Trading Off Strength and Performance in Network Authentication: Experience with the ACSA Project[1]

Jamison M. Adcock, David M. Balenson, David W. Carman,
Michael Heyman, and Alan T. Sherman[2]
Cryptographic Technologies Group
NAI Labs, The Security Research Division
Network Associates, Inc.
{jamison_adcock, david_balenson, david_carman, michael_heyman, alan_sherman}@nai.com

Abstract

The Adaptive Cryptographically Synchronized Authentication (ACSA) Project offers a new approach to data authentication in networks by trading off authentication strength and performance. In ACSA, the communicants select among various authentication gears to balance their performance and security needs. These gears include three basic groups: (1) conventional mechanisms that are computationally intensive but considered highly secure; (2) higher-speed, lower-strength mechanisms including Universal Message Authentication Codes (UMACs) and our novel inner-function group (IFG) with bit scattering; and (3) Partial MACs (PMACs) that calculate the authentication tag on only a subset of the message. We are implementing a prototype ACSA System based on the popular IPsec protocols and are demonstrating its effectiveness on high-speed network applications.

1. Introduction

Conventional authentication mechanisms do not operate at speeds fast enough to meet the demands of ultra-fast networks [1][18][23]. This disparity presents a great challenge for high-speed applications that demand network authentication. Much work has focused on devising fast authentication algorithms. The *Adaptive Cryptographically Synchronized Authentication (ACSA)* Project [7] provides a new solution to this challenge by trading off authentication strength and performance to achieve data origin authentication and connectionless integrity.

1.1. Motivation and overview

Some of the applications that motivate our work include real-time high-speed video, high-performance distributed computing, high-speed distributed storage, and 3-D virtual reality. Such applications might run on conventional platforms with high-speed network devices (*e.g.* Gigabit Ethernet or ATM networks on Pentium class machines without cryptographic hardware), on special platforms with conventional cryptographic hardware, or on computationally limited single-processor devices. The ACSA Project offers a practical and flexible solution to high-speed network authentication that can be implemented in software or hardware.

Figure 1 illustrates hypothetical strength-performance tradeoffs that ACSA might achieve. For example, the user can select a lower-speed, high-strength authentication mechanism, a higher-speed, lower-strength mechanism, or a high-speed *Partial Message Authentication Code (PMAC)* that authenticates only a portion of the message. In Section 7.2, we present a concrete instantiation of Figure 1 showing the actual strength-performance levels achieved by our implementation of an ACSA prototype system.

Throughout, we use the term "network authentication" to mean data origin authentication and connectionless integrity, as defined in Internet standards [13]. This type of authentication is without regard to the ordering of the message in a stream of messages.

[1] This work was supported by the Defense Advanced Research Projects Agency (DARPA) under Air Force Research Laboratory (AFRL) Contract No. F30602-98-C-0215.

[2] Sherman is also with the Department of Computer Science and Electrical Engineering, University of Maryland, Baltimore County (UMBC).

By "authentication strength" we mean the computational difficulty of producing a forgery. By "performance" we mean computational complexity (CPU time, memory space). The ACSA project aims to lighten the processing loads of the sender and receiver by requiring them to spend less processor resources (especially CPU time) on data authentication.

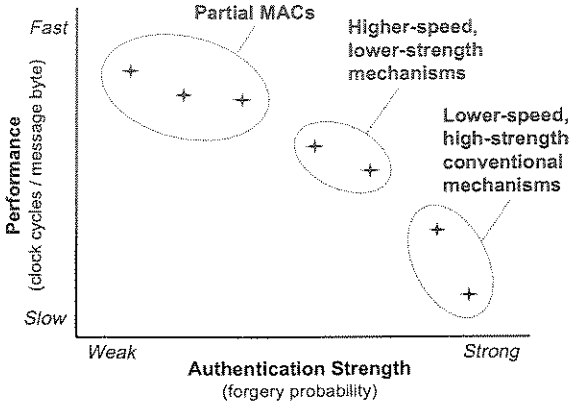

Figure 1. Hypothetical strength-performance tradeoffs for network authentication.

Most previous approaches to network authentication focused on providing high strength, with speed as a secondary concern. By contrast, we adopt the point of view that not all applications require high-strength authentication, and that faster authentication speeds can be achieved by accepting lower-strength (yet adequate) authentication methods. The ACSA System adaptively adjusts its authentication strength and speed to meet current needs based on security policy, observed authentication error rates, alarms from host or network defenses, and processor loading.

The ACSA System embodies a spectrum of authentication mechanisms—like gears of an automobile transmission—that provide various strength-performance levels of network authentication. These authentication gears are organized in the following three classes:

- lower-speed, high-strength conventional mechanisms;
- higher-speed, lower-strength mechanisms; and
- PMACs that authenticate only a portion of the message.

The lower-speed, high-strength mechanisms include algorithms such as HMAC-MD5. The higher-speed, lower-strength ACSA gears include other Message Authentication Codes (MACs) and our novel inner-function groups (IFGs) with bit scattering.

A significant benefit of the ACSA approach is that it supports communicants with the option of using *multiple authentication tags*. For example, with little overhead, the sender can compute two or more authentication tags with different strength-performance levels for the same message packet. Then, the receiver—independently from the sender—could in real time select which tag to verify. This flexibility is useful in unicast and multicast environments when the receiver cannot predict future processor loads, and in multicast environments where there may be many receivers with different processor capabilities.

To demonstrate the strength-performance levels achievable by the ACSA approach, we designed and implemented a prototype ACSA System. We based this prototype on the Internet Protocol Security (IPsec) protocols [13] and Internet Key Exchange (IKE—formerly ISAKMP/Oakley) protocol [11] because they are commonly used and provide the type of connectionless, data-origin authentication that ACSA supports.

Our prototype implementation illustrates the effectiveness of this approach to the IPsec architecture on two 400 MHz Pentium II computers operating Linux connected via a 100 MBit Ethernet network. This implementation is built from our ACSA developer's toolkit, which we are freely distributing.

In comparison with the high-strength HMAC-MD5-96, our prototype achieves speedup factors of greater than a factor of 10 for the higher-speed, lower-strength UMAC-MMX-15, and greater than a factor of 100 for PMAC-64 (using UMAC-MMX-15 as the underlying MAC algorithm). These approximate speedup factors are based on empirical measurements, as described in Section 7.

The ACSA approach is suitable for a wide range of high-speed applications. It is especially suitable for long packets such as video streams in applications that can tolerate lower levels of authentication strength. This approach, however, is not appropriate for all applications—for example, we do not recommend it for financial applications such as banking transactions requiring high levels of authentication security.

1.2. Alternative approaches

Previous work on the challenge of providing high-speed cryptographic network authentication has followed four basic approaches: First, engineers have provided separate special-purpose accelerated cryptographic hardware [1]. Despite the attractiveness of dedicated cryptographic hardware, such hardware is often expensive, supports only a limited set of algorithms and

platforms, and is soon rendered obsolete by software operating on rapidly advancing general-purpose CPUs.

Second, software engineers have attempted to speed up algorithms with parallel implementations (for example, see Nahum *et al.* [18]). Whatever gains can be achieved by this strategy can also be applied within and independently of ACSA.

Third, cryptographers have designed fast authentication mechanisms. For example, there have been significant recent advances with the Bucket [21][19], MMH [10], NMH [10], and UMAC [4] algorithms. But, if a communication system depends only on one such fast mechanism, then the system is inflexible and cannot adapt to changing needs. Also, existing communication systems alone cannot easily change the current authentication mechanism in use without a costly renegotiation of the established security association.

Fourth, some researchers have attempted to increase overall security performance by combining confidentiality and authentication functions in a dual-purpose algorithm [9]. Although it can make sense to compute, for example, AES and AES-MAC in a single dovetailed fashion, dual-purpose confidentiality and authentication algorithms risk security weaknesses and forsake the flexible engineering benefits of using separate interchangeable components.

The ACSA project distinguishes itself by taking a broad and flexible engineering approach to providing fast network authentication. Although the ACSA approach can be sped up with hardware, parallelism, and faster algorithms, its gains go beyond these optimizations through offering a fundamentally different strategy of adaptively trading authentication strength for speed.

1.3. Outline

The rest of this paper is organized in seven sections. Section 2 describes the ACSA system architecture, including its components, gears, and adaptive control. Section 3 explains the particular authentication mechanisms used within ACSA—namely conventional MACs, inner-function groups, UMACs, and PMACs. Section 4 analyzes the security of ACSA. Section 5 explains the application of ACSA to IPsec. Section 6 overviews our prototype implementation, including our developer's toolkit. Section 7 reports on our experiences with our prototype implementation, concretely analyzing the system's performance and illustrating achievable risk-performance tradeoffs. Section 8 discusses issues raised by ACSA and summarizes our conclusions. In addition, Appendix A gives pseudocode for our bit-scattering algorithm.

2. Architecture

The ACSA System [2] provides network authentication between a sender and a receiver. For each message (*i.e.* packet), the sender computes an *authentication tag*, which is appended to the message in the data channel. The receiver verifies the received message by recomputing the authentication tag and comparing it with the received tag. If the recomputed and received tags match, the receiver deems the message authentic.

In addition to the data channel, there is also a control channel between the sender and receiver. Using this control channel, the sender and receiver establish a set of *security associations*. These security associations specify a suite of authentication gears and shared authentication keys. Periodically, the sender and receiver exchange control information that can influence dynamic gear changes. All control messages are protected using standard confidentiality and authentication techniques.

2.1. Goals and requirements

The goals of the ACSA System are to make available a spectrum of network authentication mechanisms with various strength-performance tradeoffs, and to support a method for selecting when to apply these mechanisms. In addition, we impose the following two engineering requirements: the ACSA System must be able to operate on a wide range of network devices (*e.g.* firewalls, security gateways, and routers); and the architecture must be consistent with the IPsec and IKE protocols.

2.2. System components

As shown in Figure 2, the ACSA System is organized into five major modules: ACSA controller; local security and resource managers; security association and key management; network application; and security services. Figure 2 shows this view of the ACSA System for the sender; the receiver has a symmetrical organization. In this figure, arrows represent flow of application data and control information.

The main purpose of the controller is to determine which authentication gear to use. In making this decision, the controller relies on inputs from the local security and resource managers and from the network application. Additionally, when the system is performing the sender role for a given connection, the controller may receive inputs from the receiver regarding authentication errors and the receiver's processor load.

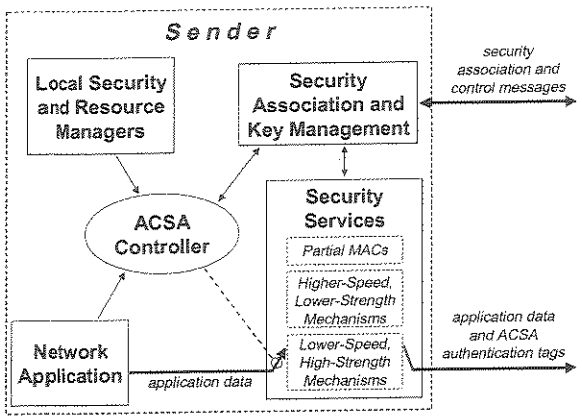

Figure 2. ACSA architecture.

The local security and resource managers represent components that may be provided by a host system. These modules offer information such as security and resource policy, processor load information, and possibly alarms from host or network defenses.

The main purpose of the security association and key management module is to establish and maintain security associations and cryptographic keys between communicants. ACSA further requires that the security association and key management module establish and maintain a suite of security associations that correspond to the ACSA gears supported for a given connection. Control messages are also sent from the receiver to the sender via the security association and key management module. In our prototype implementation, these security associations are negotiated using IKE [6].

In a system that supports ACSA, the network application may communicate to the controller ACSA-specific security and resource policy for its connection.

The main purpose of the security services module in ACSA is to perform the authentication tag computation for the gear specified by the ACSA controller. In the prototype, the IPsec module will perform all the functions of the ACSA security services module. When operating in receiver mode, the security services will also detect and report received authentication errors to the controller.

2.3. Gears, synchronization, and adaptive control

Initially, the sender and receiver agree on a set of acceptable authentication gears to use. Specifically, for consistency with IPsec protocols, for each connection, the sender and receiver initially negotiate a separate security association for each possible gear. To change gears, the controller within the sender system then selects among these agreed upon security associations based on security policy and processor load. The controller within the receiver system can effect gear changes by notifying the sender of observed authentication errors and processor resource limitations in the receiver for the current gear.

For consistency with IPsec, for each packet in our prototype system, the Security Parameter Index (SPI) [12] in the packet header specifies which gear was used to authenticate the packet. By specifying the current gear in the SPI of each packet, no other out-of-band mechanism is needed to achieve gear synchronization between the sender and receiver.

Recovery from dropped packets is performed at the application layer and is not independently guaranteed by the ACSA System. Dropping packets will not cause loss of cryptographic synchronization since the authentication gear and any derived secondary keys (see Section 2.4) depend only on the security association and the packet header.

2.4. Key management

ACSA needs key material for keyed conventional authentication mechanisms, for PMACs, and for our bit-scattering process.

During the negotiation of each connection, the ACSA System relies on the underlying network security protocol (*e.g.* IKE) to establish a common key between the communicants. This key, which we call the *primary authentication key*, is assumed to be valid for the entire lifetime of the connection. Conventional authentication mechanisms use the primary authentication key directly to compute the authentication tag. In our prototype implementation, each primary key is at least 128 bits long.

To meet the needs of additional functionality offered by ACSA, the ACSA security association and key management module derives *secondary authentication keys* from the primary authentication key using a deterministic one-way function. For example, our prototype system derives secondary authentication keys using the SHA-1 algorithm using techniques similar to those described by Matthews [17].

When performing the bit-scattering operation of IFGs or the word-selecting operation of PMACs, the ACSA System derives a secondary authentication key for each packet. By using a shorter-lifetime *secondary authentication key* rather than the longer-lifetime primary authentication key in these instances, we limit loss of any compromised keys due to any use of lower-strength authentication mechanisms.

It is important that these secondary keys be unique for different packets within a security association. Our prototype implementation achieves this objective by

seeding the generation of each secondary key for a packet with both the primary key (of the security association) and with the unique packet sequence number available in IPsec to prevent replay attacks.

3. Authentication mechanisms

The authentication gears of the ACSA System are a set of authentication mechanisms that achieve a wide spectrum of strength-performance levels. They are organized in three basic classes: (1) conventional mechanisms, including lower-speed, high-strength algorithms; (2) higher-speed, lower-strength MACs; and (3) Partial MACs (PMACs). By appropriate choice of their operational parameters, PMACs can run at a variety of strength-performance levels, including ultra-fast, lower-strength levels. This section describes these mechanisms, together with ACSA's use of multiple authentication tags.

3.1. Conventional authentication mechanisms

The lower-speed, high-assurance gears of the ACSA System include conventional authentication mechanisms, such as HMACs [3][14]. Our prototype implementation supports the following two IPsec mechanisms: HMAC-MD5-96 [15] and HMAC-SHA-1-96 [16].

These HMACs infuse a key k into a message x by the nested MAC (NMAC) rule:

$$\mathrm{NMAC}_k(x) = f_{k1}(F_{k2}(x)), \qquad (1)$$

where $k1$ and $k2$ are subkeys derived from k; f is a keyed "outer" function; and F is a keyed iterated "inner" function. For the conventional IPsec mechanisms above, both f and F are based on the same underlying hash function (*e.g.* SHA-1).

This NMAC construction enjoys some desirable security properties, as explained by Bellare *et al.* [3]: under suitable security assumptions, the NMAC is at least approximately as strong as the inner function. We also use this NMAC construction in the other gear classes.

3.2. Higher-Speed, lower-strength MACs

Currently, the ACSA System includes two types of higher-speed, lower-strength MACs: inner-function groups (IFGs) and UMACs.

Both IFGs and UMACs gain speed within the NMAC construction by using a higher-speed, lower-strength iterated inner function together with a slower-speed, high-strength outer function. Because the inner function is applied many times over a long message, and the outer function is applied only once, this design offers significant speed-ups over conventional MACs. As pointed out by Bellare, Canetti, and Krawcyzk [3], the security assumptions on the inner function can be significantly relaxed while still maintaining a high level of security for the entire NMAC. The application of this observation in creating higher-speed, lower-strength NMACs is a significant contribution of the ACSA project.

We intentionally use the terms "inner function" and "outer function" to avoid unwanted security connotations associated with the terms "MAC" and "cryptographic hash function." In particular, in an ACSA-style NMAC, the inner function needs to satisfy only a weak collision-resistance property (weaker than that typically required of a MAC), and the outer function might have to satisfy pseudorandomness security properties (stronger than those typically required of a MAC).

Following this NMAC design principle, we developed IFGs with bit scattering. Later, after learning about the recent work of Black *et al.* [4], we also incorporated the UMAC mechanism into ACSA. Although UMACs do not use bit scattering, both UMACs and IFGs exploit the NMAC construction for greater speed. Both UMACs and IFGs are related to the universal hash function paradigm of Wegman and Carter [24]. Wegman and Carter, however, invented their paradigm for strength; we exploit their paradigm for speed and strength.

3.2.1. Inner-function groups with bit scattering. IFGs are based on nested MACs [3] and consist of two or more faster-speed, lower-strength inner functions that feed a high-strength outer function. Additionally, IFGs contain a *bit scattering* function that pseudorandomly selects which bits of the message feed into which inner functions.

The main cryptographic purpose of bit scattering is to prevent an adversary from knowing which parts of the message are processed by which inner functions. Even if an adversary could create collisions in one or more of the lower-strength inner functions when attacked in isolation, the adversary would have difficulty mounting an attack against the IFG because she would not know the inputs to the inner functions. The main performance advantage of IFGs stems from being able to use faster-speed inner functions.

As shown in Figure 3, the sender creates an IFG authentication tag in three steps: bit scattering; compression with inner functions; and outer function computation.

To perform bit scattering, the sender pseudorandomly throws bits of the message in one of two or more temporary buffers called *bins*. Bit scattering is simply a data-moving operation that performs no compression. To enable the sender and receiver to synchronize

cryptographically, a function based on a secondary authentication key controls the pseudorandom scattering process.

To scatter bits, ACSA uses the following algorithm. First, the sender pseudorandomly creates a bit mask as long as the longest packet size to be supported. Second, the sender pseudorandomly selects a starting point (which we call a *mask offset*) within the mask. Third, processing pairs of data words at a time, the sender logically ANDs and ORs the data words with the corresponding mask words (and their bitwise complements) to select and interleave bits for the two bins (see Appendix A).

To save computation time, the mask can be reused within a security association. Our prototype generates one mask (as long as the maximum packet size) for each security association. This mask is created by expanding a secondary authentication key in a manner similar to the key expansion algorithm used in IKE [11]. The sender and receiver then use per-packet secondary authentication keys to compute the mask offsets for each packet.

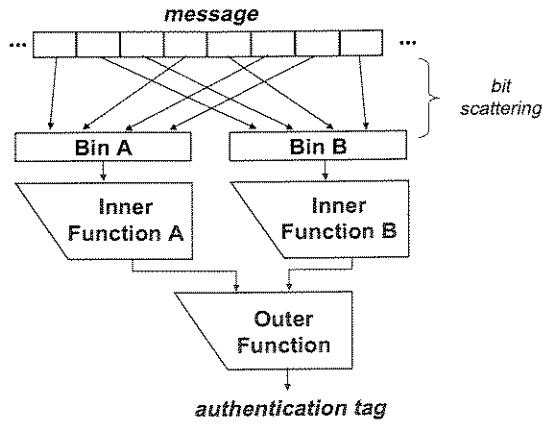

Figure 3. An ACSA NMAC realized by a two-element inner-function group with bit scattering.

Although conceptually simple, scattering bits into bins is nontrivial to perform at high speeds. On a Pentium II, our bit-scattering algorithm runs in less than 1.5 clock cycles per byte. The time to scatter bits is a significant cost of the IFG technique.

After performing bit scattering, a separate inner function is applied to each of the bins. Each inner function is keyed using a secondary authentication key. Candidate inner functions include MD4, AHA [20], MMH [10], NMH [10], Square Hash [8], Bucket [21], Evaluation Hash [22], and Division Hash [22].

The outputs of the inner functions are concatenated and fed into a high-strength outer pseudorandom function (PRF) such as SHA-1 or RC6 [4]. The PRF is keyed using the secondary authentication key as discussed in Section 2.4. The PRF output constitutes the IFG authentication tag.

3.2.2. UMACs. UMAC is a new fast message authentication algorithm by Black *et al.* [4] that runs an order of magnitude faster than high-strength conventional mechanisms such as HMAC-SHA-1. It is based on a new universal family of hash functions called *NH (nonlinear hash),* and the inventors of UMAC have proven security bounds (probability of forgery) of UMAC based on assumed bounds of NH. UMACs fit in very nicely with ACSA as plug-in higher-speed, lower-strength gears.

UMAC follows a slight modification of the aforementioned NMAC formula:

$$\mathrm{UMAC}_k(x) = f_{k1}(\, s, \mathrm{NH}_{k2}(x)\,), \qquad (2)$$

where s is a nonce provided by the sender. Thus, as do IFGs, UMAC achieves fast speeds by using a higher-speed, lower-strength inner hash. Unlike the Bucket, MMH, and NMH hash function descriptions, the UMAC paper [4] provides a complete MAC specification.

The NH function computes a tree of multiplications with increasing word sizes. NH is parallelizable and by design achieves a high performance on MMX™-enabled processors. Black *et al.* propose a family of UMACs of different strengths by truncating the NH tree of multiplications at different heights (word sizes). Although the inventors did not explicitly say so in their paper, their family is an embodiment of an authentication strength-performance tradeoff.

The ACSA System applies a set of UMACs covering a range of strength-performance levels.

3.3. Partial MACs

ACSA uses the novel concept of a *Partial MAC (PMAC)* to achieve even faster authentication speeds than those attained by UMACs and by IFGs. In a PMAC, a UMAC or higher-speed, lower-strength HMAC is applied only to some portion of the message body. Because the message header contains critical information, ACSA always authenticates the entire message header (alternatively, one might always authenticate each header with strong authentication). Care must be exercised when selecting the PMAC gear: PMACs are appropriate only for applications that can tolerate some modified bits in the message body.

We may view each message as a header followed by a sequence of message body parts (*e.g.* 64-bit words). Each PMAC has an associated parameter that specifies with what probability each message body part will be authenticated. We call this parameter the *message-*

selection percentage. Although there are several different ways in which PMACs can be realized, the main effect is the same: applying an authentication algorithm only to selected parts of the message can save significant time.

Abstractly, it is appealing to think of PMACs as follows. For each message body part, the sender tosses a biased coin to decide if that part is authenticated. On Heads, the sender includes that part in the computation of the authentication tag. On Tails, the sender does not include the part in the computation.

The coin flips are generated in a deterministic pseudorandom fashion using a secondary authentication key derived from the primary authentication key. This way, the receiver can synchronize cryptographically with the sender to know what message body parts to include in the authentication tag computation.

PMACs use a very fast inner function (*e.g.* NH) to perform compression on the selected parts of the message. As with UMAC or IFGs, the outer function is still a high-strength conventional PRF. In using the NMAC construction, we do not have to generate any fake authentication tags that might otherwise have been needed.

In our prototype system, we pseudorandomly select words of the message body to authenticate as follows. We start at the beginning. The next word to be selected is determined by a pseudorandomly-chosen offset from the currently chosen word, uniformly chosen from the interval $[1, 2L]$. Here, $L = 1/p$ (rounded to next highest perfect power of 2), where p is the message-selection percentage.

When computing PMACs, our prototype selects 64-bit words and uses UMAC-MMX-15 as the underlying authentication algorithm. We chose to operate on 64-bit words to exploit the Pentium II processor's efficiency at handling that data size.

Finally, note that cyclic redundancy codes are not promising inner-function candidates because they are slower and much weaker than UMAC-MMX-15. For example, we have estimated that CRC-32 executes at approximately one cycle per byte on the Pentium II, at least twice as slow as UMAC-MMX-15.

3.4. Multiple authentication tags

The ACSA System allows and supports the use of *multiple authentication tags*. That is, the sender can compute two or more different authentication tags for the same packet, reflecting different strength-performance levels. This concept can be realized in a variety of ways; its implementation is especially efficient for UMACs and IFGs.

In our prototype implementation, the sender can compute multiple authentication tags for IFGs with relatively little overhead. For example, if an IFG has two inner functions, A and B, then a total of three authentication tags can be separately computed for A, B, and A and B combined, with only three applications of the (non-iterated) outer function. A similar but different strategy could also be applied with UMACs and PMACs.

Multiple authentication tags afford the receiver with increased flexibility in choosing strength-performance levels adaptively. For example, in unicast or multicast communications, the receiver could choose which tag to use in real time based in part on local processor load.

Multiple authentication tags are also useful in multicast environments where different receivers may have different processor capabilities or security policies, and where the sender typically has more powerful computing capabilities than do the receivers. We are not aware of any previous suggestion to use multiple authentication tags in this fashion.

To ameliorate possible security vulnerabilities caused by making multiple authentication tags of the same message available to the adversary, the ACSA System uses a separate secondary authentication key to compute each multiple tag.

4. ACSA vulnerabilities and defenses

The goal of an adversary is to tamper with one or more data words of a packet without detection. Such tampering may include modifying, deleting, inserting, padding, permuting, or replaying bits or data words within a packet or in an authentication tag. An adversary might try attacking the ACSA System by attacking its control mechanisms and by attacking its component functions.

The strength of the system can be analyzed by considering both the computational difficulty of forging an authentication tag, and the probability of modifying a message without detection.

4.1. System attacks and defenses

To attack or influence the ACSA control mechanisms, an adversary might try to spoof, modify, or replay control information. By using strong confidentiality, authentication, and anti-replay services from the underlying network security protocol, the ACSA System protects against such attacks.

An adversary can, nevertheless, block control information. Such interference can result in degradation of service, but not loss of authentication security. The ACSA System does not try to prevent denial-of-service

attacks, which are existing vulnerabilities of IPsec and IKE.

A more subtle attack is to modify the communicant's environment, to cause the ACSA controller to use a weaker gear than it might otherwise use. For example, if the adversary could cause the sender's processor load to increase significantly, then the sender may switch to a weaker and faster gear. This vulnerability cannot be solved within ACSA. Each ACSA platform should protect its resources from tampering, not use resource loads to determine ACSA gears, or set an acceptable minimum strength gear.

In our original design of ACSA, for added protection, we had planned to keep the gear choice secret from the adversary. To comply with IPsec protocols, however, the gear selection in our prototype implementation is communicated as part of the SPI, which is authenticated but sent in the clear in each packet header. Fortunately, when IFGs are used, knowing the gear does not enable the adversary to determine which data bits and words contribute to which inner functions. Similarly, knowing the PMAC gear does not enable the adversary to determine which data words contribute to the authentication tag computation.

Because ACSA computes a single tag (or single set of tags in the case of multiple authentication tags) for each packet, it is not possible to use external timing information to determine which data words contribute to which inner functions, nor to determine which words are selected for partial authentication. Timing information can, however, reveal gear choices and the proportional mix of inner functions within an IFG.

4.2. Authentication mechanism security

For authentication mechanisms, the basic measure of security we consider is probability of forgery under the best possible attack. In this section, we estimate and bound the forgery probabilities for authentication mechanisms in each of the three major gear classes by reviewing the best known attacks on these mechanisms.

We also note that the security of bit scattering and PMACs depends crucially on the unpredictability of the secondary authentication keys. These keys are protected through the strong pseudorandom functions that generate them, and through the inclusion of the packet sequence number in their generation.

4.2.1. Conventional mechanisms. The probability of forgery for HMAC-SHA-1 and HMAC-MD5 is directly related to the keyed collision resistance of the SHA-1 and MD5 algorithms, respectively [3]. The best known keyless collision attacks on the underlying hash functions are birthday attacks, so the best known attack on these HMACs has a forgery probability of less than 2^{-64}.

4.2.2. Inner-Function groups with bit scattering. The security of IFGs depends on the security of the inner functions, the security of the outer function, and the effect of bit scattering. Since ACSA uses IFGs with a high-strength outer function and with higher-speed, lower-strength inner functions, for the purpose of the analysis, we may assume that the security of IFGs depends primarily on the inner functions and bit scattering.

Consider an IFG containing two inner functions A and B. Without bit scattering, the adversary would know which message words fed into A and which words fed into B. As a result, the forgery probability of the ensemble would be at least that of the weaker of the two inner hashes. Therefore, bit scattering is a crucial security-enhancing operation for IFGs.

When an IFG employs bit scattering, an adversary has two possible attacks: successfully guess which message bits are processed by which inner functions and attack one of the inner functions; or construct an attack that succeeds regardless of how the message bits are scattered. Since the later possibility seems implausible, it appears that to attack an IFG with bit scattering requires the adversary to guess how the bits to be modified are scattered.

Let n be a typical message length (in bits), and suppose that approximately k of the message bits are scattered into inner hash function A. Suppose further that the adversary wishes to modify t message bits. The chance of correctly naively guessing how these t message bits were scattered is:

$$(k / n)^t. \qquad (3)$$

In this sense, the bit-scattering operation appears to enhance authentication strength greatly. We assume that the adversary knows n and k, since these values could be approximated from network and timing analysis. Finally note, that from Equation 2, the scattering process creates significantly greater uncertainty for the adversary when carried out on a bit-by-bit basis rather than on words.

An adversary might try chosen-plaintext attacks in which she observes several message-authentication tag pairs computed from a common mask. However, the per-packet mask offsets, the strong outer function, and the per-packet rekeying of the outer function deter an adversary from garnering information about the mask.

4.2.3. UMACs. Black *et al.* [4] and Rogaway[3] describe forgery probabilities for various UMACs. We summarize these probabilities in Table 1.

Table 1. UMAC probability of forgery.

UMAC version	Probability of Forgery
UMAC-MMX-60	2^{-60}
UMAC-MMX-45	2^{-45}
UMAC-MMX-30	2^{-30}
UMAC-MMX-15	2^{-15}

4.2.4. PMACs. The probability of forgery of a PMAC is based overwhelmingly—not on the strength of the underlying MAC—but on its message-selection percentage. For PMACs with very small message-selection percentages, the probability of forgery is very high (nearly unity).

The utility of PMACs stems not from their ability to detect short modifications of a message; their utility arises from their ability to detect the modification of many bits or words. To this end, consider the probability Ψ_t of successfully modifying at least t message words. This probability is:

$$\Psi_t = (1-p)^t, \qquad (4)$$

where p is the message-selection percentage.

Evaluating PMAC security in the context of the number of modified data words is useful for many high-speed applications (*e.g.* high-speed video) that can tolerate some data modifications without a significant loss of fidelity.

5. Application to IPsec

ACSA is applicable to the common network security IPsec protocols. Specifically, ACSA can enhance network authentication service in IPsec by adding higher-speed, lower-strength mechanisms, and by providing an adaptive control system that adaptively switches among IPsec authentication transforms.

Currently, IPsec uses only lower-speed, high-strength authentication algorithms. The two mandatory-to-implement authentication algorithms HMAC-SHA-1 and HMAC-MD5 are so computationally intensive that they must be implemented in special-purpose hardware to support very high-speed network applications. IPsec also specifies HMAC-TIGER as an optional alternative. HMAC-TIGER, however, does not provide significant performance improvement over HMAC-SHA-1 and HMAC-MD5, except on 64-bit processor platforms. Therefore, the higher-speed, lower-strength mechanisms from ACSA offer attractive options for speeding up network authentication in IPsec.

The ACSA control system can be applied in IPsec to select authentication transforms adaptively based on current security policy, detected authentication errors, and processor load. Using the IKE protocol, ACSA initially determines a suite of authentication gears common to the communicants (see Section 2.3). This system of adaptive control gives IPsec a practical strategy for adaptively choosing from among a wider range of strength-performance levels.

6. Prototype system: Design, implementation, and toolkit

The main goals of the ACSA prototype are to demonstrate the feasibility of the ACSA model in enhancing network authentication performance, and to develop a freely available developer's software toolkit that performs and allows others to experiment with and use ACSA functions. At its highest level, the ACSA prototype software comprises an ACSA developer's toolkit, processor-specific optimizations, prototype demonstration software, and third-party network security software.

The ACSA developer's toolkit contains the core functionality that every ACSA implementation must perform. Primarily, the toolkit provides functionality for the controller module. The ACSA developer's toolkit also contains additions designed to integrate easily within most IKE and IPsec implementations.

The ACSA prototype contains processor-specific optimizations for the various supported authentication mechanisms. Optimization of conventional algorithms is important to provide an accurate baseline of the achievable authentication performance of current implementations. Optimization of the proposed authentication mechanisms such as UMACs and PMACs is necessary to demonstrate the performance enhancement that ACSA is capable of providing.

Demonstration software and third-party network security software is also included in the ACSA prototype to provide a fully operational system capable of showing the advantages of ACSA. The third-party software performs the IPsec and IKE protocols. The target platforms for the ACSA prototype are two 400 MHz Pentium II computers operating Linux connected via a 100Mbit Ethernet network connection.

In considering the costs of integrating ACSA, it is important to consider the development cost of adding ACSA to an existing IPsec implementation. Although ACSA controller software is in a self-contained module, significant modifications are required of the existing

[3] Phil Rogaway, personal correspondence (May 4, 1999).

IPsec module to perform ACSA. Specifically, new higher-speed, lower-strength authentication mechanisms must be added, and the ability to handle multiple security associations for a single connection is needed. Modifications to the existing IPsec module can be costly since changes may be required to the operating system's kernel code, or to hardware or firmware in an IPsec co-processor. Modifications to support ACSA must also be made to the IKE module, including additions to interact with the ACSA controller, to negotiate multiple security associations per connection, and to communicate ACSA control messages.

7. System performance

The computational performance of the ACSA system is almost entirely dependent on the performance of its authentication mechanisms. The administrative overhead establishing a gear suite, retrieving local processor load and security condition information, deriving keying material, and determining the appropriate gear is relatively small when amortized over all computations required to create or verify authentication tags for a high-speed network connection.

We give performance measurements for the Pentium II for two reasons. First, in order to give any meaningful detailed high-performance measurements, it is necessary to optimize implementations for some particular processor. Second, we implemented our prototype system on the commonly used Pentium II. Our Pentium II benchmarks calibrate our performance results.

The following two subsections discuss the performance of our prototype implementation on the Pentium II, focusing on the computational costs of performing selected authentication mechanisms and on the achieved strength-performance levels.

7.1. Analysis of system performance

This section describes the performance of each of the three major gear classes in our prototype implementation.

7.1.1. Conventional algorithms. For the large packet sizes that demonstrate the greatest advantages of ACSA, the computational performance of the HMAC-SHA-1 or

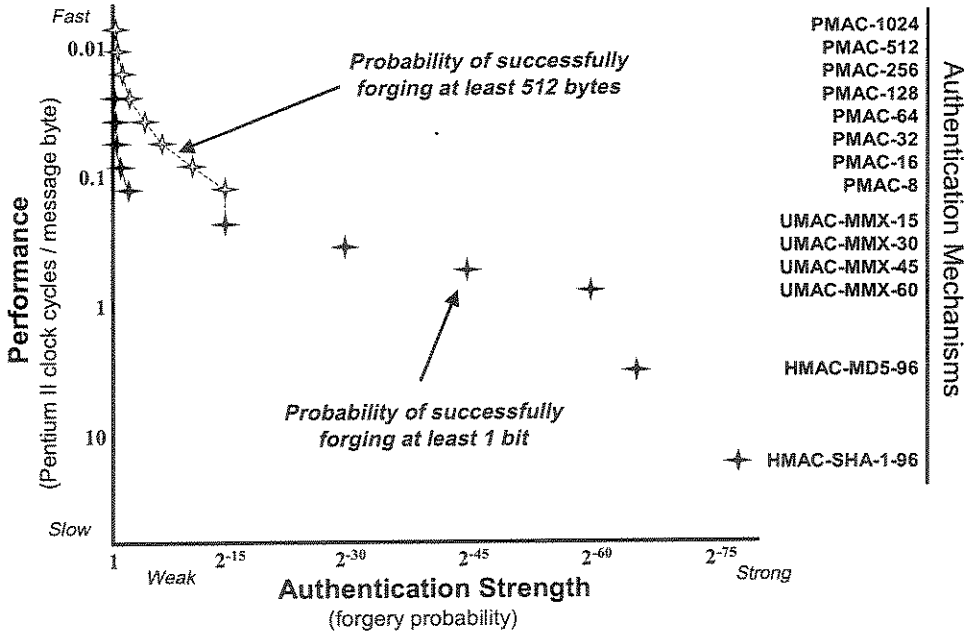

Figure 4. Strength-performance tradeoff measurements for our prototype implementation of the ACSA system.

HMAC-MD5 algorithm is almost entirely dependent on the performance of the underlying SHA-1 or MD5 algorithms, respectively. The fastest assembly language implementation of HMAC-SHA-1 we encountered runs at 12.6 clock cycles per message byte, according to Black *et al.* [4].

As for MD5, Bossalaers [5] reports that his optimized version on the Pentium runs at 5.3 cycles/byte. Since the Pentium II offers increased computational efficiency over the Pentium for most operations, we expect an optimized version of MD5 on the Pentium II to perform even faster.

7.1.2. Higher-Speed, lower-strength algorithms. Since the ACSA prototype will use the Pentium II processor, we may take advantage of the dramatic performance of the UMAC-MMX family of algorithms that use MMX functionality for much of their computations. Black *et al.* [4] report that their implementations of UMAC-MMX-60 and UMAC-MMX-30 run in 0.98 and 0.51 cycles/byte, respectively. Ted Krovetz reports that his implementation of UMAC-MMX-15 runs in 0.32 cycles/byte.[4] These measurements were taken on message data that reside in the Pentium II L1 cache.

Thus, when using an IFG with two separately-keyed inner UMAC-MMX-15 functions, the total speed of IFGs with bit scattering is approximately 1.5 + 2 * (0.32 * ½) = 1.82 cycles/byte on an MMX-enabled Pentium II.

7.1.3. Partial MACs. The performance of PMACs is dependent primarily on selecting message words and compressing the selected words. Although selecting the data words is a linear operation in message length, its dominant per-byte cost is the product of the percentage of data selected and the cost to select each data word. Similarly, the per-byte cost of compressing these words is the product of the percentage of data selected and the per-byte cost of the inner hash function. Thus, the dominant cost is reduced multiplicatively by the selection percentage. For example, in our prototype when selecting one word per 64 message words, the PMAC cost is approximately 0.05 clock cycles per message byte—over 10 times faster than our fastest non-PMAC authentication algorithm, and over 100 times faster than HMAC-MD5-96.

7.2. Strength-Performance tradeoff

Figure 4 summarizes the strength-performance levels achieved in our prototype implementation on a Pentium II. Specifically, Figure 4 shows, for various authentication mechanisms, speed versus the probability of forgery. A notable characteristic of this tradeoff is the large span of security-performance levels over which ACSA provides authentication. Also notable is the dramatic worsening of the forgery probability when PMACs are used. This increase in the forgery probability, however, is less dramatic when evaluated for large amounts of modified data.

[4] Personal email from Ted Krovetz (May 11, 1999).

8. Discussion and conclusion

ACSA System provides a new approach for meeting performance and authentication demands of high-speed networks. In ACSA, the communicants can choose various strength-performance levels, and they can change their choices within a communication association. This flexibility to choose, and to change within a connection, offers advantages over simply always using one authentication mechanism.

In addition to its overall gear-switching approach, the ACSA Project contributes and applies the novel ideas of Partial MACs (PMACs), inner-function groups (IFGs) with bit scattering, and multiple message authentication tags. Limitations of the ACSA System include lack of provable security (*e.g.* the security of IFGs with bit scattering is now heuristic), possibility of system attacks (*e.g.* an adversary might be able to alter the processor load and thereby cause a gear switch), and additional complexity. For example, in comparison with implementing a single authentication mechanism, it is more costly to implement the ACSA System and it is more difficult to verify the correctness of such an implementation. Nevertheless, such system attacks and implementation problems would likely at worst result in a lower-strength gear being used.

Our immediate future plans for ACSA are to complete and test our prototype system implementation to measure its speeds empirically at various strength levels. We also plan to investigate additional ways of choosing and changing gears adaptively, such as automatically sensing and responding to varying degrees of redundancy in the data stream, and including Quality of Service mechanisms of applications in the security association negotiations. In addition, we plan to write a draft specification (Internet RFC) for implementing ACSA with IKE and IPsec.

Open questions raised by the ACSA Project include the following. (1) What is the security of bit scattering and how much advantage does this operation contribute in relation to its performance cost? (2) Is it possible to extend ACSA ideas to data confidentiality? Although it is possible to combine ACSA with precomputed stream ciphers for fast confidentiality, we do not see how to extend such ideas as IFGs and PMACs to data confidentiality.

Using a variety of engineering techniques, the ACSA System offers a practical approach for providing data authentication on high-speed networks by trading off performance and authentication strength.

Acknowledgments

We thank Dennis K. Branstad for originally proposing the ideas of authentication strength-performance tradeoffs, partial message authentication, and gear groups. Phil Rogaway contributed very helpful detailed comments on our preliminary work. Ted Krovetz provided us with assistance in implementing UMAC. Also, we thank Hugo Krawczyk for constructive feedback.

References

[1] S. Abbott, "Cryptographic interfaces: supporting 100mbps and beyond," *Proceedings of the 1999 RSA Data Security Conference.*

[2] D. Balenson, D. Carman, M. Heyman, and A. Sherman, "Adaptive Cryptographically Synchronized Authentication (ACSA): Model and analysis," Revision 1.0 (December 7, 1998). 53 pages.

[3] M. Bellare, R. Canetti, and H. Krawczyk, "Keying hash functions for message authentication" in *Advances in Cryptology: Proceedings of CRYPTO '96*, LNCS 1109, N. Koblitz, ed., Springer-Verlag (1996), 1–15.

[4] J. Black, S. Halevi, H. Krawczyk, T. Krovetz, and P. Rogaway, "UMAC: Fast and secure message authentication" in *Advances in Cryptology: Proceedings of CRYPTO '99*, LNCS 1666, M. Wiener, ed., Springer-Verlag (1999), 216–233.

[5] A. Bosselaers, "Even faster hashing on the Pentium," presented at the rump session of Eurocrypt '97.

[6] D. Carman, J. Adcock, D. Balenson, M. Heyman, and A. Sherman, "Adaptive Cryptographically Synchronized Authentication (ACSA): Prototype system design," Revision 1.0 (May 12, 1999). 56 pages.

[7] DARPA ITO ACSA Web Page, http://www.darpa.mil/ito/psum1998/G374-0.html.

[8] M. Etzel, S. Patel, Z. Ramzan, "Square hash: Fast message authentication via optimized universal hash functions," in *Advances in Cryptology: Proceedings of CRYPTO '99*, LNCS 1666, M. Wiener, ed., Springer-Verlag (1999), 234–251.

[9] V. Gligor, and P. Donescu, "Integrity-aware PCBC encryption schemes," *Security Protocols - 7th International Workshop*, LNCS, B. Chrisianson, B. Crispo, and M. Roe, ed., Cambridge, U.K., Springer-Verlag (April 1999).

[10] S. Halevi and H. Krawczyk, "MMH: Software message authentication in the Gbit/second rates" in *Fast Software Encryption*, LNCS 1233, Eli Biham, ed., Springer-Verlag (1997), 172–189.

[11] D. Harkins and D. Carrel, "The Internet key exchange (IKE)," RFC 2409 (November 1998). ftp://ftp.isi.edu/in-notes/rfc2409.txt.

[12] S. Kent and R. Atkinson, "IP authentication header," RFC 2402 (November 1998). ftp://ftp.isi.edu/in-notes/rfc2402.txt.

[13] S. Kent and R. Atkinson, "Security architecture for the internet protocol," RFC 2401 (November 1998). ftp://ftp.isi.edu/in-notes/rfc2401.txt.

[14] H. Krawcyzk, M. Bellare, and R. Canetti, "HMAC: Keyed-hashing for message authentication," RFC 2104 (February 1997). ftp://ftp.isi.edu/in-notes/rfc2104.txt.

[15] C. Madson and R. Glenn, "The use of HMAC-MD5-96 within ESP and AH," RFC 2403 (November 1998). ftp://ftp.isi.edu/in-notes/rfc2403.txt.

[16] C. Madson and R. Glenn, "The use of HMAC-SHA-1-96 within ESP and AH," RFC 2404 (November 1998). ftp://ftp.isi.edu/in-notes/rfc2404.txt.

[17] T. Matthews, "Suggestions for random number generation," RSA Laboratries' Bulletin, No. 1 (January 22, 1996). 4 pages. http://www.rsa.com/rsalabs/html/bulletins.html.

[18] E. Nahum, S. O'Malley, H. Orman, and R. Schroeppel, "Towards high performance cryptographic software," Technical Report TR95-04, Dept. Computer Science, University of Arizona (1995).

[19] P. Rogaway, "Bucket hashing and its application to fast message authentication" in *Advances in Cryptology: Proceedings of CRYPTO '95,* LNCS 963, D. Coppersmith, ed., Springer-Verlag (1995), 313–328.

[20] P. Rogaway, "Design and analysis of message authentication codes," *Proceedings of the 1996 RSA Data Security Conference* (January 19, 1996).

[21] P. Rogaway, "Bucket hashing and its application to fast message authentication," *Journal of Cryptology,* Vol. 12, No. 2 (Spring 1999).

[22] V. Shoup, "On fast and provably secure message authentication based on universal hashing," *Advances in Cryptology: Proceedings of CRYPTO '96*, LNCS 1109, N. Koblitz, ed., Springer-Verlag (1996), 74–85.

[23] J. Touch, "Report on MD5 performance," RFC 1810 (June 1995). 7 pages. http://www.isi.edu/touch/pubs/rfc1810.html.

[24] M. Wegman and L. Carter, "New hash functions and their use in authentication and set equality," *Journal of Computer and System Sciences*, Vol. 22 (1981), 265–279.

Appendix A. Pseudocode for bit scattering

ScatterBits

Input: message, mask, mask_offset, binA, binB
Result: binA and binB are modified

Description:

Under the control of a pseudorandomly-selected mask, scatter the bits of the message two words at a time into bins A and B. Scatter the next two message words as follows. Twice use the next mask word to mark bits in the next two message words. Interleave the marked bits of the first word with the unmarked bits of the second word, and toss the result into bin A. Interleave the unmarked bits of the first word with the marked bits of the second word, and toss the result into bin B. In the beginning, start with the mask word at the specified mask offset position.

Begin

```
1.  L = length( message )
2.  for X, Y being the next pair of words in message
3.    with M being the next word in mask
4.      from mask_offset do

          % 0-fill unmarked X bits; 0-fill marked Y bits
5.        interleaveA := (X and M) or (Y and (not M))
6.        push interleaveA into binA

          % 1-fill marked X bits; 1-fill unmarked Y bits
7.        interleaveB := (X or M) and (Y or (not M))
8.        push interleaveB into binB
9.  endfor
```

End