# Reducing the Impact of DoS Attacks on Endpoint IP Security

Joseph D. Touch and Yi-Hua Edward Yang
*USC/ISI*
*touch@isi.edu / yeyang@isi.edu*

## Abstract

*IP security is designed to protect hosts from attack, but can itself provide a way to overwhelm the resources of a host. One such denial of service (DoS) attack involves sending incorrectly signed packets to a host, which then consumes substantial CPU resources to reject unwanted traffic. This paper quantifies the impact of such attacks and explores preliminary ways to reduce that impact. Measurements of the impact of DoS attack traffic on PC hosts indicate that a single attacker can reduce throughput by 50%. This impact can be reduced to 20% by layering low-effort nonce validation on IPsec's CPU-intensive cryptographic algorithms, but the choice of algorithm does not have as large an effect. This work suggests that effective DoS resistance requires a hierarchical defense using both nonces and strong cryptography at the endpoints, even within a single layer and single packet.*

## 1. Introduction

IPsec promises network layer security by applying a variety of encryption and authentication algorithms to IP packets [11]. Although this protects hosts from accepting attack traffic, it imposes a substantial computational burden as well. Common cryptographic algorithms reduce the network throughput of typical PCs by a factor of 5 or more. As a result, use of IPsec can paradoxically enable DoS attacks.

This paper explores the impact of false IPsec traffic as a DoS attack on hosts, and ways to mitigate that impact. Its primary contribution is to quantify this impact through direct measurements. Tests of a variety of IPsec algorithms indicate that the CPU bottleneck limits receive throughput and that known nonce-based validation provides a useful way to efficiently reject some attack traffic. These results also indicate that this technique becomes more effective as the algorithm becomes more computationally intensive, and suggest that such hierarchical intra-packet defenses are needed to avoid IPsec being itself an opportunity for attack.

This work was supported by the National Science Foundation under Grant SCI-0225642 Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF.

## 2. Background

Performance has been a significant issue for Internet security since its inception and affects the IPsec suite both at the IKE (session establishment) and IPsec (packets in a session) level [10][11]. This paper focuses on the IPsec level, *i.e.*, protection for established sessions. Previous performance analysis of HMAC-MD5 (Hashed-MAC, where MAC means keyed Message Authentication Code), HMAC-SHA1, and 3DES showed that the cost of the cryptographic algorithms dwarfs other IPsec overheads [6] [7] [15].

The typical response to "crypto algorithms are slow" is hardware support. Although DES sped up by 20x when converted from software to hardware, analysis of MD5 suggested that such speedup was not generally expected, especially for authentication (which cannot be precomputed as encryption can) [17].

IPsec performance is mostly seen as an economic issue – that is, cost determines whether a device can support high bandwidth with IPsec enabled. Most hardware vendors already consider load-based DoS attacks a risk for their devices, but often focus only on key overload (too many keys to cache) issues. This is sufficient because their devices are dedicated and can keep pace with line-rate, so there is no way to overload their CPU resources or thus impact other programs.

Hardware cryptographic support may not be feasible for many end hosts, which predominately are desktops and laptops. Such support is expensive beyond 100 Mbps and imposes a substantial power burden on laptops. Software support for IPsec is expected to be the norm, but subjects the host CPU to cryptographic processing load from external data sources, resulting in a serious DoS attack opportunity on the end hosts.

This paper quantifies the impact of this risk and explores the use of multi-layer protection within a single packet. Layered security is not new; many systems include layers of incrementally stronger and more computationally intensive checks. Photuris is a common example, where nonce exchanges precede more intensive exchanges, to reduce the impact of DoS attacks [9]. Layering security at different layers in the protocol stack is also common, *e.g.*, using link WEP, network IPsec, and transport TLS protection. The need for different layers of security within a single packet has not been as deeply explored, however, and this work quantifies its utility.

## 3. Performance Measurements

We measured the performance of IPsec on three dual-Xeon 2.4GHz FreeBSD/SMP 6.0 PCs connected via Intel PCI-X gigabit Ethernet to a copper Netgear switch. In the baseline case, traffic was sent from one host to a second host (with the third host idle). In cross-traffic experiments, the third host generated DoS attack traffic at maximum rates to show the largest impact achievable. In all cases, measurements compared HMAC-MD5 and HMAC-SHA1 authentication and DES and 3DES encryption in both IPsec transport and IPsec tunnel modes; only transport mode is shown except for Figure 6 as noted. Netperf 2.4.1 was used for measurement. We compared UDP performance, where message size corresponds to IPsec message size and which taxes IPsec the most.

### 3.1. Baseline IPsec performance

Figure 1 shows the performance of IP and IPsec transport mode with HMAC authentication using MD5 and SHA1, and IPsec with encryption using DES and 3DES[2]. This graph shows (as typical) that IPsec degrades throughput vs. non-IPsec by 50-80%. Throughput degrades slightly for packets larger than the link MTU due to fragmentation; the impact of this effect is smaller for IPsec because IPsec processing dominates IP fragmentation processing.

Converting this graph from Mb/s to packets/sec confirms these conclusions. Figure 3 shows that non-IPsec packet rates are constant, halving when packets are fragmented, and that IPsec packet rates depend on packet size (as does linearity in the log plot of Figure 2), and so are CPU bound (they cannot be I/O bound because IP alone has higher throughput in all cases).

[2] For all graphs, a confidence interval of 95% is typically < ±2% of the plotted value, and so is not shown.

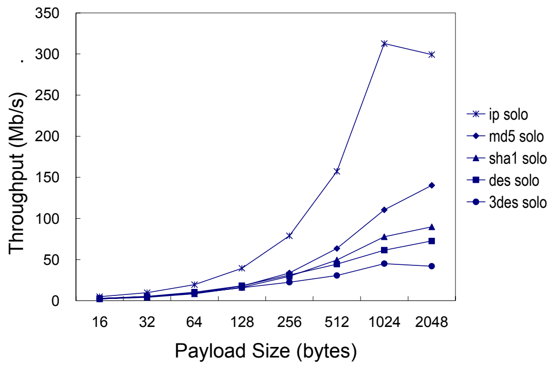

**Figure 1 Baseline IP and IPsec throughput.**

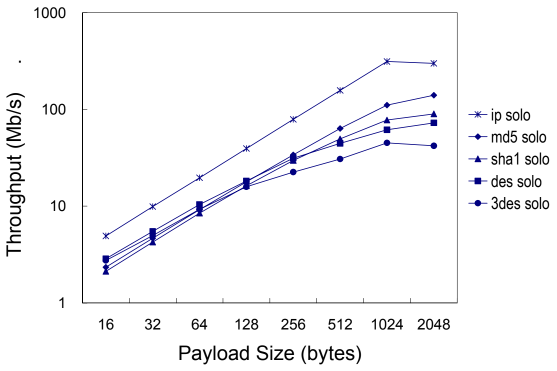

**Figure 2 Baseline using a log BW scale.**

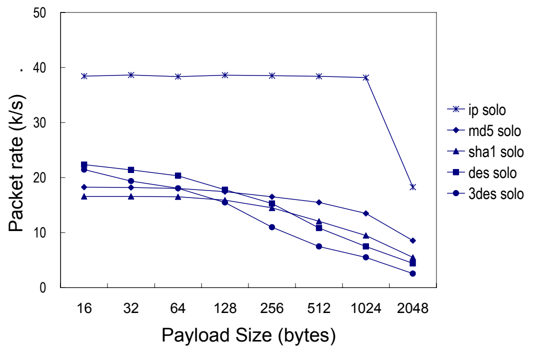

**Figure 3 Baseline IP and IPsec packets/sec.**

These graphs confirm other known results, *i.e.,* that HMAC-MD5 is faster than HMAC-SHA1, and that DES is faster than 3DES [14]. They also confirm that authentication is faster than (online) encryption. These graphs also demonstrate the substantial CPU resources consumed by a software implementation of IPsec, and thus the opportunity they provide to DoS attackers.

### 3.2. Impact of Non-Attack Cross-Traffic

Figure 4 compares the performance of an HMAC-MD5 IPsec authenticated transport mode stream between two hosts when there is no competing traffic (solo, top line) and when a non-attacking competing stream is present (md5 ok, bottom line). Under the cross traffic case, the receiver must validate twice as many packets using the appropriate cryptographic algorithm. This graph is shown with a linear bandwidth to make it easier to compare relative performance (other graph lines are discussed in the next section).

The throughput of each stream drops to half its solo value, not because the network is overloaded (from Figure 2, two HMAC-MD5 streams at the maximum host source capacity sum to less than 20% of the GigE link capacity, and less than 65% of the IP receive capacity) but because of the CPU capacity.

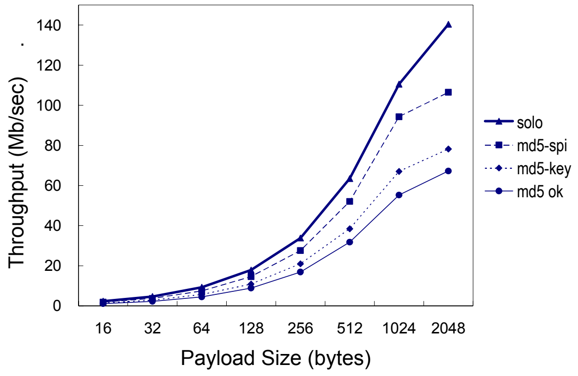

**Figure 4 Impact of cross traffic on an HMAC-MD5 authenticated stream.**

Figure 5 shows a similar result for a DES encrypted IPsec transport mode stream. Similar results were obtained for HMAC-SHA1 and 3DES.

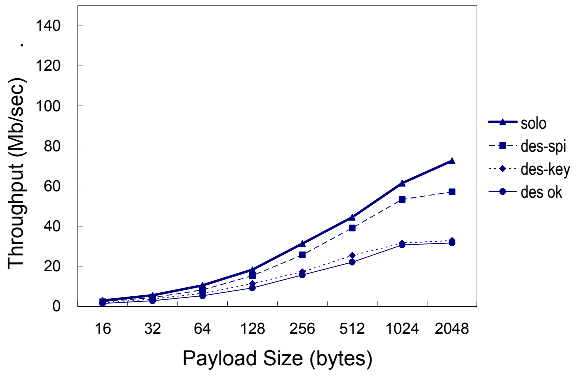

**Figure 5 Impact of cross traffic on a DES encrypted stream.**

## 3.3. Impact of Various Attack Cross-Traffic

Section 3.2 considered valid cross traffic; this section considers other traffic. An attacker may generate one of three kinds of IPsec attacks: "Alg-SPI", "Alg-key", and "Alg OK", where 'Alg' refers to the specific algorithm, *e.g.*, MD5 or DES, and 'SPI' and 'key' indicate what fails (incorrect SPI, incorrect key), or that ('OK'), nothing failed. These all assume 'spoofed' packets that correctly falsify addresses, as for services between well-known endpoints, *e.g.*, routing protocols or bulk exchange paths between proxies.

*Alg-SPI:* IPsec security associations and their packets are identified by a Security Parameters Index (SPI), an opaque ID indicating the algorithm used, active key, and other stream parameters [11]. The SPI is basically a nonce in the IP packet, visible to on-path attackers but hard for off-path attackers to guess because the SPI space is large (32 bits) and the number of active associations is typically small. Alg-SPI indicates that the attacker uses an invalid SPI and represents the least work an attacker can inflict.

*Alg-key*: Even when an attacker knows the SPI, *e.g.*, by on-path snooping, the cryptographic key is not typically known. Alg-key refers to the attacker knowing the SPI but not knowing the key. Keys are typically 128 bits or larger.

*Alg OK:* This case considers two streams coming to the receiver, or – equivalently –the attacker correctly guessing the SPI and key of a current session. This represents the most work an attacker inflicts.

All subsequent experiments here show attackers using the same algorithm as the valid traffic. Algorithm choice can affect differential (attack vs. legitimate) sender rates, to considered in future work. At the receiver, algorithm choice is irrelevant for Alg-SPI attacks because the packet is rejected before such processing. For Alg-key and Alg-OK attacks, the algorithm is fixed by the SPI at the receiver, *i.e.*, we consider an attack within an existing SPI. Attacks on different SPIs should already be controlled by per-SPI resource containment.

Figure 4 compares the impact of these various kinds of attack on an HMAC-MD5 session. Note that rejecting traffic due to a bad SPI has substantially less impact on the original stream (20%) than trying to validate incorrectly keyed packets (40%) or accepting correctly keyed packets (50%). Checking the SPI is required in all cases, and indexing the SPI in the SPD table is much faster than subsequent HMAC validation. In Alg OK, the packet is further handed to the socket.

The differential impact is more pronounced for more computationally intensive algorithms such as DES (Figure 5). Here the SPI lookup cost is unchanged but the cryptographic verification cost increases, so the relative impact of the two changes.

IPsec transport mode and IPsec tunnel mode were also compared to determine whether mode affects impact. Figure 6 shows the result for DES, in which mode effects are overshadowed by attack effects.

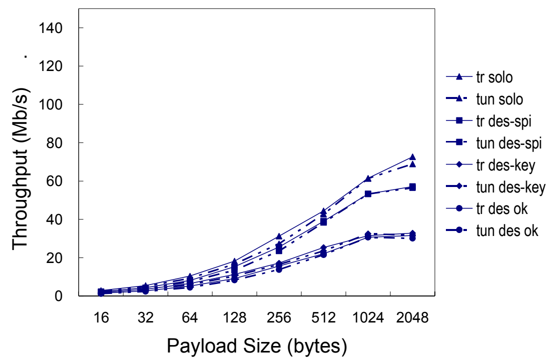

**Figure 6 IPsec DES transport *vs.* tunnel mode.**

## 4. Observations

These experiments confirm that IPsec can introduce an opportunity for CPU overload DoS attack. As shown in Figure 4 and Figure 5, when end-hosts perform software IPsec CPU can be overloaded when the attacker knows only the SPI. The DoS opportunity arises because creating attack packets – given a known SPI – is much less effort than the effort of creating valid IPsec packets (*e.g.,* arbitrary data suffices).

Valid traffic throughput under cross traffic with the wrong SPI (Alg-SPI) indicates that extra layers of filtering help even within a single packet by reducing impact of DoS attacks as early as possible. This may include checking not only the SPI, but also second nonce that could vary on different timescales than the SPI, which varies only on rekeying. That second nonce could be shorter (for faster checks) or longer (to support proxies, whose large numbers of security associations might have a densely populated SPIs).

Checking nonces is a comparatively low-effort test, especially *vs.* high-effort crypto algorithms. Static nonces protect only against off-path attacks and only if selected randomly because they are sent in the clear (unencrypted), and fail in off-path attacks when successfully guessed. The latter is rare because of the large nonce space (32 bits for the SPI, which acts like a nonce). Protection from on-path attackers and those able to guess SPIs might require 'spinning' nonces, using a pseudorandom sequences, hopping per-packet or (more efficiently) every few packets, *e.g.*, as suggested in FPAC, or even simpler [5]. An on-path attacker (or off-path successful guesser) would then have only a very short window in which it could emit attack packets with a valid nonce.

Different kinds of nonce 'spinning' algorithms would have different properties – *i.e.*, higher degrees of randomness or lower cost to compute. These algorithms might be layered to allow more than one nonce in each packet. This would allow further hierarchical application of effort to defense, and thus potentially further protect the receiver's CPU resource.

It might also be useful to consider nonce-based protections when used alone, not just in combination with current IPsec algorithms. The protection afforded would not be as strong, *e.g.,* as HMAC-MD5, but the performance penalty could be reduced. This could address the need for 'good enough' security.

## 5. Comparison of MAC Algorithms

SPI checks are much faster than current IPsec crypto algorithms, which is why it is useful to apply them as filters before investing in IPsec validation. Other, faster crypto algorithms could have similar utility. This section examines increases in cryptographic software performance by using optimized code and capitalizing on new features of common PC CPU architectures.

Crypto performance is important not only for mobile and embedded devices but also for end hosts lacking hardware support. One of the criteria for evaluating AES proposals was that the algorithm should perform well both in hardware (smart cards) and software (CPUs) [16]. Many alternative algorithms have been proposed to the popular HMAC-MD5 and HMAC-SHA1 ([2][3][8][20]). These newer MACs aim both to provide "proved" security and higher performance[3].

Original HMACs were designed as a more efficient alternative to block cipher-based CBC-MACs, because hashes (*e.g.*, MD5, SHA-1) perform much faster in software than block ciphers (DES, 3DES) [1][12]. UMAC and other newer algorithms use a universal hash function to "compress" the data before applying a block cipher (SHA1 initially, now AES) to secure the authentication with cryptographic strength [8][13]. This approach claims the benefit of reducing MAC security to that of the final block cipher.

HMAC uses a cryptographic hash function to compress the original message and UMAC 'folds' the data using a much faster 'universal hash' and then performs an AES hash on the result. UMAC's particular universal hash is designed to match SIMD-like support available on many general-purpose CPUs (notably the x86 MMX and SSE extensions).

[3] Keyed MACs MD5 and SHA-1 are known "broken" in principle if not practice, but both when HMAC'd are still considered useful.

## 5.1. Measurements

We first compare the software performance of three MAC algorithms: HMAC-MD5, HMAC-SHA1, and UMAC; we also measure the effect of optimization using x86/SSE2 assembly. The HMAC-MD5 C language implementation (hmac-md5c) is an optimized variant of RFC1321 [17][18]. Code for HMAC-MD5 and HMAC-SHA1 in i586 assembler is from OpenSSL v0.9.7d (hmac-md5a and hmac-sha1a). The UMAC implementation can be configured to compile either from C language source (umac-c) or using inline x86/SSE2 assembler (umac-sse2) [19]. Both HMAC-MD5 and UMAC here use 128-bit keys, whereas the HMAC-SHA1 uses a 160 bit key.

The tests were performed on both Pentium-4 (p4) and Athlon64 (k8) micro-architectures, both in 32-bit mode, and tested over message sizes ranging from 44 bytes to just over 2k bytes. Figure 7 shows the number of CPU cycles per byte of message authenticated for the various algorithms. HMAC-SHA1 in assembler (hmac-sha1a) has a very high number of cycles per byte, over 120 for small messages; it is clearly very inefficient compared to other algorithms, even when coded explicitly in x86 assembler. This may be a result of 32-bit modes on the p4 architectures.

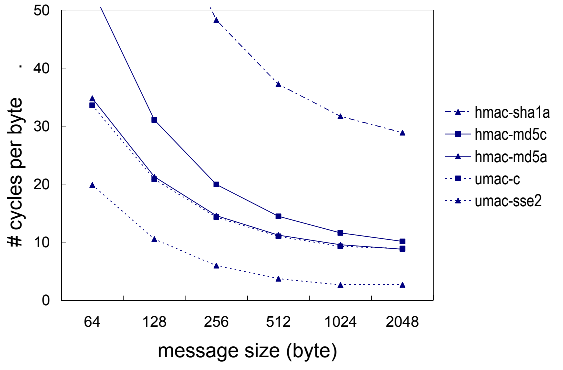

Figure 7 Clocks per byte for p4 hashes.

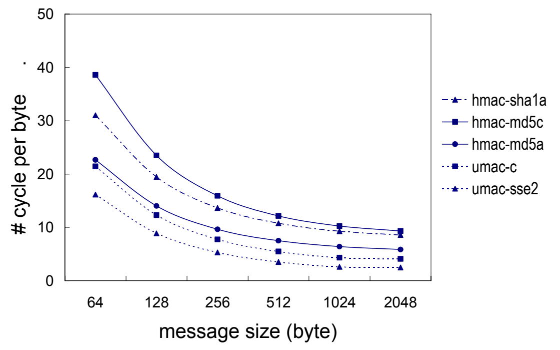

Figure 8 Clocks per byte for k8 hashes.

Note, however, that hmac-sha1a on the k8, even in 32-bit mode, performs much better, beating out hmac-md5c (Figure 8). Comparing Figure 7 and Figure 8, the relative performance of some algorithms change, but the overall effect is similar. The speedup relative to hmac-md5c is shown in Figure 9 for p4 and Figure 10 for k8; those graphs show that the overall speedup is at most a factor of 4x for umac-sse2, but that for other algorithms the speedup is limited to a factor of 2.

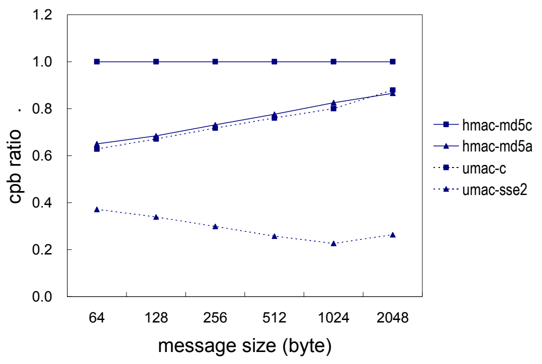

Figure 9 Speedup relative to MD5c on a p4.

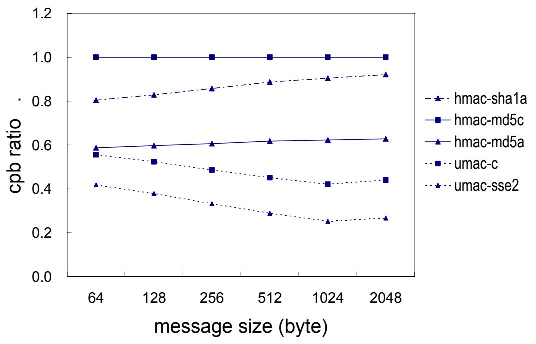

Figure 10 Speedup relative to MD5c on a k8.

## 5.2. Observations

These graphs indicate that all algorithms perform considerably worse for short message lengths due to algorithm initialization overhead and minimum crypto blocksizes, but that performance stabilizes for messages 512 bytes or larger; this means that small packets could incur higher performance penalty, i.e., that an attacker would be more effective if he sent a large number of short messages than a few long ones.

The graphs indicate that the increased efficiency of the UMAC algorithm is obtained for large messages (4x speedup), but that the impact is much smaller for shorter messages (2.5x speedup). For MD5, the benefit is converse, where assembler is faster than C code for smaller messages but not much for larger ones. Overall, the processor's micro-architecture can have great impact on performance. Notice how k8 runs hmac-sha1a and umac-c much faster than p4 – this is probably due to both SHA-1 and UMAC having high degree of ILP (Instruction-Level Parallelism) [4].

Using more efficient cryptographic algorithms and better optimized programs can improve IPsec performance by up to 4 times, it is not nearly enough to defend from such DoS attacks, since the attacker can always generate attacking traffic more "efficiently" by putting random bytes into the payload. Because the majority of IPsec processing power is spent on cryptographic operations, unless the speed of the authentication and encryption codes can match that of just IPsec tunnel processing and SPI filtering, the attackers will still have a big performance advantage than the legitimate users.

## 6. Conclusions

This paper examined the impact of DoS attacks on IPsec capacity, and demonstrated that a single attacker can degrade throughput as much as 50%. Off-path attackers have much less impact (20%) than on-path attackers because the nonce-like check of the IPsec SPI is much lower effort than validating a security signature, and the SPI is hard to guess when off-path. This suggests that a layered approach to individual IP packet security, using fast nonce checks, is useful in reducing the impact of attacks. It also suggests that nonces need to be very simple, and may need to be changed infrequently to be useful.

The paper also examined the impact of varying the security algorithm, both on the attack viability, and on overall security performance. The results suggest that more CPU-intensive algorithms benefit more from nonce-based layered security within a single packet, and that even modern high-performance algorithms may still need such measures.

### 6.1. Future Work

This work focused on the impact on existing IPsec associations; we plan to examine similar impact on IKE to determine if similar nonce systems reduce DoS impact. We plan to explore UMAC and AES in our IPsec measurements, SPI spinning using pseudorandom sequences, and the use of layered nonces for additional protection, including simpler variants of FPAC. Finally, we plan to measure the use of layered nonces without underlying strong cryptographic protection to determine the performance upper bounds.

## 7. References

[1] Bellare, M., Canetti, R., Krawczyk, H., "Message Authentication using Hash Functions – The HMAC Construction," *RSA Laboratories' Cryptobytes*, vol.2, no.1, Spring 1996.

[2] Bernstein D. "The Poly1305-AES Message Authentication Code," Proc. 12th Workshop on Fast Software Encryption, 2005.

[3] Black, J., Halevi, S., Krawczyk, H., Krovetz, T., Rogaway, P., "UMAC: Fast and secure message authentication," Proc. CRYPTO '99, pp. 216-233.

[4] Bosselaers, A., Govaerts, R., Vandewalle, J., "SHA: A Design for Parallel Architectures?" Proc. Eurocrypt '97, pp. 348-362.

[5] Calvert, K. Venkatraman, S. Griffioen, J., "FPAC: fast, fixed-cost authentication for access to reserved resources," Proc. IEEE INFOCOM 2002, pp. 1049-1058.

[6] Eastlake, D., Jones, P., "US Secure Hash Algorithm 1 (SHA1)," RFC 3174, September 2001.

[7] Elkeelany, O., Matalgah, M., Sheikh, K., Thaker, M., Chaudhry, G., Medhi, D., Qaddour, J. "Performance Analysis of IPsec Protocol: Encryption and Authentication," *IEEE Comm. Conf. (ICC) 2002*.

[8] Jaulmes, E., Lercier, R., "FRMAC, A Fast Randomized Message Authentication Code," *Cryptology ePrint Archive*, Report 2004/166.

[9] Karn, P, Simpson, W., "Photuris: Session-Key Management Protocol," RFC 2522, March 1999.

[10] Kaufman, C., Ed., "Internet Key Exchange (IKEv2) Protocol," RFC 4306, December 2005.

[11] Kent, S., Seo, K., "Security Architecture for the Internet Protocol," RFC 4301, December 2005.

[12] Krawczyk, H., Bellare, M., Canetti, R., "HMAC: Keyed-Hashing for Message Authentication," RFC 2104, February 1997.

[13] Krovetz, T., Ed., "UMAC: Message Authentication Code using Universal Hashing," RFC 4418, March 2006.

[14] Menasce, D., "Security Performance," *IEEE Internet Computing Magazine*, May-Jun 2003, pp. 84-87.

[15] Rivest, R., "The MD5 Message-Digest Algorithm," RFC 1321, April 1992.

[16] Schneier, B., Kelsey, J., Whiting, D., Wagner, D., Hall, C., "Performance Comparison of AES Submissions," *Proc. of 2nd AES Candidate Conference*, NIST, March 1999, pp. 15-34

[17] Touch, J. "Performance Analysis of MD5," Proc. ACM Sigcomm 1995, pp. 77-86.

[18] Touch, J., Optimized MD5 software, <ftp://ftp.isi.edu/pub/hpcc-papers/touch/md5-opt.tar>.

[19] UMAC web http://www.cs.ucdavis.edu/~rogaway/umac/

[20] Viega, J. and McGrew, D., "The Use of Galois/Counter Mode (GCM) in IPsec Encapsulating Security Payload (ESP)," RFC 4106, June 2005.