Analysis of IPSec Overheads for VPN Servers

Craig Shue, Youngsang Shin, Minaxi Gupta, Jong Youl Choi
Computer Science Department, Indiana University, Bloomington, IN, U.S.A.
{cshue, shiny, minaxi, jychoi}@cs.indiana.edu

Abstract

Internet Protocol Security (IPSec) is a widely deployed mechanism for implementing Virtual Private Networks (VPNs). This paper evaluates the performance overheads associated with IPSec. We use Openswan, an open source implementation of IPSec, and measure the running times of individual security operations and also the speedup gained by replacing various IPSec components with no-ops. The main findings of this study include: VPN connection establishment and maintenance overheads for short sessions could be significantly higher than those incurred while transferring data, and cryptographic operations contribute $32 - 60\%$ *of the total IPSec overheads.*

1. Introduction

IP packets do not have any inherent security, making it easy to forge and inspect their content, including the IP addresses contained in them. As a result, there is no guarantee that a received IP packet: 1) is from the claimed sender, 2) contains the original data that the sender put in it, or 3) was not sniffed during transit. *IPSec* [10], [5] (short for *IP Sec*urity) provides a method to protect IP datagrams by defining a method for specifying the traffic to protect, how that traffic is to be protected, and to whom the traffic is sent. For both IPv4 and IPv6, it offers the choice of two protocols: Encapsulating Security Payload (ESP) [9] or Authentication Header (AH) [8]. AH provides data integrity, anti-replay protection, and proof-of-data origin on received packets. ESP provides data confidentiality in addition to all that AH provides. In order to establish and periodically refresh the necessary cryptographic parameters for both these protocols, IPSec defines another protocol called the Internet Key Exchange (IKE) [7] protocol.

Both ESP and AH protocols can be used either in *tunnel* mode or in *transport* mode. The transport mode leaves the original IP header untouched and is used to protect only the upper-layer protocols. As a result, it can only be used between two end hosts that are also cryptographic end points. The tunnel mode protects the entire IP datagram by use of encapsulation and can be used to protect traffic between two end hosts, or two gateways (e.g. routers, firewalls), or between an end host and a gateway.

A popular and widely deployed use of IPSec is in establishing Virtual Private Networks (VPNs). Through the use of cryptographic primitives, VPNs allow off-site personnel to access organizational resources over the public Internet as if they were on-site. Figure 1 shows an example of a popular VPN configuration. In order to access a resource, say Resource 1, the client establishes an IPSec *tunnel* with the organization's VPN server. Typically, the *ESP* protocol is used to ensure the confidentiality of data traversing the Internet. To communicate with Resource 1, the client forms a IP datagram with the organization's local IP address (provided by the VPN server) as the source address and the IP address of Resource 1 as the destination address. It then encapsulates this datagram in another IP datagram with its present IP address as the source and organization's VPN server's IP address as the destination. Upon receipt of this IP-in-IP datagram, the VPN server strips off the outer IP header, decrypts the inner IP datagram, and sends it to Resource 1. To Resource 1, the datagram appears as though the client was on-site. Tunnel mode is preferred to the transport mode because the client can later utilize the same tunnel to access Resource 2 remotely as well.

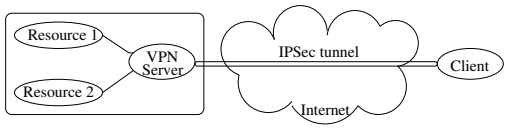

Figure 1. A popular VPN configuration.

While performance of Transport Layer Security (TLS) has been characterized [2, 1], very little research has been conducted thus far in understanding the overheads involved in using IPSec. IPSec was the focus of work in [6], where the authors examined the ESP and AH encapsulation overheads. However, a comprehensive picture of IPSec overheads was not presented because the performance penalty associated with the IKE process was not examined. This aspect needs to be understood because the IKE protocol is

usually not just run once during the establishment of a VPN connection. It is periodically run during longer VPN sessions for stronger security. Also, frequent disconnections may cause IKE to be run multiple times for clients in wireless environments. Further, the difference in overheads due to different encryption algorithms and key sizes was not investigated in the previous IPSec work [6].

In this work-in-progress paper, we present the preliminary analysis of the performance overheads associated with IPSec using Openswan [3], an open source implementation of IPSec. We focus on the tunnel mode of operation and the ESP protocol because it is the most widely deployed configuration for creating VPNs. We utilize two methods in order to analyze the performance impact of the ESP protocol, the IKE protocol, various encryption algorithms, and various cryptographic key sizes: measuring run-times for individual security operations, and replacing various IPSec components with no-ops and recording the speed-up in the run-time of various IPSec phases. The main findings from this work include: 1) overheads of the IKE protocol are considerably higher than those incurred by ESP for processing a data packet, 2) cryptographic operations contribute $32 - 60\%$ of the overheads for IKE and $34 - 55\%$ for ESP, 3) digital signature generation and Diffie-Hellman computations are the largest contributor of overheads during the IKE process and only a small amount of the overheads can be attributed to the symmetric key encryption and hashing, and 4) symmetric key encryption is the most expensive operation during the ESP process.

2. Background

IPSec integrates security at the IP layer. In order to provide higher layer services that are IPSec oblivious, it defines two new protocols, *Encapsulating Security Payload (ESP)* and *Authentication Header (AH)*. Both ESP and AH protocols encapsulate IP packets using ESP and AH headers respectively. In the case of *transport mode* of operation, the original IP header is retained and the new ESP or AH header is inserted between the IP header and the header of the higher layer transport protocol like TCP. As a result, using transport mode implies using IPSec between the actual source and destination of the IP packets. In the case of *tunnel mode* of operation, the entire IP packet to be protected is encapsulated in another IP datagram and an IPSec header is inserted between the outer and inner IP headers. The communication end points in tunnel mode are those specified in the inner header and the cryptographic end points are those appearing in the outer IP header. Due to this flexibility offered by the tunnel mode, it is the most widely used in creating VPNs. Hence, we focus on the tunnel mode in this paper.

The choice between ESP and AH protocols depends on the desired level of protection for the IP datagrams. As mentioned earlier, the AH protocol offers data integrity, anti-replay protection, and data source authentication but does not offer data privacy. The ESP protocol offers data privacy in addition to all the features offered by the AH protocol and is the protocol of choice for VPN deployment. Consequently, the subsequent description in this section focuses on the ESP protocol used for forming VPNs, even though much of it applies to AH as well. Also, the performance evaluation presented in this paper focuses on the ESP protocol.

The selection of a cryptographic mechanism is required before any IP data can be encrypted using the ESP protocol. The available primitives include using a symmetric key between the two cryptographic end points or the public keys of the end points. Since using public key encryption is computationally expensive, IPSec uses symmetric keys. But, before IPSec can use symmetric key to encrypt data, the symmetric keys must be exchanged. While out-of-band means can be used, it would not be possible to scale them for large networks. Moreover, they are not conducive to changing keys in the event the key is compromised. To accomplish this goal, IPSec defines the *Internet Key Exchange (IKE) protocol*. We now describe the IKE and ESP protocols in sections 2.1 and 2.2 respectively.

2.1. IKE Protocol

The goal of the IKE protocol is to establish and maintain shared security parameters and authenticated keys between the two IPSec end points. It uses a series of messages contained in UDP datagrams, typically directed to port 500. The IKE protocol consists of two distinct phases. The first phase establishes a symmetric IKE key between the *initiator* (VPN client in our case) and the *responder* (VPN server in our case). This key is used in the second phase to establish a symmetric IPSec key for use during ESP or AH encapsulation.

IKE defines two possible modes for the first phase of the protocol: the *main mode* and the *aggressive mode*. The main mode consists of 3 *exchanges* (6 messages in total) between the initiator and the responder. The aggressive mode requires half the number of messages but provides less negotiating capabilities; furthermore, it does not provide identity protection for the end points of the IPSec tunnel. The second phase is also called the *quick mode* and involves 3 messages.

IPSec uses the notion of *Security Association (SA)* in two different contexts. In the context of the IKE protocol, the SA defines the manner in which two end points communicate; for example, this involves agreeing on the algorithm used to encrypt traffic, the hash algorithm, and the mechanism to authenticate the other end point. IKE defines 3

categories of authentication methods (5 in total) for phase one: the first method involves the use of pre-shared keys, the next two methods use digital signatures (using RSA or digital signatures algorithms), and the last two methods use public key encryption.

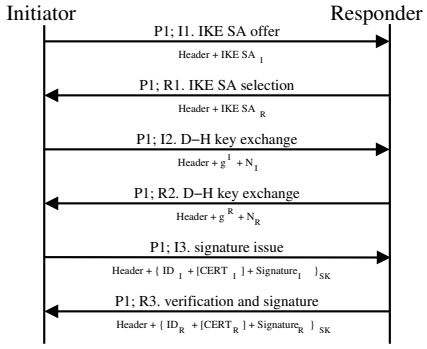

Figure 2. IKE Phase 1, Main Mode

Figure 2 shows the message exchanges for the first phase of the IKE protocol in main mode with digital signature scheme for authentication. In the first exchanges of I1 and R1, the initiator I suggests multiple SAs for the IKE protocol ($IKESA_I$) along with its cookie C_I and the responder R chooses one ($IKESA_R$) of them with its cookie C_R. In the second exchanges (I2 and R2), the initiator and the responder exchange Diffie-Hellman (D-H) public values, g^I and g^R respectively, and nonces, N_I and N_R respectively, to prevent replay attacks. In the last I3 and R3, each party computes independently the shared key SK from the previous exchanges – cookies, nonces, and D-H values – sends encryption of each identity (ID_I and ID_R respectively), a certificate for the public key verification (which is optional), and a signature on the hashed value of cookies, D-H values, SK, SAs, and identities. As a result, two parties will agree on the IKE symmetric key (SK).

When using a pre-shared secret as the authentication mechanism for phase one, the first two exchanges are identical to the digital signature approach described in figure 2. The only difference is in the third exchange, when only the identity of the end-point and a hash are sent encrypted, instead of the signature.

When aggressive mode is used instead of the main mode for phase one, the first two messages on the initiator's side are combined into a single message. Also, the responder's first, second, and third messages are combined into a single message. This cuts down the total number of messages required to establish the IKE key using phase one in half.

The second IKE phase is done in *quick mode* and accomplishes two tasks: 1) it establishes an IPSec Security Association (SA)[1] and 2) it produces an IPSec key for use during the ESP or AH encapsulation. Quick mode uses the IKE key derived from the first phase exchanges to encrypt its messages.

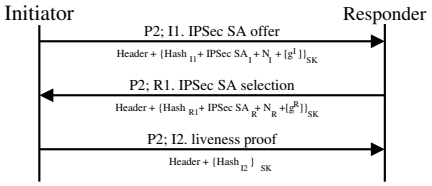

Figure 3. IKE Phase 2, Quick Mode

As shown in figure 3, the second phase of the IKE protocol is done by three message exchanges. For the first and second messages I1 and R1, the initiator I and the responder R exchange IPSec SAs ($IPSecSA_I$ and $IPSecSA_R$), nonces N_I and N_R (for replay attack protection), optional D-H values g^I and g^R, and hashes of these values and message IDs (to prove liveliness), $Hash_{I1}$ and $Hash_{R1}$. After the initiator sends one more hash ($Hash_{I2}$) using both N_I and N_R and the message ID as the third message, both parties will agree on the IPSec symmetric key for use during ESP or AH encapsulation.

For better security during longer VPN sessions, IPSec provides a mechanism to periodically refresh both IKE and IPSec keys. Refreshing the IKE key entails running both IKE phases but refreshing the IPSec key only requires running the second phase (quick mode) again.

2.2. ESP Protocol

We now describe the processing of IP packets when ESP protocol is used in tunnel mode in IPSec VPNs. For processing outbound packets, the transport layer forwards data to the IP layer, which has been enhanced with the IPSec functionality. The IP layer consults a locally maintained Security Policy Database (SPD) that defines the security services afforded to this packet. The output of the SPD dictates whether the IP layer drops the packet, bypasses security, or applies security.

If security is to be applied, the appropriate IPSec SA is consulted by looking up the SA database (SADB)[2] and the entire IP packet is encrypted and placed inside another IP packet. To facilitate the processing of this packet at the other end, an ESP header containing SA mapping information and sequence number (to prevent replay attacks) is inserted between the new IP header and the original encrypted IP packet. An ESP trailer containing Integrity Check Value

[1]The IPSec SA is different from the IKE SA. In the context of IKE, the SA meant agreeing on the authentication mechanism, encryption and hash algorithms. But, in the context of IPSec the SA defines the processing done on a specific IPSec packet by choosing between tunnel and transport modes, and between ESP and AH protocols.

[2]An IPSec SA negotiated through IKE second phase should exist at this point, if not IKE is invoked to establish it.

(ICV) is also inserted at the end of the new IP packet before sending it out.

Upon arrival of an inbound ESP packet, the SADB is consulted. If no SA exists, the received packet is discarded. Otherwise, it is de-capsulated and the appropriate IPSec key to decrypt the original IP packet is retrieved using the information contained in the ESP header of the received packet. Also, the sequence number contained in the ESP header is used to prevent replay attacks and the ICV value contained in the ESP trailer is verified to guarantee that the packet was not modified during transmission.

3. Methodology

We used Openswan [3], an open source implementation of IPSec, to examine various aspects of the IPSec protocol. This section describes the methodology used to understand the IPSec overheads, including the software and hardware used and the tests performed.

3.1. Experimental Environment

To conduct our experiments, we used two x86 Dell Optiplex GX Pentium IV machines. The first, used as the VPN server, had a 1.66GHz processor and a 100Mbps network interface card, while the second, which was used as a VPN client had a 1.8GHz processor and a 1000Mbps network card. Both the machines had 512MBytes of RAM and were connected to each other through a 100Mbps Ethernet switch.

The machines ran Debian Linux [4] with a $2.6.8$ kernel. We disabled the native IPSec support due to compatibility issues and instead used Kernel Level Internet Protocol Security (KLIPS), the shim provided by Openswan for IPSec support. The Openswan version used was $2.3.1dr3$, the latest version at the time our experiments began.

For measuring both IKE and ESP overheads we used two different approaches: 1) measuring the time taken for individual security operations (referred to as *timing measurements* subsequently) and 2) replacing various IPSec components with no-ops (referred to as *no-op measurements* subsequently). In order to characterize the amount of time spent on various cryptographic operations, we inserted assembly codes[3] to capture the CPU cycle count at which a cryptographic function started and the count at which it completed. We determined the elapsed time for each operation by gathering the elapsed cycles and multiplying this cycle count with the processor's clock speed. This technique yields nanosecond resolution, allowing for highly accurate results and minimum overhead. To get an alternate view on the overheads we replaced the computationally expensive cryptographic operations with no-ops. For both IKE and ESP, we experimented with AES and 3DES symmetric encryption schemes with key sizes of 128 and 256 bits for AES and 192 bits for 3DES. For the hashing algorithm, we tested the MD5 hashing algorithm.

[3]On Intel Pentium processors, RDTSC (Read Time Stamp Counter) instruction returns the number of clock cycles since the CPU was powered up or reset. This value is stored as a 64-bit number.

3.2. Tests Performed

IKE Testing. Out of the three categories of authentication methods prescribed for IKE phase one, Openswan supports only two: digital signatures and pre-shared secret. We examined the overheads of both of these.

The pre-shared secret authentication mechanism is the most commonly used method in real-world VPN deployments using IPSec because it does not require that the clients possess public keys. It does require additional authentication measures because organizations often keep the pre-shared secret used to establish the IKE key in public domain for easy access for their personnel. Since the overheads for the digital signature authentication mechanism are a superset of those incurred when pre-shared secret approach is used, we focus on the overheads for the digital signature approach and elaborate where the two methods diverge.

We evaluated the overheads for both *main mode* and *aggressive mode* for IKE phase one and for *quick mode* for phase two. The public and private keys required for the digital signatures method were manually configured for both the client and server. To communicate with the running IKE daemon, Openswan uses a wrapper that connects to the daemon. We timed this wrapper to measure the time to complete the IKE exchanges. We conducted 25 trials of IKE daemon start-up, connection initiation, connection termination, and tear-down.

ESP Testing. The IPSec ESP module uses the symmetric key obtained through IKE to encapsulate and decapsulate outgoing and incoming IP packets respectively. For testing its overheads, we conducted 25 trials of the time it took the IPSec VPN server to process a `ping` request packet from the client connected via an ESP tunnel.

4. Experimental Results

This section reports on the experimental results obtained for timing and no-op measurements for ESP and IKE protocols for AES (key sizes 128 and 256) and 3DES (key size 192) encryption schemes with MD5 hashing algorithm.

4.1. Timing Measurements

IKE Timings. Table 1 shows the cryptographic timing measurements for phases one and two of the IKE proto-

col for the responder (VPN server in our case) when main mode with digital signatures as the authentication method was used for phase one. The reported numbers are averaged over 25 trial runs. Here, and subsequently, the encryption algorithms and key sizes are denoted by a concatenation of the name of the algorithm and the key size (3DES192, AES128, AES256 respectively). The cryptographic operations are labeled in the same manner as in figures 2 and 3, with $P1$ and $P2$ prepended to disambiguate between the two IKE phases.

Table 1. IKE cryptographic overheads for VPN server (in ms).

MSG	OPERATION	3DES192	AES128	AES256
P1, R2	D-H comp.	17.59	17.64	17.55
P1	IKE key gen.	0.18	0.13	0.13
P1, R2	Decryption	0.17	0.06	0.06
P1, R2	Signature verif.	1.07	1.09	1.06
P1, R3	Hash verif. + gen.	0.04	0.04	0.04
P1, R3	Signature gen.	78.82	79.10	78.96
P1, R3	Encryption	0.06	0.16	0.16
P2	Decryption	0.11	0.03	0.03
P2	Hash verif.	0.02	0.02	0.02
P2, R1	D-H comp.	17.91	17.95	17.95
P2, R1	Hash gen.	0.03	0.03	0.03
P2, R1	Encryption	0.08	0.18	0.18
P2	Decryption	0.07	0.02	0.02
P2	Hash verif.	0.01	0.02	0.01
P2	IPSec key gen.	0.09	0.10	0.10
	Total:	117.59	117.93	117.66

As table 1 shows, the biggest contributer to the cryptographic overheads at the VPN server was the RSA signature generation. Out of the total 373.82ms recorded at the server for the IKE process for 3DES192[4], including 117.59ms for the cryptographic operations as shown at the end of table 1, this one operation took approximately 21% of the total time required for the IKE process for all key sizes and encryption algorithms tested. However, verification of signatures sent by the client was much more efficient (1.07ms, 1.09ms, and 1.06ms for 3DES192, AES128, and AES256 respectively).

The D-H computations were the second biggest overhead at the server, consuming a total of 35.50ms (17.59ms and 17.91ms during phases one and two respectively) for 3DES192. These overheads varied little across encryption algorithms. The overheads associated with symmetric key encryption and decryption operations in both phases are quite low and vary with the size of the data being encrypted. Due to the small difference in the key sizes available for testing, no clear conclusion about the relationship of overheads and key sizes or encryption algorithms can be drawn. Finally, hashing contributed the least to the cryptographic overheads.

The details of the overheads recorded at the initiator (VPN client) were similar to those presented here and are omitted due to space constraints. Overall, the client overheads were less since the machine used as the initiator had a 200MHz processing edge. In particular, the cryptographic overheads at the client were 107.27ms, 107.53ms, and 107.41ms for 3DES192, AES128, and AES256 respectively.

Changing the authentication method in IKE phase one main mode to pre-shared secret changed the total IKE overheads at the server to 214.99ms, 213.71ms, and 214.30ms for 3DES192, AES128, and AES256 respectively. These overheads differ from the digital signature overheads only by twice the digital signature generation overheads, one at the client and one at the server. This is expected because the remaining operations are very similar in both authentication methods. For conciseness, we have omitted the details of these results as well.

Next, we replaced the main mode by aggressive mode for IKE phase one and measured the total IKE overheads for both phases for digital signature authentication mechanism. This reduced the number of messages exchanged between the client and the server but did not change the cryptographic operations performed. The results reflected this and no difference in cryptographic overheads was observed. The overall IKE overheads decreased by only about 5ms.

Overall, the cryptographic operations contributed about 60% to the total running time of the IKE protocol for all encryption schemes and key sizes when digital signature authentication was used in main mode (32% when pre-shared key was used in main mode and 60% when digital signatures were used in aggressive mode). The remaining time was spent in the actual transmission of the messages, memory management, and other non-cryptographic operations.

ESP Timings. To infer the ESP overheads we measured the processing time of a `ping` packet at the server. Figure 4 shows the average overheads for 3DES192 over 25 trial runs when the client sends a 736 byte `ping` request to the server. The top to bottom of the y-axis in figure 4 shows the journey of an inbound ping request until the server prepares an outbound ping response packet. The inbound and outbound ESP overheads are demarcated by the dashed vertical lines and the rest of the overheads are the standard ping processing overheads incurred by the Linux TCP/IP implementation.

The key observation from figure 4 is that ESP processing for a packet is considerably faster than IKE. Also, out of the total 148.61μs inbound processing time, around 58% was contributed by symmetric key decryption. Similarly, 55% of the outbound processing time was contributed by the encryption operation. While the overheads for the hash computations were negligible in comparison with other overheads for IKE, this is not the case for ESP.

[4]This includes the IKE message processing at the client (including the cryptographic operations) because IKE messages happen in lock-step.

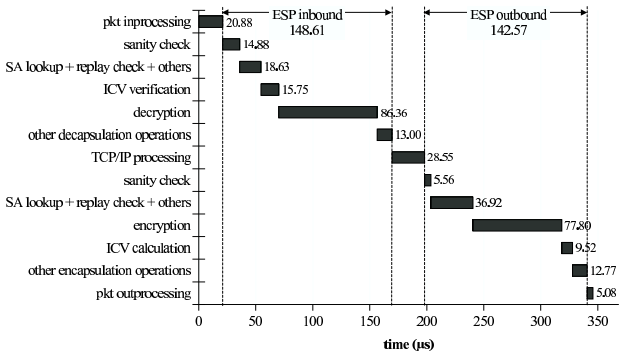

Figure 4. Server ESP overheads (3DES192).

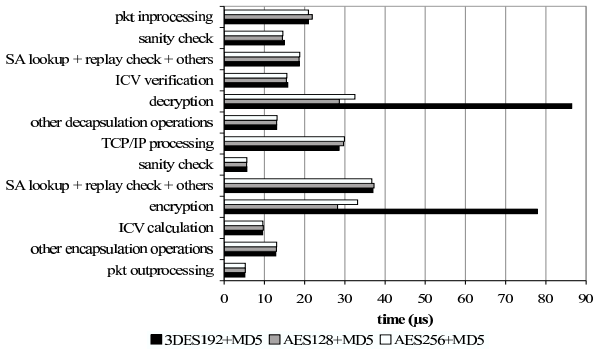

Figure 5. Comparison of ESP overheads.

Figure 5 shows the comparison between 3DES192, AES128, and AES256 for the same ping packet processing. While all other operations incur the same overhead as expected, AES256 is almost two and a half times faster than 3DES192 in spite of the bigger key size. It remains to be investigated whether this observation is hardware dependent and whether it holds true for larger transfers.

4.2. No-op Measurements

In order to infer the non-cryptographic IPSec overheads we no-oped cryptographic operations selectively and measured the speed-up. This section reports on the no-op experiments conducted.

IKE No-ops. Table 2 compares the server overheads between the full IKE implementation and its no-oped skeleton implementation where the cryptographic operations have been stripped off. Digital signatures were used for authentication during phase one, which was run in main mode.

Table 2. Comparison of full and skeleton IKE code (in ms).

	3DES192	AES128	AES256
Full IKE	373.82	370.86	371.06
Skeleton IKE	141.48	141.16	141.05

The difference between skeleton and full IKE implementations for all encryption schemes is very close to what we observed using the timing measurements and the minor deviations can be attributed to compiler optimizations that occur when the code is no-oped. The results for pre-shared secret in main mode and digital signatures in aggressive mode have been omitted due to space constraints.

ESP No-ops. Table 3 compares the processing times at the server for native TCP/IP `ping` processing, the skeleton ESP version where all the cryptographic operations have been no-oped, and the full ESP implementation (3DES192, AES128, AES256). It is noteworthy that even when all cryptographic operations are no-oped, substantial ESP overhead remains, just like in the case of IKE. The cryptographic operations contribute 55%, 34%, and 37% to the overall ESP overheads for 3DES192, AES128, and AES256 respectively. 3DES192 performs the worst of three but it remains to be investigated whether this observation holds true for larger transfer sizes as well.

Table 3. Comparison of full and skeleton ESP code (in μs).

	3DES192	AES128	AES256
Full ESP	345.69	244.13	252.16
Skeleton ESP	182.37		
Native TCP/IP	46.44		

Acknowledgments

We would like to thank Tom Zeller and Rob Henderson for insights on deployed VPNs and assistance in conducting the experiments respectively.

References

[1] G. Apostolopoulos, V. Peris, and D. Saha. Transport Layer Security: How much does it really cost? In *IEEE INFOCOM*, June 1999.

[2] C. Coarfa, P. Druschel, and D. Wallach. Performance analysis of TLS Web servers. In *NDSS*, February 2002.

[3] X. Corporation. Openswan Web-site, 2004. `http://www.openswan.org/`.

[4] Debian Linux Web-site. `http://www.debian.org/`.

[5] N. Doraswamy and D. Harkins. *IPSec: the new security standard for the Internet, intranets, and virtual private networks*. Prentice Hall, 1st edition, 1993.

[6] G. C. Hadjichristophi, N. J. Davis IV, and S. F. Midkiff. IPSec overhead in wireline and wireless networks for web and email applications. In *22nd IEEE IPCCC*, April 2003.

[7] D. Harkins and D. Carrel. The Internet Key Exchange (IKE). RFC 2409 (Proposed Standard), Nov. 1998.

[8] S. Kent and R. Atkinson. IP authentication header. RFC 2402 (Proposed Standard), Nov. 1998.

[9] S. Kent and R. Atkinson. IP encapsulating security payload. RFC 2406 (Proposed Standard), Nov. 1998.

[10] S. Kent and R. Atkinson. Security architecture for the internet protocol. RFC 2401 (Proposed Standard), Nov. 1998. Updated by RFC 3168.