODING In this section we describe the formats, together with our rants about the idiosyncracies. The distinction between ISAKMP and IKE is fuzzy, and definitions are spread among all three documents (RFCs 2407, 2408, and 2409). Messages have a fixed header, and then a sequence of what ISAKMP refers to as payloads. Similar in spirit to IPv6 extension headers, each payload starts with TYPE OF NEXT PAYLOAD and LENGTH OF THIS PAYLOAD. type of next payload fixed header 426 IPSEC: IKE 16.7 The payload types are: • 0 = end (i.e., no next payload) • 1 = SA (security association): contains DOI and "situation", a modifier of DOI, and must include payloads 2 and 3 • 2 = P (proposal): proposed SPI, or SPI in reverse direction • 3 = T (transform): cryptographic choices • 4 = KE (key exchange):, the Diffie-Hellman value • 5 = ID (endpoint identifier in phase 1, traffic selector in phase 2) • 6 = CERT (certificate) • 7 = CR (certificate request) (can include the name of the certifier from whom you'd like a certificate) • 8 = hash (some sort of checksum) • 9 = signature • 10 = nonce • 11 = notification • 12 = delete (subtype of notification, meaning you are closing this SPI) • 13 = vendor ID (can be thrown in to show what implementation you're using). To avoid dealing with a registry of vendor IDs, and allowing the field to be fixed size, this is an MD of some sort of string guaranteed to uniquely describe the vendor, such as its name and telephone number. • 14–127 reserved for future use type of next payload length of this payload payload … … type of next payload = 0 length of this payload last payload 16.7.1 ISAKMP/IKE ENCODING 427 • 128–255 = private use (i.e., so NSA can use it and not publish what they're using it for) 16.7.1 Fixed Header All messages start with a 28-octet fixed length header. The fields are: • initiator's cookie (8 octets) • responder's cookie (8 octets). Note this will =0 in the first message, since it is unknown at that point • next payload type • version (1 octet). This is worth ranting about. The version number field is divided into two 4- bit fields. The intention is that the top nibble is the major version number and the bottom nibble is the minor version number. We think the concept of split version number fields of this sort originated in the early 1980's with DECnet routing, and the intention of the split version number was that the major version would be incremented for incompatible changes, and the minor version would be incremented for compatible changes. A compatible change, at least for routing, might be something like the ability to treat high priority traffic preferentially. Since differences in the minor version wouldn't create incompatibilities, the minor version field served more as information to the higher version node than as anything to worry about if the versions were different. So two nodes would talk if they both supported the same major version number. But ISAKMP has all the complexity of a split version field without any actual advantages over a simple, single version number field. In ISAKMP you are required to

reject the connection if the other guy's version field is larger than yours. (Actually, it says you must reject it if # octets 8 initiator's cookie 8 responder's cookie 1 next payload 1 version number (major/minor) 1 exchange type 1 flags 4 message ID 4 message length (in units of octets) (after encryption) 428 IPSEC: IKE 16.7.1 the major version is larger than yours, or if the major version is the same as yours, if the minor version is larger…which works out to the same thing as "reject if the other guy's 8-bit number is larger than yours.") So the result is exactly the same as if it was just an 8-bit field, but it's more complicated to understand and will run out of numbers more quickly than an 8- bit field since there are only 16 major version numbers. Unless someone worries about numbers getting used up and mandates that you must use up all the minor version numbers before you're allowed to bump the major version number. Anyway, ISAKMP gives no insight into what the designers were trying to accomplish by splitting the field into major and minor. The first version of ISAKMP was major=0, minor=1. The current version (as of writing this book) is major=1, minor=0. Also, ISAKMP doesn't exactly say you reject it if the version is larger than yours. It says you SHOULD reject it. So implementations are free to ignore the version number, but perhaps feel a little guilty about it. • exchange type (1 octet). The values defined are: ♦ 1 = base. An exchange type defined by ISAKMP but not used by IKE. This adds an extra message to aggressive mode, so that Alice (the initiator) can send her proposed parameters before sending her Diffie-Hellman value, so that the Diffie-Hellman group could also be negotiated. ♦ 2 = identity protection. This is what is called "main mode" in IKE. ♦ 3 = authentication only. Not used by IKE. ♦ 4 = aggressive. Same as what's called "aggressive mode" in IKE. ♦ 5 = informational. Not really an "exchange", since it's a single message without an acknowledgment, used to tell the other side something such as that you are refusing the connection because you don't like the version number. ♦ 6–31 = reserved values by ISAKMP for assignment by IANA as new ISAKMP exchange types ♦ 32–239 = to be defined within a particular DOI ♦ 240–255 = for private use • flags: bit 0 (LSB): encrypted—whether the fields after the header are encrypted bit 1: commit—A flag so badly named, and so confusingly defined in ISAKMP, that IKE wound up using the same bit and the same name for almost the opposite purpose. In ISAKMP the intention seems to be that the sender, say Bob, is saying he's not yet ready to accept messages on this SA, so Alice should wait for Bob to send an "I'm ready" message. IKE inter- 16.7.2 ISAKMP/IKE ENCODING 429 preted it as a request by Bob for Alice to acknowledge this message. So ISAKMP uses it for Bob to tell Alice to wait for Bob's ack, and IKE uses it for Bob to tell Alice to send an ack. bit 2: authentication only—this means that the fields after the header are not encrypted. This bit gives no additional information over merely not setting the "encrypted" flag. And you'd probably think that this bit would always be set to the opposite of the encrypted flag, but if it had been defined that way surely someone would have noticed this flag as being useless. Instead, this flag is only set during phase 2, when the assumption is all messages are encrypted, to note a message that isn't encrypted. The only such message would be an informational message. • message ID: this is used in phase 2 in order to tie together related packets. In other words, if lots of phase 2 SAs are being negotiated simultaneously within the same phase 1 SA, this field differentiates the messages for the different SAs. The specification says that these values must be chosen randomly so that there probably won't accidentally be two phase 2 exchanges with the same message ID (see §16.5.7 Message IDs). • message length: Length of entire message, in units of octets. 16.7.2 Payload Portion of ISAKMP Messages After the fixed header comes a set of ISAKMP "payloads", reminiscent of IPv6 "next headers". Each one starts with four octets consisting of:. The encoding would be more intuitive to have each payload indicate the type of that payload rather than the following one, but this way works too. It's this way because it looks more like IPv6.

16.7.3 SA Payload Assembly of SA payload requires great peace of mind. —Paraphrase of Robert Pirsig, from Zen and the Art of Motorcycle Maintenance # octets 1 type of next payload 1 reserved (unused, set to zero) 2 length of this payload 430 IPSEC: IKE 16.7.3.1 The SA payload for IKE includes the P (proposal) and T (transform) payloads. The encoding is extremely confusing for no good reason. The SA, P and T each look like independent payloads, but ISAKMP defines Ts as being carried inside a P, and Ps carried inside an SA payload. For example, if you have an SA payload that includes 2 proposals, the first of which includes 4 transforms, and the second of which includes 2 transforms, you'd have the payloads SA, P, T, T, T, T, P, T, T. 16.7.3.1 Ps and Ts within the SA Payload The P payload indicates what "protocol" you're trying to negotiate, e.g., phase 1 IKE, ESP, AH, or IP compression. For phase 1 IKE, there would only be a single P within an SA, because you're only trying to negotiate phase 1 IKE. For phase 2 IKE, there might be several Ps within an SA, because you might be making a proposal for AH only, ESP only, AH+ESP, or any of those plus IP compression. Within a P there are a set of T payloads. Each T payload indicates a complete suite of cryptographic algorithms needed by that P. For instance, for Phase 1 IKE, you need 4 (authentication (integrity protection), hash, encryption, and Diffie-Hellman group), and optionally a lifetime. There is a default lifetime of 8 hours, so if no lifetime appears within a T, it is the same as including it explicitly with 8 hours. For an AH proposal, there's only an authentication (integrity protection) algorithm, and optionally a lifetime. Each P payload is assigned a number by the initiator. If there are two P payloads with the same number, it implies both payloads constitute a single proposal. For instance, if Alice would like to have the SA do both ESP and AH, she would include two P payloads, both with the same proposal number, and if Bob accepts that proposal number, he is accepting an SA that will do both ESP and AH. Of the P numbers offered by Alice, Bob chooses one or refuses them all. Each T payload also includes a number. For instance, suppose a P payload has been assigned the number 3, and has associated T payloads for that P numbered 1, 2, 3. The other side might accept proposal #3, transform suite #2. 16.7.3.2 Payload Length in SA, P, and T Payloads The PAYLOAD LENGTH in the SA payload is the length of the entire set of the payloads consisting of the SA and all Ps and Ts associated with that SA. The payload length of each P is the length of that P payload plus the T payloads that follow. The payload length of each T payload is actually the length of that T payload. 16.7.3.3 Type of Next Payload The TYPE OF NEXT PAYLOAD field in the SA payload is set to whatever follows the SA and all proposals, usually 0 (nothing) for main mode phase 1 IKE, or 4 (KE payload) for aggressive mode 16.7.3.4 ISAKMP/IKE ENCODING 431 phase 1 IKE, or 10 (nonce) for phase 2 IKE. The TYPE OF NEXT PAYLOAD field in the P payload is either P or 0 (if it is the last proposal within an SA group). The NEXT PAYLOAD in a T payload is either T or 0 (if it's the last transform within a P group). 16.7.3.4 SA Payload Fields In the diagram we assume the SA payload contains two proposals, with the first containing three transforms and the second containing one transform. • type of next payload, length of this payload (explained in §16.7.3 SA Payload) SA payload containing 2 proposals, first proposal has four T payloads, second proposal has two T payloads: type of next payload length of entire bundle SA type of next payload = P P type of next payload = T T type of next payload = T T type of next payload = T T type of next payload = 0 T type of next payload = 0 P type of next payload = T T type of next payload = 0 T whatever follows (if anything) # octets 1 type of next payload 1 reserved (unused, set to zero) 2 length of this payload 4 DOI 1 for IPsec situation P payload T payload T payload T payload P payload T payload 432 IPSEC: IKE 16.7.4 • DOI (domain of interpretation). The idea of this field is so that all parameters do not need to be centrally defined and given numbers. Instead, if someone wants to define their own protocols with ISAKMP, they only need a single number assigned, a DOI value, and then all other parameters are interpreted according to that DOI. If you don't know about that DOI value you

can’t interpret the SA payload. In IKE the DOI=1. • situation: In ISAKMP the situation field is variable length, and its length and definition depend on the DOI. In IKE the situation field is defined in RFC 2407 as a 4 octet bit mask, of which 3 bits have been assigned. These bits don’t really make much sense. They were put there because the military types wanted them and they probably won’t be used for anything. ♦ least significant bit (0x01): identity only. As described in RFC 2407 this bit indicates whether there will be an identity payload during the exchange. But you don’t need a bit to tell you that. Just parse the message! Apparently someone thought it was a good idea, but nobody on the mailing list seems to remember what they wanted it for. ♦ next bit (0x02): secrecy. This means that the initiator requires military-style labels, and if this bit is set, the situation field is followed by variable-length data that specifies a sensitivity level and a compartment bitmask. ♦ next bit (0x04): integrity. This means that the initiator requires military-style labels for integrity, and if this bit is set, the situation field is followed by a military-style (sensitivity level, compartment) bitmask. If both the secrecy and integrity bits are set, then the integrity label follows the secrecy label. • P and T payloads (nested as described in §16.7.3.1 Ps and Ts within the SA Payload) 16.7.4 P Payload As explained before, this is not a free-standing payload, but is always nested inside the SA payload. # octets 1 type of next payload 1 reserved (unused, set to zero) 2 payload length (length includes nested Ts) 1 proposal number 1 protocol ID 1 SPI size 1 # of T payloads nested within this P variable SPI 16.7.5 ISAKMP/IKE ENCODING 433 • proposal number: the nickname given to this proposal so that the responder can accept this proposal by using that assigned number • protocol ID: the protocol being proposed (e.g., phase 1 IKE=0, ESP=3, AH=2, IPcomp=4) • SPI size: the size of the SPI that will be the ID of the SA this protocol is attempting to negotiate. In the case of phase 1 IKE, the SPI is the (initiator cookie, responder cookie) pair. The ISAKMP spec says in that case the SPI size can be anything, and the initiator is welcome to put whatever it wants into there, but the responder must ignore it. In the case of phase 2 IKE, the SPI is 4 octets, so the SPI size field is set to 4. • # of T payloads nested within this P: self-explanatory • SPI: The actual value of the SPI 16.7.5 T Payload As explained before, this is not a free-standing payload, but is always nested inside a P, nested inside the SA payload.. • transform number: the nickname given to this transform so that the responder can accept this transform by using that assigned number • transform ID: defined within the DOI and within the protocol (and the values are in RFC 2407). For instance, AH has values defined for MD5, SHA, and DES (CBC residue). ESP has a bunch of values defined for various encryption algorithms such as DES, 3DES, RC5, IDEA, and of course null. # octets 1 type of next payload 1 reserved (unused, set to zero) 2 length of this payload 1 transform number 1 transform ID 2 reserved variable attributes 434 IPSEC: IKE 16.7.6 • attributes: a sequence of variable-length fields. Each one consists of:. ♦ AF flag (attribute format flag)=0 means the value of this attribute fits within 2 octets. AF=1 means the value of this attribute takes more than 2 octets, so instead of putting the attribute value into the following 2 octets, the length is put there instead, and the variable length value follows. 16.7.6 KE Payload • the first 3 fields are the same as in all the other payloads • key data can be thought of as the transmitter’s Diffie-Hellman number, though in theory some exchange other than Diffie-Hellman might have been negotiated. Its length depends on the chosen crypto suite. 16.7.7 ID Payload The ID payload has different purposes in phase 1 and phase 2. In Phase 1 it is the name by which each side is known to the other, so that that identity can be authenticated. In phase 2 it is a description of the traffic to be transmitted on the SA (such as IP address ranges). Also, in phase 1, there is only one ID payload, which is the name of the transmitter. In phase 2 there are two ID payloads, one specifying a description of the sources of packets to be forwarded across the SA, and one specifying a description of the destinations of packets to be forwarded across

the SA. # octets 2 AFflag | attribute type 2 attribute length (if AF=0), attribute value (if AF=1) variable attribute value (if AF=1) # octets 1 type of next payload 1 reserved (unused, set to zero) 2 length of this payload variable key data 16.7.8 ISAKMP/IKE ENCODING 435 . • the first 3 fields are the same as in all the other payloads • ID type: several are defined in RFC 2407, such as IPv4 address, IPv4 address range (IPv4 address, mask), IPv6 address, IPv6 address range (IPv6 address, mask), domain name (e.g., prenhall.com), user name (e.g., radia@alum.mit.edu), X.500 name. 16.7.8 Cert Payload • first 3 fields, same as other payloads • certificate encoding: a few are defined in ISAKMP, such as X.509 certificate for a signature key, X.509 certificate for a public key with usage key exchange, PGP certificate, Kerberos token • certificate, the certificate, variable length # octets 1 type of next payload 1 reserved (unused, set to zero) 2 length of this payload 1 ID type 1 protocol ID 2 port variable Identification Data # octets 1 type of next payload 1 reserved (unused, set to zero) 2 length of this payload 1 certificate encoding variable certificate 436 IPSEC: IKE 16.7.9 16.7.9 Certificate Request Payload The certificate request payload contains:. • first 3 fields, same as other payloads • certificate type: same as "certificate encoding" defined in Certificate Payload • certificate authority: depends on certificate type, but for X.509 it would be the name of the issuer or trust anchor. This is intended to help the other side know which certificate to send. 16.7.10 Hash/Signature/Nonce Payloads These are all so straightforward and similar that to save space we're putting them in the same section. They all consist of the standard first 3 payload fields, followed by the hash or signature or nonce (depending on the payload type). 16.7.11 Notify Payload This can be used to inform the other side of error conditions, or anything else that someone might think of in the future. ISAKMP defines a lot of the notify messages, and leaves some values reserved "for private use" or "DOI specific". You might think that ISAKMP would not need to reserve values for DOI-specific. If the DOI in the notify payload indicates 0 (meaning ISAKMP), you'd go to the ISAKMP document to find the definition of that notify type. If the DOI is something else, you'd go to the spec defining those DOI-specific values. # octets 1 type of next payload 1 reserved (unused, set to zero) 2 length of this payload 1 certificate type variable certificate authority # octets 1 type of next payload 1 reserved (unused, set to zero) 2 length of this payload variable hash or signature or nonce 16.7.12 ISAKMP/IKE ENCODING 437 However, ISAKMP's assumption is that if ISAKMP defines a value, the ISAKMP definition applies, regardless of the DOI. But for all DOIs other than ISAKMP, they would assign values from the same space as all non-ISAKMP DOIs and the definitions in their specification would apply. • first 3 fields, as in other payloads • DOI, the DOI under which the rest of the payload should be interpreted • protocol ID, (e.g., AH, ESP) • SPI size: size of the SPI. • notify message type: something like "error: authentication failed" • SPI • notification data: whatever extra data might be associated with that notify message type. 16.7.12 Vendor ID Payload This is defined in ISAKMP. It's intended to allow you to announce what vendor-proprietary extensions you support. The vendor ID is supposed to be a hash of a string such as "Example Company IPsec version 97.1", so that it is highly likely to be unique. It is OK to include multiple vendor ID payloads, to alert the other side that you support the proprietary extensions of all those vendors. # octets 1 type of next payload 1 reserved (unused, set to zero) 2 length of this payload 4 DOI 1 protocol ID 1 SPI size 2 notify message type variable SPI variable notification data # octets 1 type of next payload 1 reserved (unused, set to zero) 2 length of this payload variable vendor ID 438 IPSEC: IKE 16.8 16.8 HOMEWORK 1. Suppose if Alice's aggressive-mode IKE connection initiate is refused, Alice starts up another aggressive-mode connection initiate with her next (and weaker) choice of DiffieHellman group, rather than starting a main-mode exchange telling Bob all her supported Diffie-Hellman groups. What is the vulnerability, given an active attacker? (See §16.5.1 Aggressive Mode and Main Mode.) 2. What are the relative advantages of the various key types (pre-shared secret keys, public signature

keys, public encryption keys) as a basis for an authentication exchange? 3. Show how someone who knows both Alice's and Bob's public encryption keys (and neither side's private key) can construct an entire IKE exchange based on public encryption keys that appears to be between Alice and Bob. 4. Write out the shortened version of main-mode public signature keys that hides Alice's ID from an active attacker. Explain why the 6-message version described in §16.5.10.1 Public Signature Keys, Main Mode allows parallel computation of gab mod p, whereas the shortened version does not. 5. How can you modify aggressive mode with public signature keys (see §16.5.10.2 Public Signature Keys, Aggressive Mode) to hide the endpoint identifiers from eavesdroppers? Which identity will be hidden from an active attacker? Give a disadvantage of this. (Hint: see §16.8 Arranging for Parallel Computation.) 6. For Photuris, and for each of the Phase 1 IKE variants, say how it performs on hiding endpoint identifiers. Does it hide the initiator's and/or the responder's IDs from eavesdroppers? How about active attackers? (Hint for one tricky case: what are the implications if Alice sends the hash of Bob's certificate as she can optionally do in the public encryption variants?) 7. Design a protocol in which one side has a public signature key and the other side has a public encryption key. 8. Design a protocol in which authentication is one-way since only one side has a public key. Do the protocol with a public signature key. Now do it with a public encryption key. 9. In the public encryption key case, SKEYID is defined as hash(nonces, cookies). SKEYID is supposed to be something that is not computable except by Alice and Bob. Why can't an eavesdropper or active attacker calculate SKEYID? PART 4 ELECTRONIC MAIL This page is intentionally left blank 441 17 ELECTRONIC MAIL SECURITY This chapter gives an overview of the issues involved in making electronic mail secure. First we discuss non-security-related issues such as distribution lists and mail forwarders. Then we list various security features, and in the remainder of the chapter we describe how each of these features might be implemented. A lot of the techniques are equally applicable to encrypting or integrity-protecting any sort of data, such as files kept on a file server or data stored on backup media. In the following chapters we discuss particular secure mail standards—S/MIME, PEM, and PGP. Another standard, X.400, was a collaboration between CCITT and ISO. The security aspects of it were never fully specified and never caught on, though some of the terminology has caught on, and some of the concepts (such as proof of delivery) are interesting to mention, since these concepts are not in S/MIME, PEM, or PGP. We won't discuss X.400 in detail, but we will mention some of the functionality the designers envisioned. 17.1 DISTRIBUTION LISTS Electronic mail allows a user to send a message to one or more recipients. The simplest form of electronic mail message is where Alice sends a message to Bob. Usually, a mail system allows a message to be sent to multiple recipients, for example: To: Bob From: Alice Care to meet me in my apartment tonight? 442 ELECTRONIC MAIL SECURITY 17.1 Sometimes it is impossible or inconvenient to list all recipients. For this reason, mail is often sent to a distribution list, a name that stands for a set of recipients. For instance, a message might be sent to Taxpayers. There are two ways of implementing distribution lists. The first way involves sending the message to a site at which the list is maintained, and that site then sends a copy of the message to each recipient on the list. We'll call that the remote exploder method. The second method is for the sender to retrieve the list from the site where it is kept, and then send a copy of the message to each recipient on the list. We'll call that the local exploder method. Sometimes a member of a distribution list can be another distribution list. For instance, the mailing list Security Customers, used to advertise security products, might include law enforcers, bankers, Democratic National Committee, locksmiths, and members of organized To: Bob, Carol, Ted From: Alice Care to meet me in my apartment tonight? Distribution List Maintainer Figure 17-1. Remote Exploder Recipient 1 Sender Recipient 2 Recipient 3 msg msg msg msg Distribution List Maintainer Figure 17-2. Local Exploder

Recipient 1 Sender Recipient 2 Recipient 3 msg msg msg get list list 17.1 DISTRIBUTION LISTS 443 crime. It is possible to construct a distribution list with an infinite loop. Suppose someone is maintaining a mailing list for cryptographers. Someone else is maintaining one for cryptanalysts. The cryptanalysts point out that they also want to hear the information sent to the cryptographers mailing list, so the distribution list Cryptanalysts is added to the Cryptographers mailing list. And for similar reasons Cryptographers is added to Cryptanalysts. The mail system must handle infinite loops in distribution lists in a reasonable manner, i.e., it must send each recipient at least one copy of each message but not an unreasonable number of copies of any. Loops like this effectively merge the mailing lists. (See Homework Problem 1.) As we described above, there are two methods of implementing distribution lists. The advantages of the local exploder method are: • It is easier to prevent mail forwarding loops. • If there are multiple distribution lists, it is possible for the sender to prevent duplicate copies being sent to individuals on multiple lists. • If the network billing is usage-based as opposed to flat-fee, it is easier for the sender to know in advance just how much bandwidth will be consumed to transmit the message. There are several advantages to the remote exploder method: • It allows you to send to a list whose membership you are not allowed to know. (To U.S. spies living abroad from the IRS: friendly reminder—the tax deadline is April 15. Being caught or killed is not one of the grounds for automatic extension.) • If distribution lists are organized geographically, you need only send one copy of a message over an expensive link to a remote area. (To Citizens of France from the U.S. government: Thanks for the big statue.) • When the distribution list is longer than the message, it is more efficient to send the message from the sender to the distribution list site than to send the distribution list to the sender. (To people of planet earth: Greeting. Unless you stop transmitting reruns of the I Love Lucy show to us we will be forced to destroy your planet.) • When distribution lists are included on distribution lists, it would be time-consuming to track down the whole tree to get to all the individuals. Instead, the message can be making progress as it is sent to the various exploders. Parallelism is exploited. 444 ELECTRONIC MAIL SECURITY 17.2 17.2 STORE AND FORWARD The simplest implementation of electronic mail consists of sending a message directly from the source machine to the destination machine. In order for a message to be successfully delivered under that scenario, it is necessary for both the source and destination machines to be running, and reachable from each other on the network. This might be especially inconvenient if the user machines are only occasionally connected to the network, for example a portable PC that dials into the network periodically. Thus came the concept of electronic post office boxes. Instead of sending mail directly to the user's workstation, the mail is instead sent to a machine which is more or less permanently on the network. When the user's workstation attaches to the network, it reads the mail from the appropriate mail storage machine. In general, the mail infrastructure consists of a whole mesh of mail forwarders. X.400 calls them Message Transfer Agents, or MTAs. The mail processing at the source and destination machines is done by a program known by X.400 as the User Agent or UA. Mail is not simply sent from the source machine to a mailbox machine for the destination. Instead, it gets forwarded from UA to MTA to ... to MTA to UA. Some reasons for needing multiple MTAs along a path from source to destination are: • The path from source to destination might be intermittent. For instance, there might be portions of the network that are only occasionally connected, via some dial-up link. • For security reasons, the MTAs might need to authenticate other MTAs as well as user machines. It might then be necessary to have a chain of MTAs to find a path from source to destination where each link is between a pair of MTAs that trust one another. • For security reasons, a company might want a security gateway, a place through which all mail has to be forwarded. Usually the purpose of such a gateway is to prevent any access to the company's

network except for mail. For instance, it would prevent people from logging in from a site external to the company's network. Figure 17-3. Mail Infrastructure Operation UA UA MTA MTA MTA MTA MTA MTA MTA MTA MTA MTA 17.3 SECURITY SERVICES FOR ELECTRONIC MAIL 445 • Different parts of the network might be using different protocol suites, for instance TCP/IP in some places and OSI in others. There are interesting issues with how one supports routing with MTAs. • How do MTAs find out about "neighbor" MTAs? This is usually done with manual configuration, though potentially they might be able to find each other in a directory service. • How do MTAs compute a path to a destination, or at least find the next "closer" MTA? Again, this is usually done with manual configuration. 17.3 SECURITY SERVICES FOR ELECTRONIC MAIL This section describes the kinds of security services one might desire for electronic mail. In following sections we describe how these features might be implemented. Most electronic mail systems do not provide most of these features. Even those designed specifically for security often only provide for some of these features. • privacy—the ability to keep anyone but the intended recipient from reading the message. (see §17.5 Privacy) • authentication—reassurance to the recipient of the identity of the sender. (see §17.6 Authentication of the Source) • integrity—reassurance to the recipient that the message has not been altered since it was transmitted by the sender. (see §17.7 Message Integrity) • non-repudiation—the ability of the recipient to prove to a third party that the sender really did send the message. This feature is also sometimes called third party authentication. The term non-repudiation means that the sender cannot later deny sending the message. (see §17.8 Non-Repudiation) • proof of submission—verification given to the sender that the message was handed to the mail delivery system (the same basic idea as sending certified mail through the U.S. postal service). With certified postal mail you just receive proof that you sent something to a particular address on a particular date, but with electronic mail it is possible to have the mail system verify acceptance of the contents of a particular message, perhaps by signing the message digest of the contents of the message. (see §17.9 Proof of Submission) • proof of delivery—verification that the recipient received the message. Postal mail has a similar feature (return receipt requested), but again it only verifies that something was deliv- 446 ELECTRONIC MAIL SECURITY 17.4 ered on a particular date to the recipient. With electronic mail it is possible to verify the contents, as we mentioned under proof of submission. (see §17.10 Proof of Delivery) • message flow confidentiality—an extension of privacy such that Carol not only cannot know the content of the message Alice sent Bob, but cannot even determine whether Alice sent Bob a message. (see §17.11 Message Flow Confidentiality) • anonymity—the ability to send a message so that the recipient can't find out the identity of the sender. (see §17.12 Anonymity) • containment—the ability of the network to keep certain security levels of information from leaking out of a particular region. Methods for implementing this are in §17.13 Containment. • audit—the ability of the network to record events that might have some security relevance, such as that Alice sent a message to Bob on a particular date. This would be fairly straightforward to implement, but is not mentioned in any of the secure mail standards, so we don't have a section on it. • accounting—the ability of the mail system to maintain system usage statistics. In addition to providing clues for system resource management, this information allows the mail system to charge its clients according to their usage. For example, the system might charge by number of messages sent, as long as the system itself authenticates the source of each message to ensure that the proper party is billed. Again, there's not much to say about this, so we don't have a separate section on it. • self destruct—an option allowing a sender to specify that a message should be destroyed after delivery to the recipient. This allows Alice to send a message to Bob that Bob cannot forward or store. The mail system will decrypt and display the message, but then delete it. (Good morning Mr. Phelps...). This can be implemented by marking

the message as a self-destruct message, and having the mail program at the destination cooperate by deleting the message immediately after displaying it. • message sequence integrity—reassurance that an entire sequence of messages arrived in the order transmitted, without any loss. 17.4 ESTABLISHING KEYS Most security services are best provided by cryptographic means. But cryptography requires keys. If Alice wants to send a message to Bob, what keys does she have to know? How does she reliably and efficiently learn these keys? Who is involved in sending mail between Alice and Bob? Well, 17.4.1 ESTABLISHING KEYS 447 Alice and Bob, of course. But there's also the mail infrastructure. And there might be distribution list exploders. In some cases keys are shared between Alice and Bob. In other cases keys are shared between Alice and the mail infrastructure. In still others, it might be between Alice and the distribution list exploder, and between the distribution list exploder and Bob. For now, we'll explain how Alice and Bob establish the proper keys between each other. The mechanisms depend on whether public keys or secret keys are being used. When we discuss specific services, we'll explain who needs keys. 17.4.1 Establishing Public Keys If Alice wants to send a signed message to Bob, she can just sign the message and send it, either hoping Bob will already have her certificate, can obtain it if necessary, or she can include her certificate in the email message. However, if she wants to send an encrypted message to Bob, she needs to know his public key before she can construct the message. There are various methods by which she may discover Bob's public key: • she might have received Bob's public key through some secure out-of-band mechanism, and installed it on her workstation • she might obtain it through a PKI (e.g., looking it up in a directory) (see Chapter 13 PKI (Public Key Infrastructure)) • the email system could allow piggybacking of certificates (and perhaps CRLs) on email messages. Alice can send her certificates by sending Bob a signed message. If she doesn't already know Bob's public key, she can request that he send her a signed email message with his certificates (and perhaps relevant CRLs) attached. 17.4.2 Establishing Secret Keys How can Alice and Bob establish a shared secret key for email? The simplest way is with some other means of private communication, for instance by meeting in person in a private place, or by talking on the phone (if they aren't paranoid about the phone being tapped). This strategy is OK for a few scattered private parties, but doesn't scale well at all. A more scalable strategy would be for Alice to obtain a ticket for Bob from a KDC (see § 9.4 Mediated Authentication (with KDC)), and include that ticket with her first message to Bob. 448 ELECTRONIC MAIL SECURITY 17.5 17.5 PRIVACY Most people think that electronic mail is private. But there are many ways in which inquiring minds can read your messages: • An eavesdropper can listen to the message while it is being transmitted on the wire. Perhaps the network provides link encryption, in which case this one threat is avoided. • Relay nodes (routers or mail forwarders) might have software to store messages and divulge them to people other than the intended recipients. The relay node might be doing that because it has been compromised by a bad guy, or it might be the intended function of the node. For instance the relay node might be a security gateway, and the owners of the network may want the ability to monitor mail to and from the outside. There are often conflicting security needs. Alice wants to keep the contents of her message to Bob private. But Alice's employer might want the capability of monitoring her mail to ensure she isn't revealing company secrets. Another example is that citizens might want to be able to carry on private conversations, but their government might want to be able, under appropriate circumstances (of course), to monitor all communications. 17.5.1 End-to-End Privacy Alice might want to send a message to Bob in such a way that only Bob can read it. She can't depend on the network keeping the message secret, but she can ensure that nobody but Bob can read the message by using cryptography to encrypt the message. The natural assumption is that if Alice wants to send Bob an encrypted message, she encrypts it using Bob's public key (if public