

Adaptive Cryptographically Synchronized Authentication (ACSA)¹

Model and Analysis

Revision 1.0

December 7, 1998

David M. Balenson, Principal Investigator
David W. Carman, Co-Principal Investigator
Michael D. Heyman
Dr. Alan T. Sherman

Cryptographic Technologies Group
TIS Labs at Network Associates, Inc.
3060 Washington Road (Rt. 97)
Glenwood, MD 21738

¹ Sponsored by the Defense Advanced Research Projects Agency (DARPA) under Rome Laboratory Contract No. F30602-98-C-0215.

Contents

Executive Summary.....	4
1 Introduction	6
1.1 Scope	6
1.2 Overview of Document.....	7
2 ACSA Model.....	8
2.1 Background	8
2.2 ACSA System Components.....	9
2.2.1 Overview	9
2.2.2 Controller.....	9
2.2.3 Security Association and Key Management Module	11
2.2.4 Security Services Module	12
2.2.5 Network Defenses	12
2.2.6 CPU Load Monitor	12
2.2.7 Local Policy Interface	12
2.2.8 Application Policy Interface.....	12
2.2.9 Network Application.....	12
2.3 Gears, Gear Synchronization, and Adaptive Gear Control	13
2.3.1 Security Association Negotiation and Establishment	13
2.3.2 ACSA Session Establishment.....	13
2.3.3 Steady State Operation	16
2.3.4 Gear Switching.....	16
2.4 Methods of Constructing Authentication Tags.....	18
2.4.1 Conventional Methods.....	18
2.4.2 MAC Groups	20
2.4.3 Probabilistic Authentication.....	23
3 Analysis of the ACSA Model.....	25
3.1 Security.....	25
3.1.1 Overview	25
3.1.2 Attacks on ACSA Authentication Types.....	25
3.1.3 Attacks on ACSA Control Messages	27
3.1.4 Environmental Attacks.....	28
3.1.5 Timing Attacks.....	28
3.1.6 Replay Attacks	29
3.2 Performance	29
3.2.1 Performance-Security Tradeoff.....	29
3.2.2 Other Performance Factors	30
3.3 Optimization of Model Parameters.....	31
3.3.1 Optimal Number of Gears.....	31
3.3.2 Optimal Number of Tags in a MAC Group with Subset Tags	31
3.4 Other Considerations	32
3.4.1 Applicability	32
3.4.2 Implementation Issues.....	32
Appendix A Candidate Authentication Mechanisms.....	33
A.1 Block Cipher Based MACs.....	33
A.1.1 DES-MAC.....	33
A.1.2 Other Block Cipher Based MACs.....	33

- A.1.3 Pre-computed stream cipher based XOR MAC 34
- A.2 Inner MAC Function Candidates 36
 - A.2.1 SHA-1, MD5, MD4..... 36
 - A.2.2 Bucket Hash..... 37
 - A.2.3 Alternate Hash Algorithm..... 38
 - A.2.4 Multilinear Modular Hash and Non-linear Modular Hash..... 39
 - A.2.5 Cryptographic CRC 41
 - A.2.6 Keyed Pseudo-Random-Function (PRF) 42
- A.3 Performance Results For Selected Algorithms..... 43
- A.4 Pseudo Code..... 45
 - A.4.1 ACSA-HMAC..... 45
 - A.4.2 DO SHA-1 45
 - A.4.3 DO Bucket Hash..... 45
 - A.4.4 DO AHA Hash 45
 - A.4.5 Do MMH Mac 46
 - A.4.6 Do NMH Mac..... 47
 - A.4.7 Keyed Pseudo-Random-Function (PRF) MAC..... 47
 - A.4.8 Do CRC Hash..... 48
 - A.4.9 Do Panama Hash 48
- Appendix B Acronyms 49
- Appendix C Glossary 50
- Appendix D References..... 52

Figures

- Figure 1 - Network Authentication Process 8
- Figure 2 - ACSA System Components..... 9
- Figure 3 - CPU Load Optimization Algorithm 17
- Figure 4 - Block Diagrams of Nested MAC, HMAC-SHA1-96 and HMAC-MD5-96..... 19
- Figure 5 - Block Diagram of a MAC Group with Two Inner MAC Functions..... 21
- Figure 6 - Block Diagram of a MAC Group with Subset Tags..... 22
- Figure 7 - Block Diagram of Probabilistic Authentication..... 23
- Figure 8 - Comparison of Speed versus Security for ACSA Authentication Mechanism Types . 30
- Figure 9 - DES-MAC..... 33
- Figure 10 - Block Cipher MAC 34
- Figure 11 - Pre-computed Cipher XOR MAC 35
- Figure 12 - Bucket Hash 37
- Figure 13 - Bucket Hash Tradeoffs 38
- Figure 14 - Alternate Hash Algorithm..... 39
- Figure 15 - Multilinear Modular Hash and Non-linear Modular Hash 40
- Figure 16 - Cryptographic CRC..... 41
- Figure 17 - Performance Results for Selected Algorithms..... 44

Executive Summary

Providing authentication is essential to network security. Conventional authentication mechanisms, however, cannot operate at speeds fast enough to meet the demands of ultra-fast networks. The Adaptive Cryptographically Synchronized Authentication (ACSA) project² is exploring an effective solution to the dilemma of wanting fast networks and strong network authentication. As with conventional authentication methods, a sender constructs an authentication tag from an authentication key and a packet of network data. The authentication tag and network data are sent to and verified by a receiver. The choice of which of various methods of constructing authentication tags is first determined when a security association is established between the sender and receiver. However, unlike conventional methods, the ACSA model provides a novel performance-security tradeoff in which the network adaptively varies the level of authentication assurance based on processor load and authentication errors. The proposed method provides acceptable communication speed and authentication by changing the probability of detecting malicious activity, but in a manner such that an adversary cannot determine the current, cryptographically controlled security level.

The ACSA model supports three basic types of authentication mechanisms:

- *Conventional authentication mechanisms* are computationally intensive but provably secure. Standard algorithms included HMAC-SHA1-96 and HMAC-MD5-96.
- *MAC groups* consist of a nested MAC comprising multiple fast, albeit weak, inner MAC functions and a strong, computationally intensive, outer MAC function. MAC groups are potentially much faster than conventional authentication mechanisms because they use faster algorithms. MAC groups apply network data to the inner MAC functions in a manner known only to the sender and receiver, thus providing additional protection from a potential adversary.
- *Probabilistic authentication* requires even less computation time than do MAC groups by performing the authentication tag calculation on only a subset of the network data message. As with MAC groups, probabilistic authentication is applied to the message subset known only to the sender and receiver.

Initial analysis shows that by using combinations of the above three mechanisms, ACSA can maintain a performance-security tradeoff at acceptable levels of authentication security for many high-speed network applications. Computationally intensive conventional mechanisms provide high levels of security. MAC groups containing fast inner MAC functions provide significant reductions in the tag computation time, thus providing large (up to at least a factor of five) performance advantages over conventional mechanisms for large packet sizes (64 Kilobytes or more). Probabilistic authentication requires significantly less tag computation time (up to factor of 100) versus conventional mechanisms, but significantly reduces the system's ability to detect modification of a single data word. Nevertheless, probabilistic authentication remains a viable choice for many applications because it detects modification of larger amounts of network data.

² Sponsored by the Defense Advanced Research Projects Agency (DARPA) under Rome Laboratory Contract Number F30602-98-C-0215.

The ACSA System provides a novel, practical, adaptive strategy for balancing the performance, authentication security, and implementation cost requirements for many network applications.

1 Introduction

The provision of authentication is essential to the security of networks. However, conventional authentication mechanisms cannot operate at speeds fast enough to meet the demands of ultra-fast networks. The DARPA-sponsored Adaptive Cryptographically Synchronized Authentication (ACSA) project is exploring an effective solution to the dilemma of wanting fast networks *and* strong network authentication. The ACSA solution provides a novel performance-security tradeoff in which the network adaptively varies the level of authentication assurance based on current risk and network load. The proposed method provides acceptable communication speed and authentication by changing the probability of detecting malicious activity, but in a manner that an adversary cannot determine the current, cryptographically controlled, security level.

Even the fastest cryptographic authentication implementations are far too slow for currently available ultra-fast networks. Some estimates indicate there could be at least a factor of 100 difference between conventional authentication speeds and that of ultra-fast networks. While cryptographic-based data-stream source authentication techniques provide high security, they induce time delays that are unacceptable. However, risks change dynamically within a network, based on factors such as initialization, reconfiguration, soft failures, and intentional attacks. Acceptable authentication systems that adapt to changing risks and network loading can be created using probabilistic selection methods and algorithms.

The overall goal of the ACSA project is to provide a strong authentication solution that is computationally efficient enough to meet the demands of ultra-fast networks. The project is structured around three primary objectives:

1. identify a spectrum of practical cryptographic authentication mechanisms that can be used selectively like gears of an automobile transmission to provide various security/performance levels of data authentication under the assumption that strong authentication cannot be provided 100% of the time.
2. design a control system that establishes and maintains acceptable levels of authentication in an environment where the computational load of participating network devices and the threats to the network are dynamically changing.
3. implement a prototype system that can be used to demonstrate effective authentication by adapting the authentication process to satisfy dynamic speed requirements. The prototype system will be based on the IPSec and IKE architectures in an effort to explore wide applicability of the ACSA solution.

1.1 Scope

ACSA addresses only one element of the total network security solution: connectionless integrity and data origin authentication between a sender and a single receiver. Although ACSA was designed with eventual multicast authentication in mind, our model and analysis address only ACSA applicability to point-to-point sender-receiver relationships.

Conventional key management and confidentiality services are required by the ACSA system, but are not described in detail and are deemed outside the scope of ACSA. Although ACSA does not provide confidentiality, it may be used in addition to and independent of network data confidentiality services. Furthermore, the ACSA model provides many building blocks that may be applicable to a confidentiality solution for ultra-fast network applications.

Although the ACSA solution appears most beneficial to CPU-limited platforms, the ACSA model is designed to be applicable to all network devices that require authentication security. Analysis of ACSA will not examine the benefits of using high-speed cryptographic hardware or multi-processor platforms to perform authentication tag computations, although the ACSA solution should apply and prove beneficial to platforms with these configurations as well [Nahum, et.al].

1.2 Overview of Document

The rest of this document is organized into two main sections. Section 3 defines the model, including its components, gears, and methods of constructing authentication tags. Section 4 provides initial analysis of the model, focusing on security threats and countermeasures, performance-risk tradeoffs, and optimization of model parameters. In addition, Appendix A lists and describes specific candidate authentication mechanisms that can be used within the ACSA system.

2 ACSA Model

2.1 Background

The ACSA model describes an efficient yet acceptably secure method of providing network data authentication between a sender and receiver. Figure 1 shows a basic authentication process involving a sender and receiver where an authentication tag is appended to a message. We use the terms message and packet interchangeably.

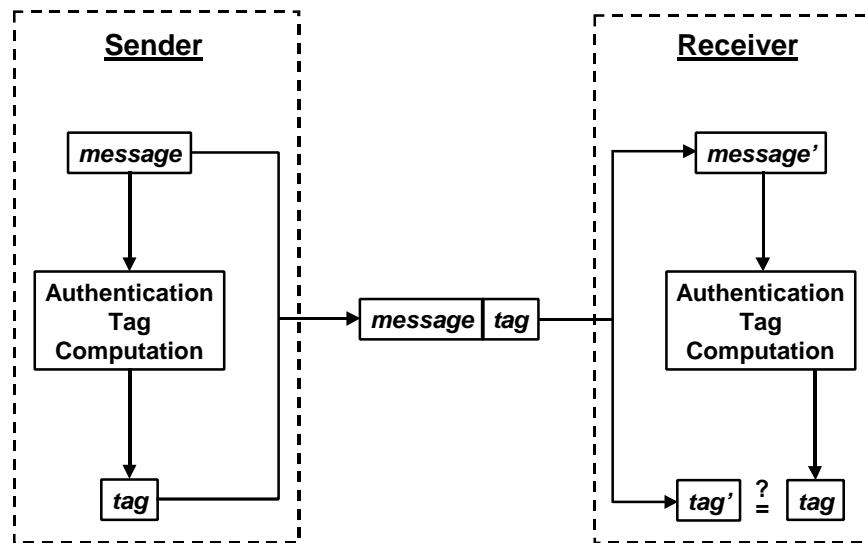


Figure 1 - Network Authentication Process

The definition of authentication used in this document is similar to that defined in IPSec [IP Authentication Header98], where the authentication tag is “used to provide connectionless integrity and data origin authentication.” One method of providing authentication of a message is through use of a digital signature. Digital signatures are based on computationally intensive public key operations, however, making them too slow for high-speed network use.

Faster authentication alternatives to digital signature include message authentication codes (MAC) such as HMAC-SHA1-96 and HMAC-MD5-96. Although HMAC-SHA1-96 and HMAC-MD5-96 have been adopted as standards for popular security protocols such as IPSec, they still may be too slow for some network applications. For large amounts of data, the speed of these functions is determined by the underlying hash algorithms, SHA1 and MD5, respectively. On a Pentium processor, SHA-1 and MD5 require approximately 53 and 21 clock cycles per 32-bit data word processed. These values may seem quite reasonable for many applications, but are untenable for high-speed networks on CPU-limited platforms.

Therefore, a goal of the ACSA model is to include faster authentication methods and provide ways of selecting when to apply a particular authentication method to a data packet in accordance with a continual monitoring of the number of packets failing to

authenticate. The ACSA model must be applicable to a wide range of network devices, including security gateways, firewalls, routers and end-user personal computers. Furthermore, the ACSA model must be capable of being mapped to the IKE and IPSec network security protocols.

The following sections describe the various methods of constructing authentication tags, how these methods are used in the ACSA system, and the components of the ACSA system that perform these functions.

2.2 ACSA System Components

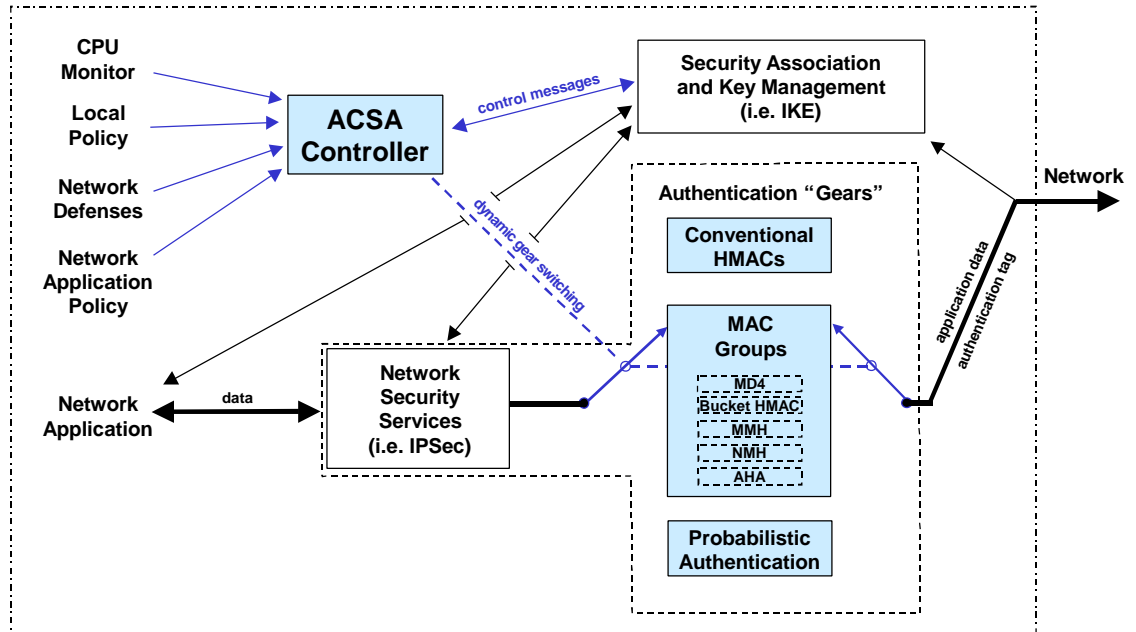


Figure 2 - ACSA System Components

2.2.1 Overview

2.2.2 Controller

The heart of the ACSA system is the ACSA Controller. The ACSA Controller receives a number of inputs from other system components including:

- local network policy,
- local network application policy,
- processor load,
- network defense alarms,
- authentication errors, and
- remote authentication errors and ACSA parameters.

The ACSA Controller is primarily responsible for determining the appropriate authentication mechanism, hereinafter called the gear, to use in providing authentication

for data exchanged between nodes sharing a given security association. The ACSA Controller individually selects and maintains a gear for each security association.

For each security association, the node functions as either the sender or the receiver. Similarly, for each security association, the ACSA Controller will also function in the sender or receiver mode. In the sender mode, the ACSA Controller must determine the gear to be used and notify the receiver of the determined gear. In the receive mode, the ACSA Controller negotiates gear information with the sender mode ACSA Controller, setting the gear as denoted by the sender mode ACSA Controller.

2.2.2.1 Sender Mode Functions

When in the sender mode, the primary functions of the Controller are to determine initially determine the gear, switch the gear as necessary, and disseminate the gear information to the receiver.

Initial Gear Determination and Dissemination Functions

- Receives remote ACSA parameter information from receiver via an ACSA control message.
- Determines the pairwise gear suite based on local and remote ACSA parameter information.
- Determines gear based on local system information and remote ACSA parameter information – details of gear determination are included in Section 2.3.2.2.
- Sends gear information to the Network Security Services module so the gear may be applied to application data associated with this security association.
- Sends gear information to the receiver via an ACSA control message.

Gear Switching and Dissemination Functions

- Receives updated information from receiver via ACSA control messages concerning authentication errors, CPU load, etc.
- Periodically re-determines gear based on local and remote information as described in Section 2.3.4.
- Sends updated gear information to the Network Security Services module so the updated gear may be applied to application data associated with this security association.
- Sends updated gear information to the Security Association and Key Management module to be forwarded to the receiver.

2.2.2.2 Receive Mode Functions

When in the receive mode, the primary functions of the Controller are to notify the sender of ACSA parameters and errors, and receive and locally disseminate gear information.

Gear Establishment and Maintenance Functions

- Upon security association establishment, determines ACSA parameter information based on local environment factors such as network policy, application policy, etc.
- Sends ACSA parameter information to remote sender via an ACSA control message.
- Receives actual or updated gear and gear suite information from the ACSA sender via the security association and key management module.
- Sends gear information received from the sender to the Network Security Services module so the gear may be applied to application data associated with this security association.
- If the local CPU load is too heavily loaded (and possibly on the basis of other factors), the receiver's Controller autonomously selects either to perform computations of a subset_tag of MACs Groups instead of the whole_tag on some packets, or does not perform any authentication tag computations at all on some packets.
- Periodically re-determines ACSA parameter information based on local information, specifically CPU load, receive authentication errors, and local alarms.

Autonomous Gear Switching

In the receive mode, the ACSA Controller will usually perform ACSA authentication according to the gear established and disseminated by the sender. However, if the CPU is loaded greater than the desired level, the receiver's Controller may instruct the Network Security Services Module to instead operate in one of two modes: (1) authenticate a subset tag of a MAC Group instead of the entire message, or (2) do not authenticate at all and simply pass the message to the application unverified.

2.2.2.3 Local Gear Suite Determination (Both Sender and Receiver Modes)

The ACSA Controller performs local gear suite determination upon initial ACSA gear establishment. Two main factors contribute to local gear suite determination: local security policy and application performance-security policy. Local security policy includes a delineation of the minimal authentication security to be applied to all ACSA authentication operations. Application performance policy denotes the minimum performance/speed/rate/quality-of-service desired by the application. Application security policy denotes the minimum authentication security to be applied to ACSA authentication operations involving this application, when this policy differs from that specified in the local security policy.

2.2.3 Security Association and Key Management Module

2.2.3.1 Conventional Functions

The ACSA system is not a complete network security protocol, rather, it requires a complete underlying network security protocol to perform its functions securely. The Security Association and Key Management module must generate the authentication key and confidentiality keys securely. The authentication key is used to provide

authentication for ACSA control messages as well as for authenticating network application data. The confidentiality key is used for securing ACSA control messages (and possibly also security network application data).

2.2.3.2 ACSA Control Message Support

If key management and control messages transit via the Security Association and Key Management module, this module must be augmented to also handle ACSA control messages. All ACSA control messages must be encrypted and authenticated.

2.2.4 Security Services Module

The Security Services module provides authentication security for network application data by appending an authentication tag. To support ACSA, the Security Services module will be augmented to include implementations of ACSA gears and report authentication errors to the Controller. If the Security Services module is responsible for passing control messages, it must then be augmented to handle ACSA control messages as well.

2.2.5 Network Defenses

Network defenses include firewalls, intrusion detection devices and other network security devices that detect suspected adversarial actions. Network defenses may notify the ACSA Controller of suspected or actual network or server attacks. The Controller may then switch the system to use a more secure gear based upon being notified of an alarm.

2.2.6 CPU Load Monitor

The CPU Load Monitor is responsible for providing the Controller with the current average processor utilization of the platform's CPU(s). The average utilization should be over a short enough time span to show the effects of various gear shifts (assuming all other process utilization being static), but over a long enough time span to include a representative group of the other processes being handled by the platform's CPU(s).

2.2.7 Local Policy Interface

The Network Policy Manager provides performance-security tradeoff information that assists the Controller in determining the appropriate gear suite.

2.2.8 Application Policy Interface

Individual network applications may have performance requirements (minimums), security requirements, or both. The ACSA model accounts for these requirements by allowing each application to send these requirements to the ACSA Controller in order to more precisely determine the ACSA gear suite.

2.2.9 Network Application

This component refers to the actual network application that desires authentication. The application should provide quality-of-service and security requirements to the ACSA Controller, if available. The application should also report any detected authentication errors to the ACSA Controller since there may be authentication errors which are detected at the application layer and not the network security services layer. Functionally, this information should be provided to the Controller via the same interface as the application policy as discussed in Section 2.2.8.

2.3 Gears, Gear Synchronization, and Adaptive Gear Control

In addition to providing various authentication mechanisms, the ACSA system provides a method to adaptively control which authentication mechanism to use for a given environment. The set of various authentication mechanisms, including their various permutations, can be thought of as gears of a transmission. Analogous to how when a vehicle needs to operate faster, it shifts the transmission into a higher gear, the ACSA system can similarly 'shift' into a faster gear. Since usually faster gears provide less authentication security, this approach provides a performance-security tradeoff. How these gears can be adaptively 'shifted' and synchronized between the sender and the receiver to provide this performance-security tradeoff will be addressed in this section.

An ACSA gear will generally be defined by the method of authentication mechanism (conventional vs. MAC group vs. probabilistic authentication) and, where applicable, it's inner MAC composition. For instance, a gear may be defined as a MAC group, composed of 50% Bucket Hashing, 25% MMH, and 25% AHA computations. Detailed gear specification will be described in a later version of this document.

To establish and maintain the optimal ACSA gear effectively, ACSA-enabled communicants will transition through four main states: (1) Security Association Negotiation and Establishment, (2) Gear Negotiation and Establishment, (3) Steady State Operation, and (4) Gear Switching. Synchronization of these states between the sender and the receiver will occur via the exchange of *ACSA control messages*. ACSA control messages are simply ACSA-specific messages that contain either ACSA parameter or status information. ACSA control messages will usually be protected via the confidentiality and authentication mechanisms provided for in the negotiated security association.

2.3.1 Security Association Negotiation and Establishment

Security association (SA) negotiation and establishment may be performed regardless of whether communicants participate in an ACSA system. However, ACSA requires a SA be established before gear negotiation and establishment, since exchanged ACSA parameter information should remain confidential.

2.3.2 ACSA Session Establishment

ACSA session establishment occurs either during or soon after SA negotiation and establishment. Each ACSA session must correspond to a one and only one SA. The ACSA session establishment steps are as follows:

1. First, the sender verifiably indicates that it is ACSA capable. This indication may be via a digitally signed capability field, certificate extension, or some other verifiable attestation. ACSA authentication will only take place if all participants are ACSA capable.
2. A receiver then sends ACSA session and gear information to the sender via ACSA-specific encrypted and authenticated messages.
3. The sender processes both its local parameters and this received information and determines the ACSA session parameters and the appropriate initial gear to begin exchanging data. If not all receive nodes are ACSA-capable, or if the intersection of the local and remote gear suites is null, then ACSA operation will not take place. (Even if ACSA operation cannot occur, non-ACSA authentication services may still be applied. The method by which non-ACSA authentication services are negotiated and provided for are outside the scope of the model.)
4. Finally, the sender notifies the receiver of the ACSA session parameters and the determined gear.

How the ACSA-specific messages are appended or interact with the SA negotiation will be dependent on the network security protocol in which these messages are embedded. Each network security protocol may require a unique design to embed ACSA-specific messages in the most efficient and interoperable fashion.

Sender- vs. Receiver-Controlled Determination of the ACSA Gear – For point-to-point relationships between a sender and single receiver, the ACSA design choice of having the sender determine the gear seems arbitrary. However, when ACSA’s applicability to multicast is considered, selecting the sender to determine the gear seems more applicable, especially if some of the receivers may be “receive-only”.

ACSA Session Parameters - During initial gear establishment, session parameters, such as those listed below, are sent from the receiver to the sender. Based on the local sender ACSA parameters, the sender will establish the *pairwise session parameters* for the duration of the SA:

- *Data word size* – This value is intended to represent the data size a receiver network device CPU desires to move data. This value will almost always be the machine size (i.e. Pentium-class = 32, Alpha, Merced = 64).
- *Heartbeat Interval* – The timeout interval during which an ACSA participant should expect to receive a control message from the remote party.
- *Gear suite* – An enumeration of the conventional gears and inner MAC functions that the receiver wishes to support for this session. If probabilistic authentication is supported, the receiver will note the minimum percentage of data words that must contribute to the authentication tag calculation. Although the network device may support a wide range of gears, some very slow or very insecure gears may be excluded from the gear suite due to network and/or application policy, quality-of-service, and/or performance considerations. The exchange of gear suite information will establish a common set of gears referred to as the *pairwise gear suite*.

Gear Parameters - A gear may be specified by the following parameters:

- *Gear type* – “Conventional,” “MAC Group,” or “Probabilistic Authentication”

- *Composition* –
 - For “Conventional”: the algorithm ID of the conventional algorithm (this usually will be strictly interoperable with conventional algorithms already in mandatory or optional use within the underlying network security protocol)
 - For “MAC Group”:
 - ◆ an enumerated list of the inner MAC functions which include:
 - ❖ the algorithm ID of the inner MAC function
 - ❖ any gear specific parameters for the inner MAC function (e.g. for Bucket Hashing, the number of output buckets)
 - ❖ the percentage of data words which contribute to the inner MAC function
 - ◆ the algorithm ID of the outer MAC function
 - ◆ any gear specific parameters for the outer MAC function
 - For “Probabilistic Authentication”:
 - ◆ the algorithm ID of the inner MAC function (usually the fastest gear of the pairwise gear suite)
 - ◆ any gear specific parameters for the inner MAC function
 - ◆ the percentage of data words which contribute to the inner MAC function
 - ◆ the algorithm ID of the outer MAC function
 - ◆ any gear specific parameters for the outer MAC function

2.3.2.1 Pairwise Session Parameter Determination

Upon receiving the receiver’s session parameters, the sender determines its local session parameters and then calculates the pairwise session parameters as follows:

- Pairwise data word size: The larger of the receiver and sender’s data word sizes.
- Pairwise heartbeat interval: The smaller of the receiver and sender’s desired heartbeat intervals
- Pairwise gear suite: The intersection of the receiver’s gear suite and the sender’s gear suite (if the intersection is the null set, ACSA operation cannot occur)
- Base gear: The most secure (and slowest) gear of the pairwise gear suite (i.e. supported by both the receiver and the sender)—usually this gear will be a conventional gear supported by the underlying network security protocol
- Max gear: The fastest (and least secure) gear of the pairwise gear suite (i.e. supported by both the receiver and the sender)

2.3.2.2 Initial Gear Determination

There are many factors to be considered in determining the initial gear. First, if the initial gear were to always be set to a well-known gear, the adversary could also know this and would therefore have the advantage of knowing what gear to attack. If the well-known gear was very secure, the adversary’s attack would be unsuccessful, but the performance impact of starting off in such a computationally intensive gear could cause problems on some network device platforms. Alternatively, if probabilistic authentication were selected, an adversary would be able to take advantage of the fact that not all data words of a session’s initial packets would contribute to the authentication tag calculation.

If MAC groups are supported, it is recommended that a pseudorandomly determined mix of inner MAC functions be selected for the initial gear. The pseudorandom mix might be constructed such that the average gear selected equals the median performance gear of the gear suite. However, as long as the initial gear is reasonably secure and not predictable by an adversary, its not clear too many other requirements need to be levied on it.

2.3.3 Steady State Operation

During operation of a network security protocol, packets are exchanged according to the established SA. If the SA were supplemented with ACSA gear negotiation, nodes perform authentication operations according to the negotiated gear. If an authentication mechanism value must be identified in a plaintext fashion during normal communications, ACSA participants must always identify the mechanism using a generic ACSA algorithm ID to prevent disclosure of the actual gear to the adversary.

During normal ACSA operation, *Heartbeat messages* are sent from the receiver back to the sender periodically. Heartbeat messages may contain authentication error, CPU load information, missing packet information and desired gear information. Other ACSA control messages such as *Authentication Error*, *Synchronization Error*, and *Gear Switch Request* messages may be sent from the receiver to the sender. Heartbeat messages provide a mechanism for the sender to confirm that the receiver is gear synchronized, and allows the sender to adapt the gear selection more to the receiver's desires, as discussed in the next section.

If a SA expires and must be re-established, a new ACSA session must also be established.

2.3.4 Gear Switching

The gear is switched under the following conditions:

- a receiver detects a condition it determines could require or desire a gear change, notifies the sender, and the sender determines a gear change is appropriate,
- the sender does not receive an ACSA control message from the receiver within the heartbeat interval,
- the sender detects a local environment condition and determines it requires a gear change, and
- the sender periodically pseudorandomly changes the gear.

The rationale for periodically pseudorandomly changing the gear includes the following: (1) it deters the adversary from determining the gear via long-term averaged information; (2) it lessens the benefit to an adversary of compromising a gear since less data will be associated with it; and (3) it gives the adversary less time to make use of a successful attack before the gear switches again. The size of this "period" will be a tradeoff between keeping the long-term averaged CPU cost of gear switching very low, while switching often enough to accomplish the three goals above.

CPU Load Optimization – Since policy seldom changes during a session and alarms will rarely occur (we hope), CPU load will be the main environmental factor that will vary within the ACSA system during an ACSA session. Thus, CPU load will be the main environmental factor driving gear switching. Since a major goal of the ACSA system is to provide authentication without overloading the network device’s CPU, a basic gear switching algorithm that addresses this need *in the absence of recent alarms or authentication errors* is shown in. Further specification of what specific actions must occur for “switch to less computationally intensive gear” and “switch to more secure gear” will be platform and situation dependent.

		Receiver Processor Utilization		
		Too Heavily Loaded	Near Desired CPU Load	Lightly Loaded
Sender Processor Utilization	Too Heavily Loaded	switch to less computationally intensive gear	switch to less computationally intensive gear	switch to less computationally intensive gear
	Near Desired CPU Load	if current gear is already a fast MAC group, switch to a MAC group with subset tags, else switch to less computationally intensive gear	maintain current gear	maintain current gear
	Lightly Loaded	if current gear is already a fast MAC group, switch to a MAC group with subset tags, else switch to less computationally intensive gear	maintain current gear	switch to more secure (more computationally intensive) gear

Figure 3 - CPU Load Optimization Algorithm

Alarm or Authentication Error-Based Gear Switching – Upon detection of either a local or remote alarm or authentication error, the sender may choose to switch to a more secure gear. Gear switching need not occur when a single alarm or error is detected, and depends upon local and application policy. Also, local alarms, remote alarms, and authentication errors may have different thresholds for causing a gear switch.

Once the sender determines the gear should be switched, the following protocol is invoked:

1. The sender securely sends an ACSA control message, called the *gear switch message*, to the receiver that includes the new gear information and which packet

the new gear will begin to be applied, called the *gear switch point* (e.g. an IPsec authentication header sequence number).

2. The sender then waits for the receiver to reply with an ACSA control message acknowledging the gear change.
3. After receipt of the sender's gear switch message, but before the expiration of the heartbeat interval, the receiver replies with a gear switch acknowledgement message.
4. The receiver continues authenticating using the previous gear until the gear switch point.
5. If the sender doesn't receive a gear switch acknowledgment message, from the receiver before the gear switch point, the sender sends a synchronization error message to the receiver(s) and switches gears to the base gear. If the sender does receive a gear switch acknowledgment message, it switches to the new gear.
6. If the receiver receives a synchronization error message from the sender, it begins authenticating using the base gear at the gear switch point. Otherwise, it begins authenticating using the new gear at the gear switch point.
7. If the receiver detects an authentication error on the first packet after the gear switch point, it sends a synchronization failure message to the sender and thereafter authenticates using the base gear, regardless of the reason for the failure. In this case, the sender will eventually switch back to the base gear, either upon receipt of the synchronization failure message or upon lack of receipt of the heartbeat message within the heartbeat interval.

2.4 Methods of Constructing Authentication Tags

The gears of the ACSA model provide three general methods for constructing authentication tags: conventional authentication mechanisms, MAC groups, and probabilistic authentication. Conventional mechanisms include existing standard algorithms that provide strong security but are relatively slow when applied to high-speed network applications. MAC groups offer a novel technique to provide acceptable security at much higher speeds. Probabilistic authentication offers even faster performance at the expense of a more dramatic decrease in authentication security. This section describes these three methods.

2.4.1 Conventional Methods

In IPSec [IPSec98], a popular network security protocol, data authentication is provided via the calculation of one of two mandatory-to-implement HMAC algorithms: HMAC-SHA1-96 [HMAC-SHA1-96] and HMAC-MD5-96 [HMAC-MD5-96]. The computation of the HMAC results in an *Integrity Check Value* (ICV) hereinafter called an *authentication tag*. The HMAC algorithm is an instantiation of a more general construct called a *nested MAC* (NMAC) that was described in [Bellare, Canetti, Krawczyk96]. The general construction of the NMAC is shown in Figure 4, and described mathematically as follows:

$$\text{NMAC}_k(x) = f_{k_1}(F_{k_2}(x)) , \quad \text{where } x \text{ is the message being authenticated}$$

The “ $F_{k_2}(x)$ ” portion of the nested MAC function is referred to as the *inner MAC*, whereas the “ $f_{k_1}()$ ” portion of the nested MAC function is referred to as the *outer MAC*.

As is shown in

Figure 4, the HMAC-SHA1-96 standard is an instantiation of an nested MAC construction where:

1. $k_1 = k \oplus \text{opad}$, where \oplus denotes the XOR operation, and where opad is the outer pad constructed by repeating the hexadecimal byte ‘36’ to the input block size of the digest,
2. $k_2 = k \oplus \text{ipad}$, where ipad is the inner pad constructed by repeating the hexadecimal byte ‘5c’ to the input block size of the digest,
3. F is an SHA-1 hash of the message concatenated with k_2 ,
4. f is an SHA-1 hash of the output of F concatenated with k_1 , and
5. the output of f is truncated to 96 bits to form the authentication tag.

Similarly, the HMAC-MD5-96 standard is an instantiation of nested MAC identical to HMAC-SHA1-96 except that both F and f use the MD5 hash function.

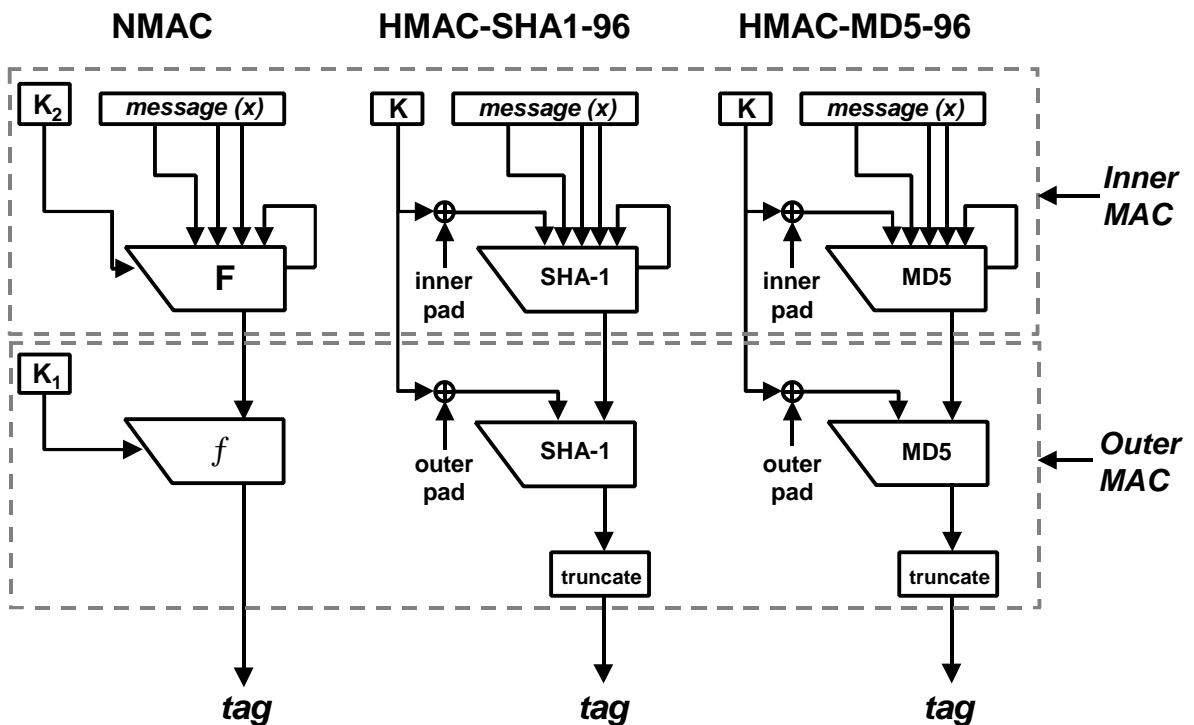


Figure 4 - Block Diagrams of Nested MAC, HMAC-SHA1-96 and HMAC-MD5-96

Performance – The performance of the HMAC-SHA1-96 and HMAC-MD5-96 algorithms are dominated by the speed of the SHA-1 and MD5 hash algorithms, respectively. SHA-1 computation time is approximately 52 Pentium clock cycles per 32-bit message data word, whereas MD5 computation time is approximately 21 clock cycles.

Security - Both HMAC-SHA1-96 and HMAC-MD5-96 are provably secure with excellent security bounds.

2.4.2 MAC Groups

Creating faster Nested MACs - When f , the outer MAC, and F , the inner MAC, are comprised of strong hash functions, the nested MAC construction is provably secure with excellent security bounds. Furthermore, in [Bellare, Canetti, Krawczyk], the authors state that “the weak-collision-freeness assumption made in the theorem can be replaced by the much weaker assumption that the inner hash function is collision resistant to adversaries that see the hash value only after it was hashed again with a different secret key. This extra strength of NMAC is demonstrated by the fact that current methods for finding collisions in MD5 and MD4 seem useless for attacking NMAC, even if both the inner and the outer hash functions are either MD4 or MD5.”

It is this property of the nested MAC whereby the inner MAC may be weaker than the outer MAC that we exploit in ACSA. Since the inner MAC will be applied iteratively, and the outer MAC only once, a computationally efficient, albeit weaker, inner MAC can provide a significant performance enhancement to the nested MAC calculation. Faster inner MAC functions can be composed from hash functions faster than MD5 and SHA-1. Several candidate hash functions currently exist, including MD4, Bucket Hashing, Multilinear Modular Hashing (MMH), Cyclic Redundancy Code-based Hashing (CRC-64), and Alternative Hash Algorithm (AHA). Inner MACs composed of such algorithms as Bucket Hashing and MMH, may be constructed as tree MACs in order to reduce the amount of data to be passed to the outer MAC. A more complete treatment of candidate functions appears in Appendix A.

Even faster inner MAC alternatives - Beyond simply providing faster keyed hash functions, we hypothesize that secure and even more efficient keyed pseudorandom functions (e.g. $F(k1, x)$) exist as well. Instead of merely concatenating the key to the message and then hashing (e.g. $F(k1 || x)$), it is proposed that $k1$ could be more pervasively involved in the MAC computation. For instance, since the inner MAC will be computed via many successive iterations, $k1$ could be included in each iteration of the MAC computation, rather than just the final iteration. The benefit of such pervasive use of the key in the inner MAC computation could result in a further relaxation of the collision resistance requirement for the function used within the inner MAC. At time of writing, however, this concept does not appear to have been examined by the cryptographic community.

Attacks against weak inner MACs – One of the challenges of providing authentication security for unencrypted data is that a third-party adversary has access to almost all the inputs to the authentication tag computation. As a result, when a nested MAC construction is used to generate the authentication tag, the adversary can attack a weak inner MAC, even if it doesn't know the authentication key(s). For instance, an adversary might be able to find a collision of the inner MAC computation, thus finding another

message that would have the same authentication tag. Despite their security vulnerability, we would like to use weak inner MACs because of their computational efficiency.

Protecting weak inner MACs – To increase the authentication security of gears with weak inner MACs, we need to reduce the adversary’s knowledge of the values that contribute to the authentication tag. To do this, we create a method of authentication called a *MAC group*, comprising two or more inner MAC functions that feed a single outer MAC, as shown in Figure 5.

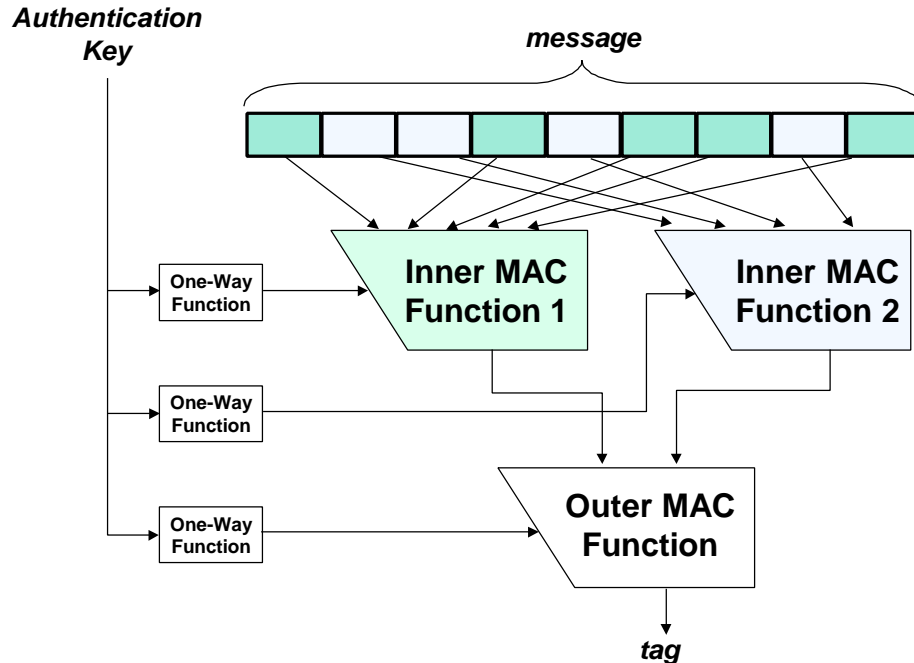


Figure 5 - Block Diagram of a MAC Group with Two Inner MAC Functions

When implementing a MAC group, we propose using a pseudorandom probabilistic function based on the authentication key to determine which inner MAC each individual data word will contribute to in calculating the authentication tag. By basing the function on the authentication key, we prevent the adversary from knowing the inner MAC a given data word contributes. Thus, even if a weak inner MAC is used, an adversary can not readily attack this weakness, since she does not know which data words contribute to the weak inner MAC, and which contribute to the other inner MAC(s).

The benefit of MAC groups may be better realized in light of the following attack. Assume an adversary is able to reliably determine collisions off-line for a weak inner MAC function. Using only this weak inner MAC for the entire message would allow the adversary to successfully modify the message and generate the same authentication tag. However, if a MAC group is instead used, and one of the data words modified by the adversary contributes to another inner MAC, the adversary’s attack will fail (with great probability). If the adversary modifies many data words or the weak inner MAC only processes a small percentage of the message, the odds an adversary’s attack will not be detected are relatively small.

MAC Groups with Subset Tags – An advantage of MAC groups with subset tags is that they may allow the receiver to verify an entire message, or a designated subset of the message, at the receiver’s choosing. To provide this capability, the sender must generate multiple authentication tags for each of the designated subsets and for the entire message. A MAC Group construction that generates subset tags is shown in Figure 6. As shown in the figure, the example MAC Group generates three authentication tags, *subset_tag1*, *subset_tag2*, and *whole_tag*, instead of the usual single authentication tag per message. (One could consider the concatenation of the three tags to constitute a single MAC Group authentication tag.)

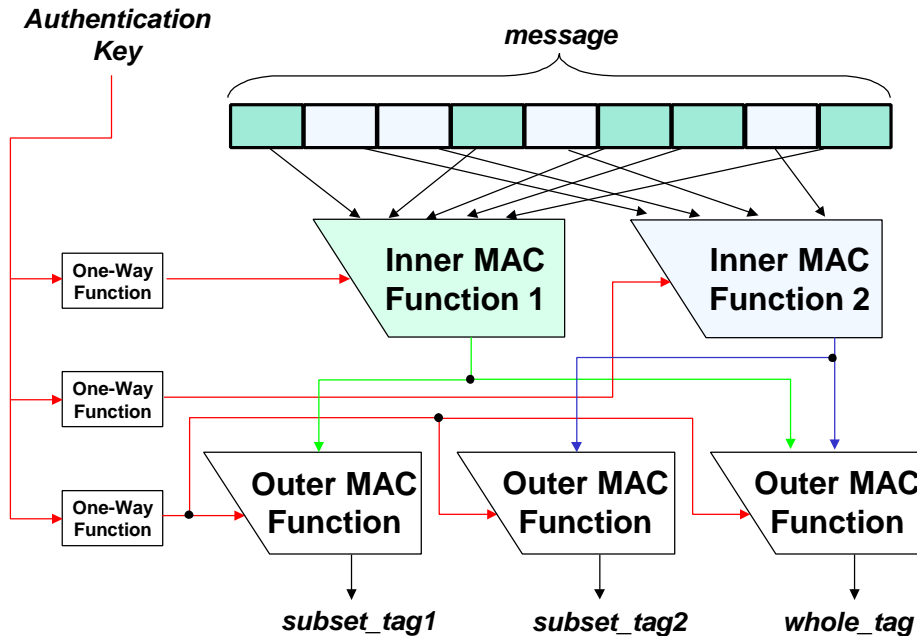


Figure 6 - Block Diagram of a MAC Group with Subset Tags

Advantages of MAC Groups with Subset Tags - The advantage of a MAC Group with subset tags construction is that the receiver can autonomously decide to verify the entire message using *whole_tag*, or only verify the subset of the message that contributed to a *subset_tag* computation. Verifying a *subset_tag* instead of *whole_tag* requires that only one of the two inner MAC functions be calculated and thus reduces the computation time required by the receiver. Thus, MAC Groups with subset tags are particularly effective when the receiver has less computational capability available for authentication tag computation than the sender.

Disadvantages of MAC Groups with Subset Tags – To accommodate MAC Groups with subset tags, the authentication tag size must be increased to include the multiple tags. Thus, gears that do not generate subset tags would be required to additionally generate pseudorandom data to pad the additional tag field so an adversary would be unable to distinguish gears that do and do not generate subset tags from gears based on the authentication tag size. Also, the security of MAC groups with subset tags drops dramatically, to that of probabilistic authentication, when a receiver verifies only the subset tag (i.e. *subset_tag1* or *subset_tag2* in Figure 6). Furthermore, generating MAC groups with subset tags contributes a modest increase in computation time for the

sender over similar MAC group computations since the outer MAC function must be computed multiple times, rather than just once.

Applicability of MAC Groups with Subset Tags to Multicast Relationships – MAC groups with subset tags may provide further advantages in multicast relationships since it is more likely that receivers of various computational capabilities will exist. Thus, it is more likely that some receivers with computational capabilities less than the sender could benefit from the probabilistic security of verifying a subset of the message.

2.4.3 Probabilistic Authentication

Once the inner and outer MAC computations have been made as efficient as practical, such as using fast inner MAC functions in MAC groups, further performance enhancement of nested MAC-based computations can only feasibly occur by reducing the iterations of the inner MAC function. Reducing the inner MAC iterations can be accomplished by reducing the amount of message, x , which is used in the authentication tag computation. Once portions of the message are not included in the authentication tag computation, the probability of the receiver detecting modification of a given data word, a basic measure of security, drops substantially. This method, called *probabilistic authentication*, represents a dramatic change in the tradeoff between security and speed. A block diagram showing how only a subset of data words of a packet contribute to the authentication tag when using probabilistic authentication is shown in Figure 7.

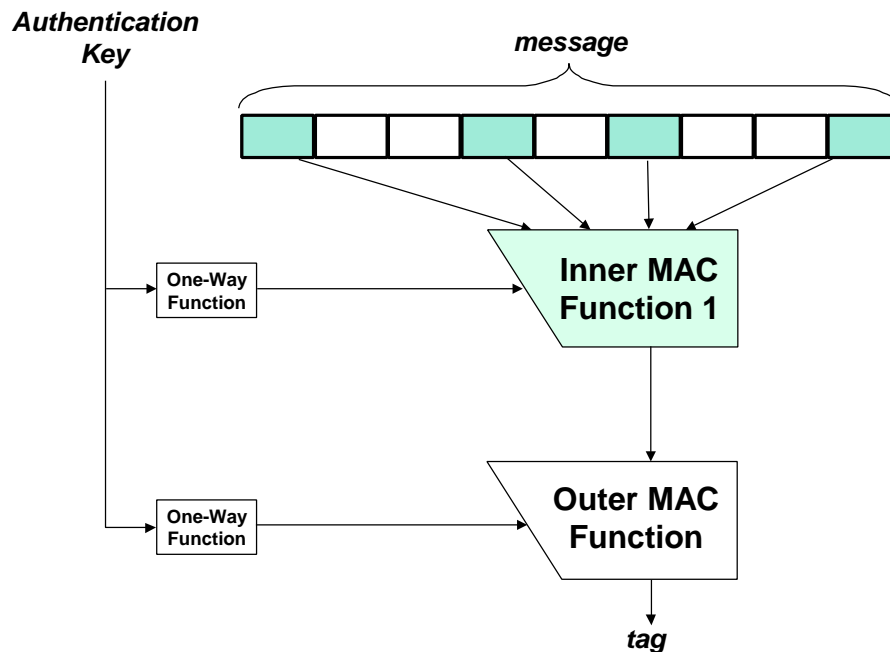


Figure 7 - Block Diagram of Probabilistic Authentication

There are many methods of reducing the message, x , from simply truncating, to more sophisticated algorithms. We propose using a keyed pseudorandom probabilistic

function to determine whether each individual data word will be used in the authentication tag calculation. Since an adversary will not have access to the key used to determine if a data word is included in the authentication tag calculation, the adversary can not know if her attack will succeed. Specifically, the probability that the receiver will detect an adversary modifying a data word will be equal to the probability that a given data word has been included in the authentication tag calculation. Although the probability of detection of a single modified data word within a sparsely-computed-upon message may be quite low, modification of multiple data words is exponentially more likely to be detected. Thus, high-speed network applications that can tolerate a few modified packets, such as video or audio streams, may perform quite satisfactorily using probabilistic authentication.

Although it would appear the use of multiple inner MACs would be applicable to probabilistic authentication as it was to MAC groups. This is not the case, however, since the security of probabilistic authentication is dominated by the probability that a data word contributes to the authentication tag computation. Since the use of multiple inner MACs incurs an overhead to determine which inner MAC a data word contributes, it effectively decreases the security of probabilistic authentication for a given speed. Therefore, probabilistic authentication will only be analyzed in terms of using a single inner MAC.

3 Analysis of the ACSA Model

3.1 Security

3.1.1 Overview

3.1.1.1 Analysis Assumptions

ACSA Methods and Implementation Known to Adversary - Security analysis of the model assumes that the adversary will be aware of the methods and specific implementation of ACSA on a target network device platform. ACSA does not rely on the obscurity of its methods or its implementation for any of its security.

Security of the Underlying Network Security Protocol - In general, analysis of ACSA security assumes that the underlying network security protocol(s) that would support an instantiation of ACSA are fundamentally sound. That is, that the mechanisms for confidentiality, authentication (other than that provided by ACSA), and key management provide requisite security.

3.1.1.2 Threats and Attacks

The overall security of ACSA depends on the strength of its gears and the inability of an adversary to determine which or whether a gear is applied to a given data word of a packet. The security goal, as previously stated, is not to provide impenetrable authentication for all applications in all environments, but rather an acceptable level of security for many ultra-fast network applications.

The assumed goal of the adversary is to tamper with one or more data words of a packet without detection. Such tampering may include modifying, deleting, inserting, padding, permuting, and/or replaying individual bits or many data words within a packet and/or authentication tag. To deter tampering, the authentication system should prevent the adversary from gaining sufficient information to achieve the primary goal, such as sensitive ACSA parameters, or that a weak gear or no gear is applied to a given data word. Additionally, the adversary should not be able to force the system into degrading security, nor disrupting the system, without detection.

This security analysis is intended to identify possible attacks on the ACSA gears and architecture, describe how the ACSA system deters these attacks, and provide heuristic evidence of the authentication security provided by ACSA. This analysis will not evaluate the particular authentication mechanisms used for individual gears, nor evaluate any component protocols (such as IPSEC or IKE) or cryptographic primitives.

3.1.2 Attacks on ACSA Authentication Types

3.1.2.1 Conventional Authentication Mechanisms

When ACSA is used in point-to-point relationships, the conventional authentication mechanisms normally used will be HMACs. HMACs are regarded as provably secure

provided the underlying hash functions used are at least weakly collision resistant [Bellare, Canetti, Krawczyk].

ACSA slightly alters the use of conventional HMACs to use a different authentication key for each packet. By changing the key for each packet, we prevent one of the more quantifiable attacks against HMACs. Unfortunately, from an analysis perspective, we are not left with any documented attacks against HMACs that are quantifiable. In the interest of quantifying HMAC security as shown in Figure 8, we'll make the conservative assumption that the probability of successfully forging an HMAC is equivalent to finding collisions in the key-less hash function used to construct the HMAC. Later revisions of this document will attempt to more accurately quantify the security of HMACs that are used in ACSA.

3.1.2.2 MAC Groups

The security of MAC groups is based on the same security principles as those for Nested MACs. Unlike strong Nested MACs, however, MAC groups may comprise algorithms that are not strongly collision resistant. The resultant security of MAC groups is dependent on several factors, the most important of which are: (1) the collision resistance of the inner MAC, (2) probability that a data word contributes to a given inner MAC, and (3) the number of data words that an adversary modifies. Although MAC groups are composed of multiple inner MACs, the most effective attack for an adversary is to find a collision assuming all modified data words contribute to the weakest inner MAC.

To analyze this attack, we must first determine the collision resistance of the inner MAC. If the inner MAC is composed of a keyed hash function, the collision resistance is based on (1) the collision resistance of the hash function, (2) the strength of the key, and (3) the method by which the key is combined with the hash function computation. If the key is merely concatenated with the message, the collision resistance of the inner MAC can be as weak as that of the key-less hash function. If, as [Bellare, Canetti, Krawczyk] suggest, the key is set to the hash function's initialization value, the strength of the key dominates the collision resistance of the resulting MAC.

If an adversary can find collisions in an inner MAC function, the adversary must also determine the data words that contribute to the inner MAC if she hopes to generate a collision of the MAC group tag. Since each data word pseudorandomly contributes to one of multiple inner MACs, the adversary's ability to guess which data words contribute to a given inner MAC decreases as the size of the message increases. For the size of messages for which MAC groups will be used, at least several hundred data words, an adversary's ability to guess the data words that contribute to a given inner MAC is vanishingly small.

It is possible that a hash function that belongs to a given inner MAC may have a weakness whereby an adversary can find a collision without knowing the entire subset of the message that contributes to the inner MAC. However, all of the *modified* data words would have to contribute to the inner MAC for such an attack to be successful. Thus, if an adversary performs many modifications, the probability that all of the modifications contribute to the inner MAC decreases exponentially.

Quantifying the security of MAC groups appears to be difficult for the same reasons HMAC security is difficult to quantify. In the interest of quantifying MAC group security as shown in Figure 8, we will make the conservative assumption that the probability of successfully forging a MAC group is equivalent to the greater of (1) finding collisions in the weakest of the key-less hash functions and (2) the probability that all of the modified data words contribute to the same weak inner MAC. Later revisions of this document will attempt to more accurately quantify the security of MAC groups.

3.1.2.3 Probabilistic Authentication

The chance of successful forgery when probabilistic authentication is used is approximately equal to the chance that none of the modified data words has been included in the authentication tag calculation. Although the strength of the inner MAC function may appear to have some impact, this effect is negligible when compared to the relative security of not including all data words into the authentication tag computation.

3.1.3 Attacks on ACSA Control Messages

3.1.3.1 Spoofing

Spoofing of ACSA control messages is prevented by protecting the ACSA control messages using strong confidentiality, authentication, and anti-replay services from the underlying network security protocol.

3.1.3.2 Denial of Service Attacks

There are several denial-of-service attacks available to the adversary with varying effectiveness. The general result of these attacks is degradation or prevention of performance, but no loss of data authentication security.

Deletion or modification of ACSA control messages – The deletion of control messages will cause different effects based on which control message is deleted. In this section we examine the effect of deletion or tampering on the six ACSA control messages.

- *ACSA Hello message* – Deleting or tampering with the initial ACSA Hello message sent from the sender to the receiver will likely cause the receiver to either not attempt to negotiate an ACSA session, or not to negotiate the security association altogether. If the receiver elects to continue negotiating the security association without ACSA, it is assumed conventional authentication mechanisms will be used, thus resulting in only degradation in performance, not security.
- *ACSA Parameter Reply message* – Deleting or tampering with the ACSA parameter sent from the receiver to the sender reply message during ACSA session establishment prevents the ACSA session from occurring. The sender, if properly implemented, should revert to the conventional authentication mechanisms of the underlying network security protocol, thus resulting in only degradation in performance, not security.
- *Session Establishment message* – Deleting or tampering with the Session Establishment message sent from the sender to the receiver prevents the receiver from authenticating received packets. The receiver will send a synchronization error

message to the sender and attempt to verify packets by using the base gear. If the sender receives the synchronization error message from the receiver, it will switch into the base gear and send a gear switch message to the receiver. If the sender does not receive a synchronization error message from the receiver within the heartbeat time interval, it will switch into the base gear and send a gear switch message to the receiver. Thus, the sender and receiver will revert to communicating using the base gear, resulting in a temporary degradation in performance, but not in security.

- *Gear Switch message* – Same effects as the Session Establishment message.
- *Heartbeat message* – Deleting or tampering with the Heartbeat message sent from the receiver to the sender will prevent the sender from receiving a valid Heartbeat message from the receiver within the heartbeat time interval. Thus, the sender will switch into the base gear and send a gear switch message to the receiver. Thus, the sender and receiver will revert to communicating using the base gear, resulting in a temporary degradation in performance, but not in security.
- *Synchronization Error message* – Deleting or tampering with the Synchronization Error message sent from the receiver to the sender will prevent the sender from receiving a valid message from the receiver within the heartbeat time interval. Thus, the sender will switch into the base gear and send a gear switch message to the receiver anyway.

3.1.4 Environmental Attacks

Since the ACSA controller determines gear suites and the current gear based on its environment, if an adversary can influence a sender or receiver's environment, it can possibly cause a weaker gear to be used. There are two main methods of deterring such an attack. First, if either of the communicants sets a policy to not use "weak" gears, any adverse environmental influence by an adversary will be limited to the gears permitted by policy. Second, ACSA platforms should protect their resources from tampering by an adversary or not use such resources for determining ACSA gears.

3.1.5 Timing Attacks

The currently selected gear will determine the authentication tag computation time and will possibly also affect the rate at which packets are sent and received. If the gear affects the network data rate (highly likely in some scenarios), it is likely an adversary will be able to deduce at least the approximate gear, if not the exact gear, that a sender is using to generate authentication tags. Although this fact may be undesirable from a security perspective, it is not clear the ACSA system can feasibly prevent the adversary from gleaning this gear information. Knowing the gear, however, is necessary but not sufficient for the adversary to attack the system.

Knowing the gear when the ACSA system is using a strong authentication mechanism will likely be of no value, except knowing when *not* to attack the system, since an attack will be detected. MAC groups have been designed such that even if the adversary knows the composition of the MAC group, it will not know which data words will contribute to which inner MAC functions, unless it knows the authentication key. The analysis of the effectiveness of attacking MAC groups shown in Section 3.1.2.2 assumes that the adversary has already obtained knowledge of the gear through some method

such as timing analysis. Timing analysis will be most effective against probabilistic authentication, since presumably the adversary will be able to then determine the relative chance of modifying one or more data words without being detected.

3.1.6 Replay Attacks

Replay attacks should be addressed by the underlying network security protocol, and generally fall outside the scope of ACSA. However, provided the packets are sequentially ordered and are associated with a sequence number, the ACSA system provides two mechanisms to deter replay attacks:

1. The key(s) used in the authentication tag calculation will be dependent on the packet sequence number, and
2. If the packet sequence number is included in the packet in a standard fashion, the ACSA implementation may be designed to always include the packet sequence number in the calculation of the authentication tag, even when using probabilistic authentication.

3.2 Performance

3.2.1 Performance-Security Tradeoff

Since the performance of ACSA can be set to any desired speed, it is only useful to discuss performance in the context of the performance-security tradeoff. To perform this analysis, we assume the security of the various authentication mechanism types is quantified based on the assumptions of Section 3.1.2. The performance of conventional mechanisms used in this analysis is shown in Figure 17. The performance of MAC groups are based on the dominant inner MAC function for a given security, also using the performance results shown in Figure 17. The performance of probabilistic authentication is based on decreasing the tag computation time commensurate with the decrease in the percentage of data words that are included in the tag computation for a given NMAC. All performance figures are based on measurements and estimates of speed of assembly language implementations on a 200 MHz Pentium (not Pentium Pro/II).

Figure 8 - Comparison of Speed versus Security for ACSA Authentication Mechanism Types

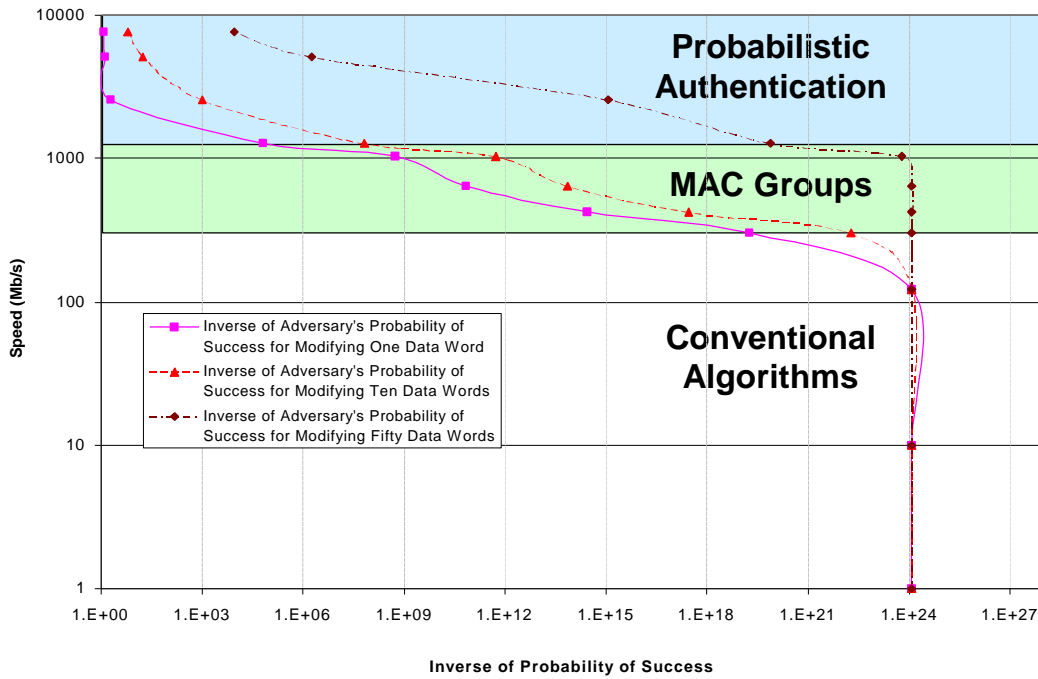


Figure 17 demonstrates the tradeoff that exists between the three authentication mechanism types for forgery attacks of one, ten and fifty data words. Conventional algorithms provide strong security for all three attacks, but with limited performance. MAC groups offer a speed gain up to a factor of five over the fastest conventional algorithms, but at the expense of a reduction in authentication security when the adversary only modifies a small number of data words. When many data words are modified, however, MAC group security is not necessarily much worse than conventional algorithms, if at all. Probabilistic authentication provides even greater performance, but at the expense of an even more drastic reduction in security, especially when the adversary only modifies a few data words. However, when the adversary does modify a large number of data words, even probabilistic authentication becomes a desirable tradeoff because of its performance superiority.

3.2.2 Other Performance Factors

3.2.2.1 Packet Size

For both MAC groups and probabilistic authentication implementations, there are a few operations that will be performed regardless of packet size. Specifically, the calculation of the packet-specific authentication key and the Outer MAC computation must both be performed once for all packet sizes. Furthermore, specific ACSA implementations may elect to include all of the packet header information in the probabilistic authentication computation, regardless of the underlying percentage of authenticated data words being used for application data.

In general, the packet sizes of interest to ACSA are in the tens of thousands of bytes, since ACSA is intended to primarily address ultra-fast network applications. For packets of this size, the contribution of these size-independent computations is relatively small. Although ACSA functions for any packet size, its performance gains will not be dramatic for smaller packet sizes.

3.2.2.2 Implementation

Certain implementation factors may significantly effect ACSA performance, and thus the ACSA performance-security tradeoff. For instance, there may be certain performance advantages to a processor performing ACSA authentication computations on the entire packet at once, as opposed to functioning in a streaming data mode. Also, some processors may perform some gears relatively faster than other processors, resulting in a different ordering of gears for the performance-security tradeoff. Real world ACSA implementations using heterogeneous platforms may exhibit behavior that alters the existing performance-security tradeoff.

3.3 Optimization of Model Parameters

3.3.1 Optimal Number of Gears

Increasing the number of gears available to ACSA communicants generally makes it more difficult for an adversary to determine the current gear, thus making a forgery attack more difficult. Since an adversary may be able to detect the current gear via timing analysis, its not clear how valuable more gears might be.

Conversely, the cost of including more gears may not be very great either. From a practical point of view, the cost of more gears will likely be based on the development costs of standardizing, implementing and interoperability testing different gears. For any sizeable distribution of ACSA on heterogeneous platforms, the number of gears will likely be limited to a few at most.

3.3.2 Optimal Number of Tags in a MAC Group with Subset Tags

There are generally three types of tags that are desirable for a receiver to authenticate:

1. A tag computed using all of the data words of the message.
2. A tag computed using at least half of the data words of the message.
3. A tag computed using no more than half of the data words of the message.

Since MAC groups containing two inner MACs can generate these three tags with only two extra outer MAC computations, the added computational cost for the sender seems relatively small. However, providing more than two subset tags would further increase the required authentication tag size for all gears, impacting the performance of all gears.

Therefore, the optimal MAC group composition when subset tags are used is one tag for the entire message and two subset tags, one of which is computed over at least half of the data words of the message.

3.4 Other Considerations

3.4.1 Applicability

As was discussed in the performance analysis, ACSA is not particularly beneficial for short messages, but rather beneficial to ultra-fast applications such as audio or video where modification of a single data word is not important. Although ACSA targets ultra-fast application where undetected modifications of a few data words may be tolerable, ACSA is flexible enough to satisfy general network authentication needs.

3.4.2 Implementation Issues

Issues that should be considered when planning to implement ACSA are discussed below.

3.4.2.1 Need channel for new control messages

The underlying network security protocol should be sufficiently extensible to allow ACSA control messages to be added. Furthermore, the network security protocol should be able to provide confidentiality and authentication protection of these messages.

3.4.2.2 Complexity of MAC Groups and Probabilistic Authentication

Both the MAC groups and probabilistic authentication methods are relatively complex and will require significant software development efforts to operate in a reliable manner.

3.4.2.3 Need to Implement Code in Assembly Language

Reaping the performance gains of ACSA will require much of the code added to the Network Security Service module be implemented in the target processor's assembly language. Higher-level languages such as C, C++ and Java will likely be unable to provide sufficiently efficient implementations to be worthwhile. The necessity of implementing these modules in assembly language adds an additional cost to the software development of ACSA implementations.

Appendix A Candidate Authentication Mechanisms

A.1 Block Cipher Based MACs

MACs use a secret key known by the sender and receiver to generate a value to assure the message has not been tampered with. There is no source authentication (in a multicast environment, there is no way to tell who sent the message).

A.1.1 DES-MAC

Functional

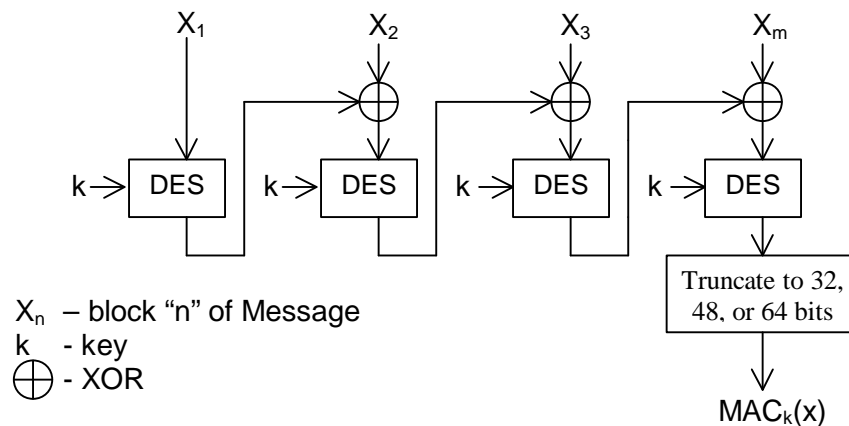


Figure 9 - DES-MAC

The DES-MAC is described in the ANSI X9.9 and X9.19 standards.

Performance - The speed is approximately 170 clock cycles per 32 bit word [Bosselaers] The only setup time is for rekeying and is usually negligible over even moderate amounts of data.

Block Size and Pad Requirements - The block size is 64 bits with the last block left justified and padded with 0 value bits [ANSI X9.9].

Security - This MAC is provably secure with mediocre security bounds [Rogaway96][Bellare, Kilian, Rogaway]. The probability of collision is less than or equal to $3 \cdot 2^{-l}$ where "l" is the length of the output in bits [Bellare, Kilian, Rogaway].

A.1.2 Other Block Cipher Based MACs

These have a similar structure to the standard DES-MAC.

Functional

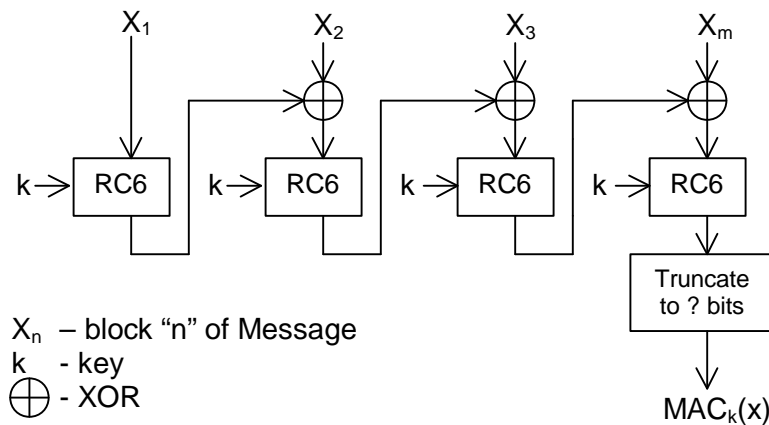


Figure 10 - Block Cipher MAC

A block cipher is run in CBC mode and a part of the last encrypted block is output.

Performance - Any block cipher can be used. One of the fastest strong block ciphers is RC6. RC6 computes at approximately 73 clock cycles per 32 bit word [Gladman] or approximately three times faster than the DES-MAC. RC6 performs even faster if the decrypt operation is used instead of the encrypt operation—increasing performance to approximately 61 clock cycles per 32 bit word [Gladman]. RC6 setup time is about 12,000 clock cycles, but only has to be done at re-keying. After less than 20K of data, setup overhead contributes less than 5% of the total time (<.00025 seconds at gigabit speeds).

A 3DES cipher based MAC runs at about 465 clock cycles per 32 bit word [Bosselaers]. Senders and receivers perform the same amount of work.

Block Size and Pad Requirements - The block size is that of the cipher. There are no pad requirements defined.

Security – A block-cipher based MAC is as secure as the block cipher used [Bellare, Kilian, Rogaway]. 128-bit block-sized ciphers such as RC6 hold an advantage over 64-bit block ciphers like DES because the forging probability is a function of $2^{-(\text{bits per MAC})}$. For a DES-MAC with a 32-bit output, an exhaustive attack reduces the key space to a relatively small 2^{24} possibilities. For a 96 bit RC6-MAC, an exhaustive attack, which will take 2^{64} more trials, reduces the key space to 2^{32} possibilities. In either case, a second text-MAC pair almost certainly determines a unique MAC key [Menezes, van Oorschot, Vanstone].

The ANSI X9.9 and X9.19 standards give a range of truncations for the DES-MAC with no recommendation. Without truncation, it becomes easier to extend a CBC-MAC with one or more zero value message blocks [Bellare, Kilian, Rogaway].

The probability of collision is less than or equal to $3 \cdot 2^{-l}$ where "l" is the length of the output in bits [Bellare, Kilian, Rogaway].

A.1.3 Pre-computed stream cipher based XOR MAC

Functional - A block cipher (RC6 here) is run in counter mode to produce a pseudo random permutation (PRP). This PRP can be pre-computed and cached for use when the message arrives for authentication.

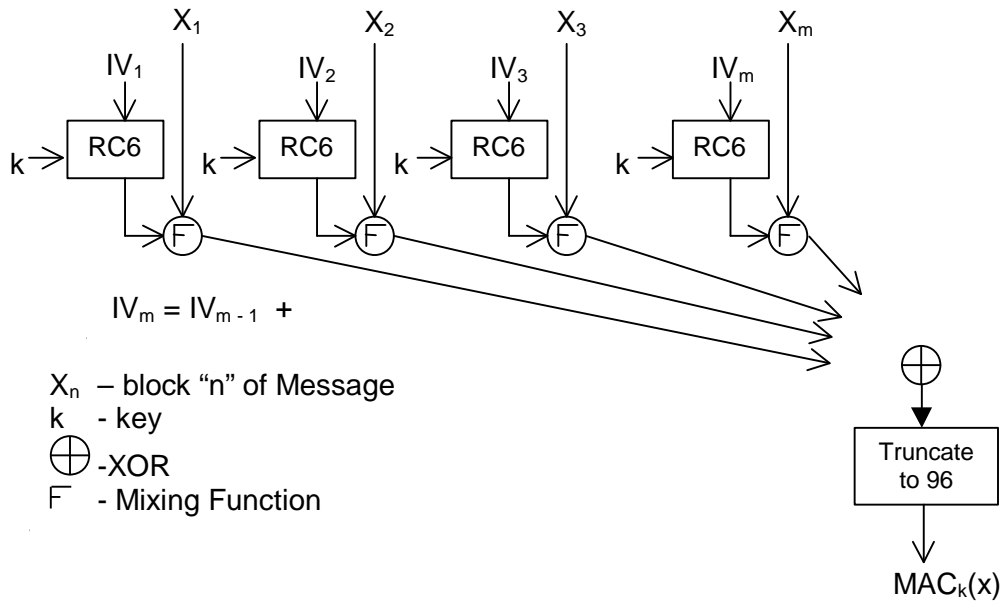


Figure 11 - Pre-computed Cipher XOR MAC

Performance - RC6 is used for its performance [Gladman]. If F is an XOR, this gives just 4 clock cycles per 32 bit word (unoptimised). That sounds great until you include the setup time. The key setup we will ignore because that becomes negligible before even 100K is authenticated. There is about 72 clock cycles per word using RC6 in encrypt mode, 59 clock cycles in decrypt mode.

This is comparable (very slightly slower) than the block cipher CBC based MAC mode. The advantage comes in if you have a CPU with a varying duty cycle such that the stream to XOR with the plaintext can be computed before the plaintext arrives. The other advantage here is in the ability to run the computations in parallel in multiprocessor environments.

When running in a machine with varying duty cycles, the cached stream size becomes an issue. The cached size must be large enough to make it worthwhile to use this method. Emptying the cache will cause a performance penalty which will cause this method to degrade to slightly slower than the normal cipher MAC variant.

Block Size and Pad Requirements - The block size is that which F requires. There are no pad requirements defined.

Security - Using an XOR for F , the security of this method is very poor (swapping any two blocks results in the same MAC) This can be fixed using a function that does better mixing than XOR. The problem is that functions of this nature take more clock cycles. The simplest " F " that would probably be good enough (no analysis has been done) is a multiply. Faster yet (on most machines) would be an add. Another option could be to keep a 1-KByte table (256 by 32-bit, or a 2K 256 by 64-bit) which is left shifted one byte

for every byte read from the message - the message byte would be used as an index to lookup in the table.

Speed of the multiply method would probably similar to the MMH at 6.2 clock cycles per 32-bit word. The speed of the table lookup would probably similar to the AHA which approaches 5 clock cycles per 32-bit word.

A.2 Inner MAC Function Candidates

A.2.1 SHA-1, MD5, MD4

Performance - SHA-1 runs in approximately 52 clock cycles per 32 bit word, MD5 in 21 clock cycles per 32 bit word, MD4 in 15 clock cycles per 32 bit word [Bosselaers].

Block Size and Pad Requirements - The block size for these algorithms is 512 bits. The message is padded by first appending a 1 value bit and then the minimum number of 0 value bits to ensure that its length in bits plus 448 is divisible by 512. A 64-bit binary representation of the original length of the message is then concatenated to the message [RSA 3.3.6][Schenier].

Security - There are methods of finding collisions in MD4 that take under an hour on a PC (type not specified but 1996 vintage). The limitations of the collisions are that small specific changes can be made within the first 512-bit block (or any block where the current state is known) if there is a larger area in that block that can be modified more extensively. It looks promising that these attacks can be extended to MD5. It is much less likely that these attacks can be extended to SHA-1 [Dobbertin].

A.2.2 Bucket Hash

Functional

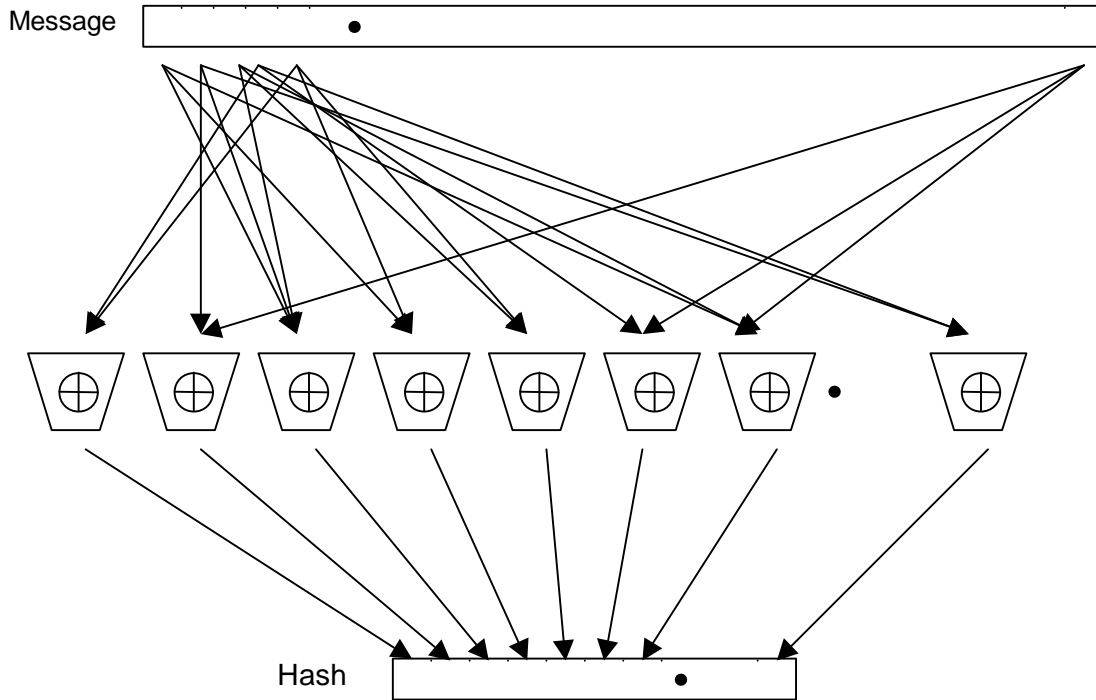


Figure 12 - Bucket Hash

Each block of the message is XORed into 3 buckets. The resulting bucket values are then concatenated to form the hash.

Performance - The bucket hash runs between 4.2 and 14 clock cycles per 32 bit word. 4.2 clock cycles per 32 bit word happens for smaller hash sizes using self modifying code. Real world hash sizes with non-self modifying code will likely execute at 10 clock cycles per 32 bit word.

Block Size and Pad Requirements - The block size is the same as the bucket size. The padding requirements are not defined.

Security - The security is based upon the collision probability which works out to about $\frac{(N*(N-1)*(N-2))}{(((N*(N-1)*(N-2)) - 36)) * 3312N^6}$ where "N" is the number of buckets. The following figure, Figure 13, shows a graph of collision probability versus number of buckets. Super imposed on that graph is a work estimate (in imaginary units). This work estimate is based upon the output length of the bucket hash for various lengths of input message.

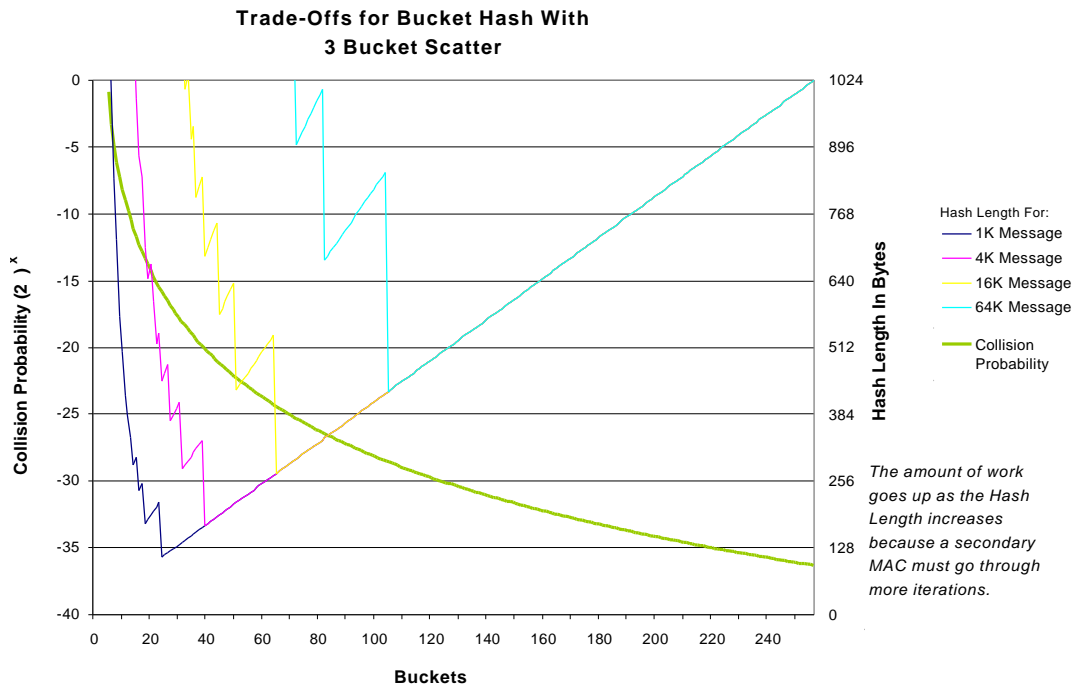


Figure 13 - Bucket Hash Tradeoffs

A.2.3 Alternate Hash Algorithm

Functional

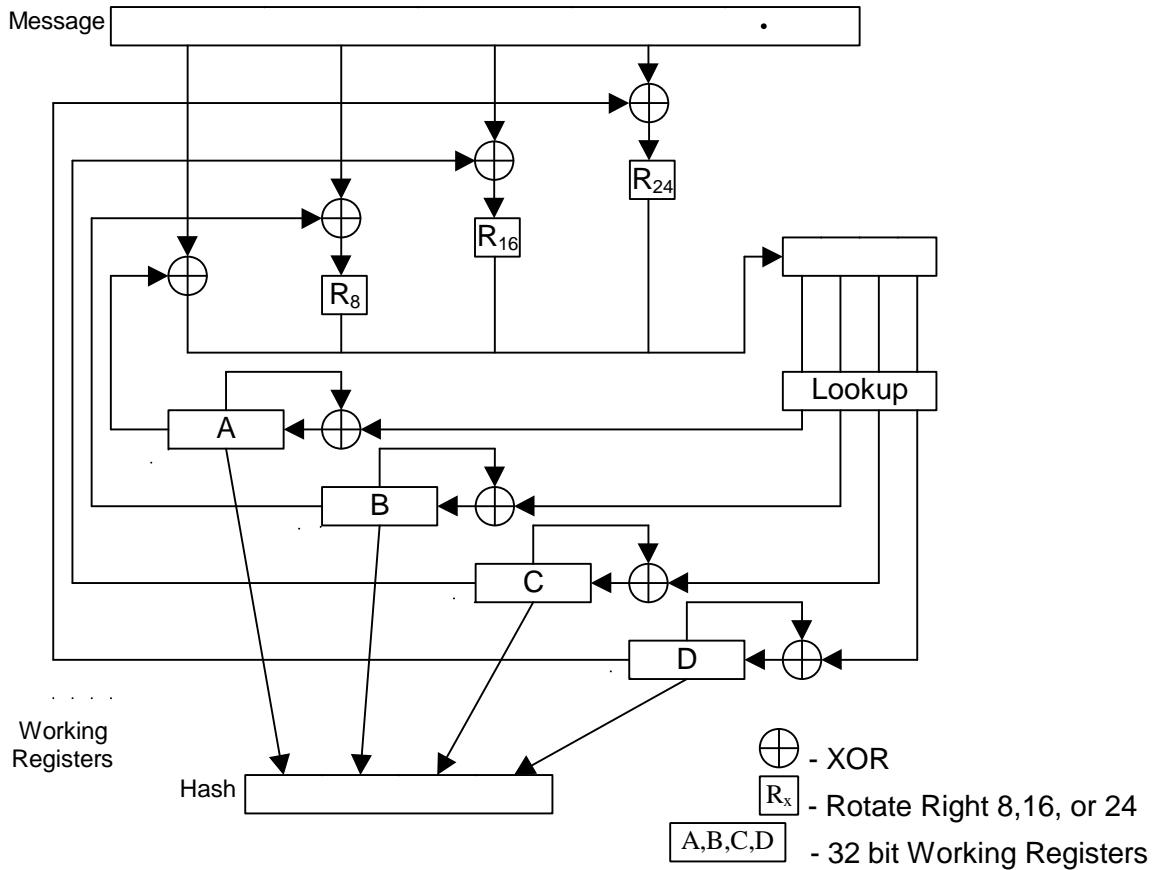


Figure 14 - Alternate Hash Algorithm

AHA processes the message in 32-bit chunks. The first word is processed first, the second word is processed second, the third word is processed third, etc. A 32-bit word is pulled from the message, XORed with A, B, C, or D, possibly rolled and each byte of the word is used in a table lookup to update A, B, C, and D. The final hash is A, B, C, and D concatenated. In this way, each byte of the message contributes to 32 bits of the output through the table lookup, and then possibly through the whole output if the message is long enough that the 32-bit value that that byte contributed to is used in another table lookup.

Performance - Using speed relative to MD5 and the performance of MD5 on a Pentium, AHA performance is approximately 5 clock cycles per 32-bit word. [Naftulin, Fishkin].

Block Size and Pad Requirements - The block and padding requirements are not defined. The pseudo-code implementation in this document requires a 32-bit block size.

Security - This is not provably secure but is theory grounded. [Rogaway96]

A.2.4 Multilinear Modular Hash and Non-linear Modular Hash

Functional - Multilinear Modular Hashing and Non-Linear Modular Hashing are actually MACs based on a small amount of computations.

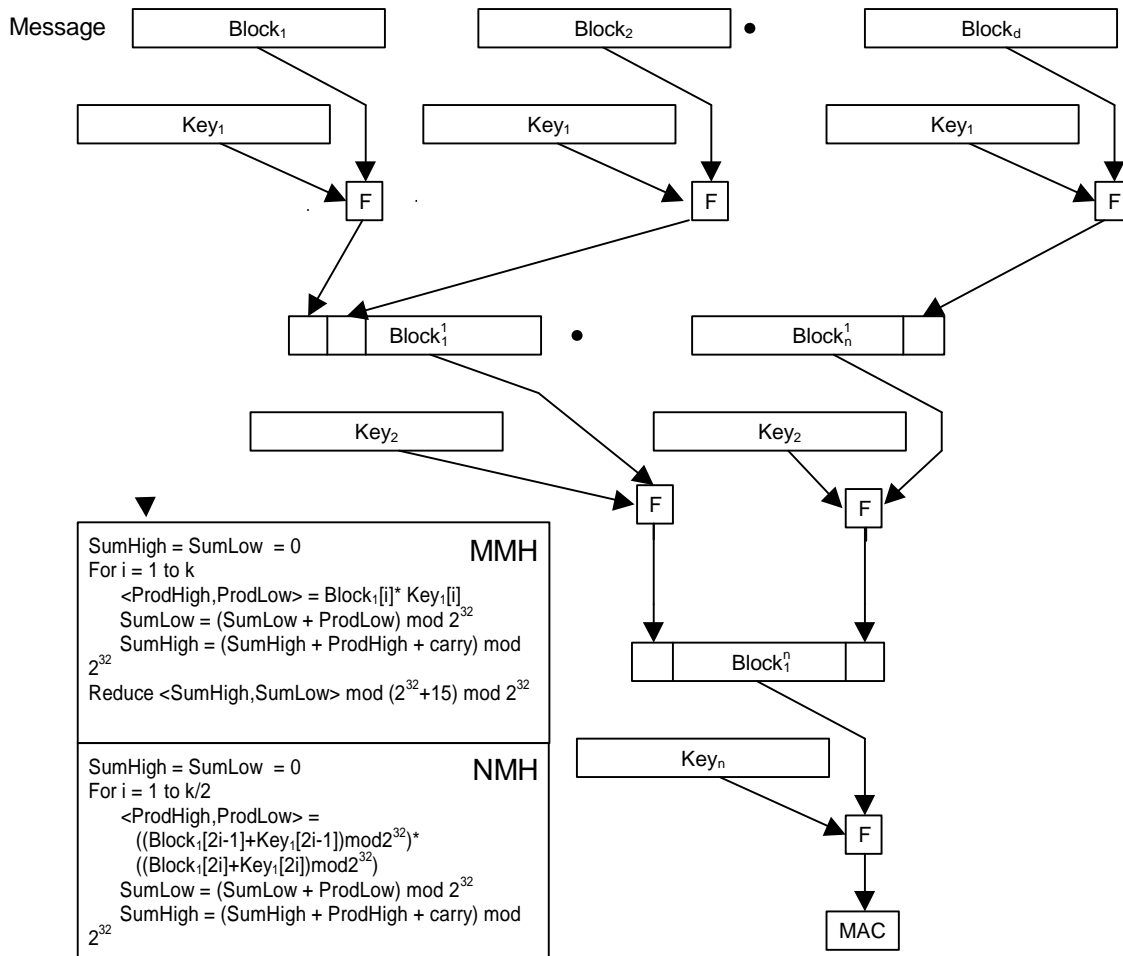


Figure 15 - Multilinear Modular Hash and Non-linear Modular Hash

Performance - In a test case, there was 6.1 clock cycles per 32 bit word for Multilinear Modular Hashing when the key and the message are in the cache and the block size is 32 words. NMH substitutes an add for a multiply every other time, so worst case should be 5.9 clock cycles per 32 bit word.

As evidenced by Figure 15, the key size is based upon the height of the tree and the size of the blocks. Actually the key size is based on **blockSize*ROUND_UP(log_{blockSize}(messageSize))** where **blockSize** is the number of (32 bit) words in a block and **messageSize** is the number of (32 bit) words in a message. Generally, as block size increases, key size increases (there is a minimum at e, 2.718..., but that is too small to worry about for other reasons). Computation size decreases as the block size increases because the 15 clock cycle modular reduction occurs once per block. The security also may also increase as the block size increases because the probability of collision increases with the height of the tree, which is less for larger block sizes.

The key will have to be created in a pseudo-random function from a seed key. The amount of key material is not known at initial setup (it depends on message length). Needed key material can either be derived as needed or the tree can be limited to two levels with a different hash used on the, now much reduced ($1/1024^{\text{th}}$ for 32 word block sizes), remaining material [Halevi and Krawczyk].

Block Size and Pad Requirements - The block and padding requirements are not defined. The pseudo-code implementation in this document requires a 32-bit block size.

Security - MMH and NMH are ϵ -AXU (ϵ - almost XOR Universal). The probability of collision is $1.5/2^{30 * (\text{height of tree})}$ [Halevi and Krawczyk].

A.2.5 Cryptographic CRC

Functional

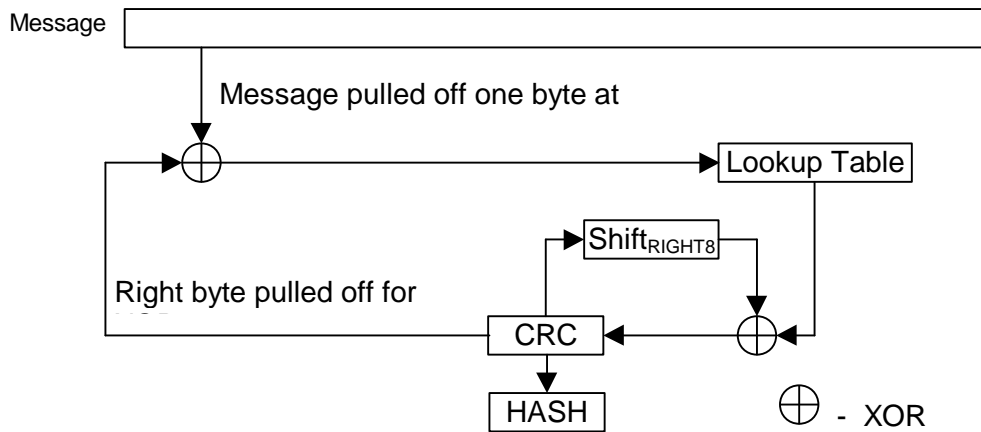


Figure 16 - Cryptographic CRC

The “Table” is built from a secret irreducible polynomial in GF(2). It has 256 entries that are the width of the desired HASH.

Performance – The usage of a table lookup is similar to that of the AHA algorithm, thus, it should run at a similar speed for a 32 bit CRC. The table must be built prior to running the CRC. An irreducible polynomial in GF(2) can be found in O(2) using mostly XOR and SHIFT operations [Krawczyk].

Block Size and Pad Requirements - The block size is 8 bits. Pad requirements are not defined.

Security - The probability of collision for a CRC hash is $(m+n)/(2^{n-1})$ where “m” is the maximum size in bits of the fake message inserted by the adversary and “n” is the CRC length in bits [Krawczyk]. For a CRC of 32 bits, the probability of collision is about 2^{-12} for a 64K-byte message. Increasing the CRC size to get more security will increase the processing time. This amount is not known even for an increase to 64 bits.

A.2.6 Keyed Pseudo-Random-Function (PRF)

Functional

$PRF(K,X) = F_n(K_n, X_n, F_{n-1}(K_{n-1}, X_{n-1}, F_{n-2}(K_{n-2}, X_{n-2}, \dots)))$;

$K = \text{Key} = (K_1, K_2, K_3, \dots, K_n)$;

$X = \text{Message} = (X_1, X_2, X_3, \dots, X_n)$;

$F = \text{Fast simple function} = 4 \text{ clock cycles per } 32 \text{ bit word or less. May have collision resistance of less than } 2^{-10}$.

The "HASH()" in this case is a series of short (1-3 clock cycle) functions on the source. These functions may include:

- 1) Copy
- 2) XOR of
 - a) Input
 - b) Rotate of input
 - c) Table lookup of input

The operations go into a buffer that is the block size for "E_k()"(for example: 512 bits for MD5).

Performance - The object of this MAC is to reduce the computation time to less than any keyed hash method such as AHA.

Block Size and Pad Requirements - The block and padding requirements are not defined.

Security - This MAC needs to be only weakly collision free to be the inner part of a strong HMAC.

A.3 Performance Estimates For Selected Algorithms

Authentication	Cycles per 32 bit word	Mbits per second on 200MHz Pentium	Ratio
Pre- computed cipher	4.00	1600.00	1.25
Keyed Psuedo Random Function	4.50	1422.22	1.111
AHA	5.00	1280.00	1
NMH	5.87	1090.29	0.852
MMH	6.20	1032.26	0.806
Bucket	10.00	640.00	0.5
MD4	15.00	426.67	0.333
MD5	21.00	304.76	0.238
Panama	25.00	256.00	0.2
SHA- 1	52.31	122.35	0.096
RC6- MAC	61.00	104.92	0.082
DES- MAC	170.00	37.65	0.029

Table 1 - Performance Estimates for Selected Algorithms

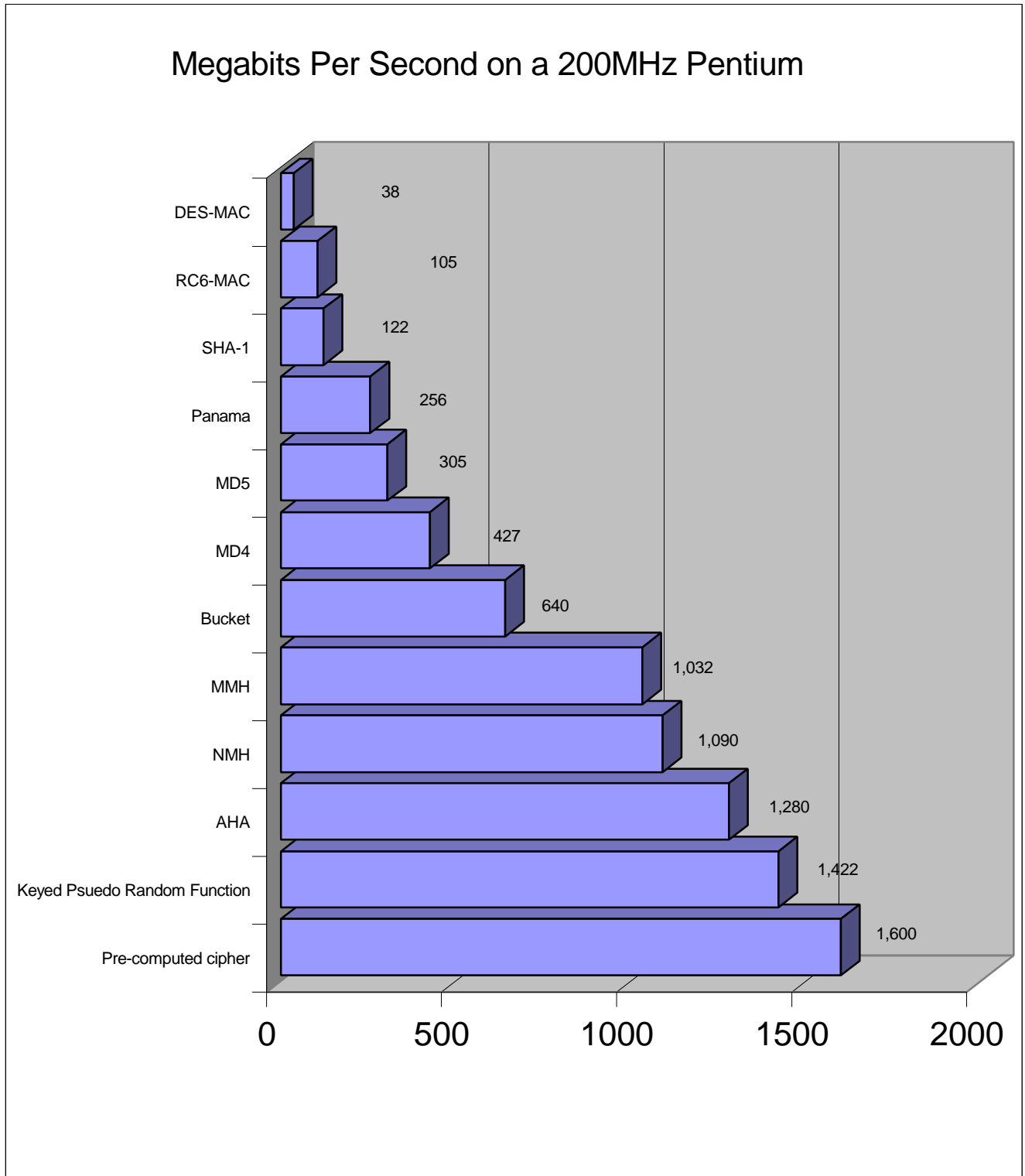


Figure 17 - Performance Estimates for Selected Algorithms

A.4 Pseudo Code

A.4.1 ACSA-HMAC

```
// return 96 bit sha-1 hash of gears
asca_hmac(key, buf, buflen, gears, res)
  for i = 0 to key.gears.Max
    gears.do[i](gears.info[i], buf, buflen)
  end for
  sha_1_96(key, gears.res, gears.reslen, res)
  return
```

A.4.2 DO SHA-1

```
//
// "gear.res" is 160 bits
//
sha1_do(gear, buf, buflen)
  i = key.offset[0]
  J = 0
  k = 0
  while i < buflen
    gear.buf[j] = buf[i]
    j = j + 1
    if j > gear.bufsize
      sha1.update(gear.buf)
      j = 0
    end if
    i = key.offset[k]
    k = k + 1
  end while
  sha1.result(gear.buf, j, gear.res) // load gear.res
  return
```

A.4.3 DO Bucket Hash

```
//
// "gear.res" is all the "bucket.val"s concatenated
//
bucket_hash_do(gear, buf, buflen)
  foreach bucket in gear.bucket
    i = 0
    forever
      if bucket.offset[i] >= buflen
        break
      end if
      bucket.val = bucket.val xor buf[bucket.offset[i]]
    end forever
  end foreach
  return
```

A.4.4 DO AHA Hash

```
//
// "gear.res" is 4 32 bit words (128 bits)
//
// Note, this implementation is different from the Rogaway
// implementation in that it allows the buffer to be a
// multiple of one 32 bit word as opposed to a multiple
// of 4 32 bit words. Is this valid?
```

```

//
aha_hash_do(gear, buf, buflen)
  i = j = 0
  while gear.offset[i] < buflen
    x = gear.res[j] xor buf[gear.offset[i]]
    gear.res[j] = gear.res[j] xor table[x and 0xFF]
    j = (j + 1) mod 4
    x = x rshift 8
    gear.res[j] = gear.res[j] xor table[x and 0xFF]
    j = (j + 1) mod 4
    x = x rshift 8
    gear.res[j] = gear.res[j] xor table[x and 0xFF]
    j = (j + 1) mod 4
    x = x rshift 8
    gear.res[j] = gear.res[j] xor table[x and 0xFF]
    j = (j + 2) mod 4
    i = i + 1
  end while

```

A.4.5 Do MMH Mac

```

//
// "gear.res" is a 32 bit value
//
do_mmh_mac(gear, buf, bufLen)
  index = 0
  maxLevel = 0
  forever
    //
    // Fill up the lowest block from the buffer
    //
    for block[0].index = 0 to (block[0].blockSize - 1)
      if gear.offset[index] >= bufLen
        break
      end if
      block[0].buf[block[0].index] = buf[gear.offset[index]]
      index = index + 1
    end for
    //
    // Check if done reading message
    //
    if block[0].index != block[0].blockSize
      break
    end if
    //
    // Do the bottom reduction and any reductions up the
    // tree as necessary
    //
    level = 0
    forever
      block[level + 1].buf[block[level + 1].index] =
        mmh_reduction(gear.key[level],
          block[level].buf,
          block[level].index - 1)
      if block[level + 1].index < gear.blockSize
        block[level + 1].index = block[level + 1].index + 1
        break;
      else
        level = level + 1
        if maxLevel < level
          maxLevel = level
        end if
      end if
    end if
  end if

```

```

    end forever
end forever
//
// Do up to the final reduction
//
for level = 0 to maxLevel - 1
    block[level + 1].buf[block[level + 1].index] =
        mmh_reduction(gear.key[level],
                      block[level].buf,
                      block[level].index + 1)
    block[level + 1].index = block[level + 1].index + 1
end for
//
// Do the final reduction
//
gear.res =
    mmh_reduction(gear.key[maxLevel],
                  block[maxLevel].buf,
                  block[maxLevel].index)

return

//
// mmh reduce function, returns a 32 bit value
//
mmh_reduction(key, data, len)
    sumHigh = sumLow = 0
    For i = 0 to (len - 1)
        <prodHigh, prodLow> = data[i] * key[i]
        sumLow = (sumLow + prodLow) mod 2^32
        sumHigh = (sumHigh + prodHigh + carry) mod 2^32
    return Reduce <sumHigh, sumLow> mod (232+15) mod 2^32

```

A.4.6 Do NMH Mac

The “do_nmh_mac()” function is similar to the “do_mmh_mac()” function except that it calls the “nmh_reduction()” instead of the “mmh_reduction()” and must assure that the size of the data passed to the reduction function is an even number of 32 bit words, padding with 0 as necessary.

```

nmh_reduction(key, data, len)
    sumHigh = sumLow = 0
    for i = 0 to ((len/2) - 1)
        <ProdHigh, ProdLow> =
            ((data[2i-1] + key[2i-1]) mod 2^32) *
            ((data[2i] + key[2i]) mod 2^32)
        sumLow = (sumLow + prodLow) mod 2^32
        sumHigh = (sumHigh + prodHigh + carry) mod 2^32
    Reduce <sumHigh, sumLow> mod (2^32 + 15) mod 2^32

```

A.4.7 Keyed Pseudo-Random-Function (PRF) MAC

```

//
// "gear.res" is concatenation of all "bucket.val"s
//
do_prf_mac(gear, buf, bufLen)
    foreach bucket in gear.bucket
        // initial load
        bucket.val = buf[bucket.initindex]
        // xor
        j = 0
        while bucket.xorindex[j] < bufLen
            bucket.val = bucket.val xor buf[bucket.xorindex[j]]

```

```

        j = j + 1
    end while
    // table lookup
    j = 0
    while bucket.xorindex[j] < bufLen
        bucket.val = bucket.val xor table[BYTE0(buf[bucket.tabindex[j]])]
        bucket.val = bucket.val xor table[BYTE1(buf[bucket.tabindex[j]])]
        bucket.val = bucket.val xor table[BYTE2(buf[bucket.tabindex[j]])]
        bucket.val = bucket.val xor table[BYTE3(buf[bucket.tabindex[j]])]
        j = j + 1
    end while
    // rotate
    j = 0
    while bucket.xorindex[j] < bufLen
        bucket.val =
            bucket.val xor rot(buf[bucket.rotindex[j]], bucket.rotval)
        j = j + 1
    end while
end foreach

```

A.4.8 Do CRC Hash

```

//
// "buf" is "bufLen" octets long
//
do_crc(buf, bufLen)
    crc = 0xFFFFFFFF
    for i = 0 to bufLen - 1
        crc = table[(crc XOR buf[i]) AND 0xff] XOR (crc RIGHTSHIFT 8)
    end for
    return crc

```

A.4.9 Do Panama Hash

```

// similar to sha-1 pseudo code
// pseudo code for the Panama Hash function itself is not ready yet.

```

Appendix B Acronyms

ACSA	Adaptive Cryptographically Synchronized Authentication
AES	Advanced Encryption Standard
AH	Authentication Header
AHA	Alternate Hashing Algorithm
CPU	Central Processing Unit
DES	Data Encryption Standard
ESP	Encapsulating Security Payload
HMAC	Hash-based Message Authentication Code
ICV	Integrity Check Value
IETF	Internet Engineering Task Force
IKE	Internet Key Exchange
IPSec	Internet Protocol Security
ISAKMP	Internet Security Association and Key Management Protocol
MAC	Message Authentication Code
MD5	Message Digest algorithm 5
MMH	Multilinear Modular Hashing
NMAC	Nested Message Authentication Code
NMH	Non-Linear Modular Hashing
SA	Security Association
SHA-1	Secure Hash Algorithm
TIS	Trusted Information Systems

Appendix C Glossary

ACSA control messages – ACSA-specific messages that contain ACSA parameter or status information.

ACSA protocol – A set of messages that allow two (or more) ACSA participants to communicate ACSA control information securely.

ACSA system – The set of components and interactions that provide high-speed authentication between network devices using any flexible network security protocol.

Application - The actual network application that requires authentication services.

Authentication tag – A checksum value that is associated with a packet of data in order to provide connectionless integrity and data origin authentication.

Base gear – The “fallback”, low-risk gear that two ACSA communicants switch to whenever a synchronization error is encountered.

Controller – An ACSA component which determines the gear and maintains synchronization between sender and receiver.

Data word – A chunk of data the size of the CPU machine size—assumed to be 32 bits in much of our analysis.

Desired gear – The gear that satisfies the risk-performance tradeoff for a given application for a local node’s conditions.

Gear – An authentication algorithm. The definition of a gear may include the probability that a given data block will be authenticated.

Gear switch point – The packet in which the latest ACSA gear is used instead of the previous gear.

Heartbeat message – A secure ACSA control message sent from an ACSA-capable receiver to an ACSA-capable sender. The Heartbeat message helps ensure gear synchronization between the communicants and allows provides for routine reporting of authentication errors and other status information.

Inner MAC – The MAC function that is applied to the message in a Nested MAC computation.

Latest gear – The newest gear being used by the ACSA system. This gear will replace the previous gear at the gear switch point.

MAC group – A method of authentication comprising two or more inner MACs that feed a single outer MAC.

Message – A set of data associated with a single authentication tag. This term is used interchangeably with packet.

Nested MAC – A MAC function where the output of one MAC is fed into the input of another MAC. The output of the second MAC is the output of the Nested MAC.

Network defenses – Firewalls, intrusion detection devices, and the like which may detect an adversary’s attack on the host or network.

Network Policy Manager – An ACSA-specific interface that allows a network administrator to define a set of performance and/or risk factors that may be used to improve the node's performance-risk tradeoff.

Network Security Services module – A module responsible for apply authentication security services to network application data.

Node – A sender or receiver network device.

Outer MAC – The MAC function that is applied to the output of the inner MAC in a Nested MAC computation.

Packet – A set of data associated with a single authentication tag. This term is used interchangeably with message.

Pairwise gear suite – The set of gears common to a sender-receiver pair.

Pairwise session parameters – ACSA session parameters used for ACSA operations for the duration of the security association.

Previous gear – The gear applied to packets prior to the gear switch point.

Probabilistic authentication – A method of authentication where only a subset of the entire message is used to compute the authentication tag.

Security association – A simplex (uni-directional) logical connection, created for security purposes. All network application data traversing an SA is provided the same security processing.

Security Association and Key Management module – A module responsible for establishing, negotiating, modifying and deleting security associations.

Tree MAC – A construction of a MAC where layers of MACs feed into other layers of progressively less MACs which include contributions from progressively more of the message.

Appendix D References

- [Bellare, Canetti, Krawczyk] M. Bellare, R. Canetti and H. Krawczyk, *Keying hash functions for message authentication*, Advances in Cryptology – CRYPTO '96 Proceedings, LNCS 1109, N. Koblitz ed., Springer Verlag, 1996.
- [Bellare, Kilian, Rogaway] M. Bellare, J. Kilian, P. Rogaway, *The Security of the Cipher Block Chaining Message Authentication Code*, July 1997.
- [Bosselaers] A. Bosselaers, *Fast Implementations on the Pentium*, <http://www.esat.kuleuven.ac.be/~bosselae/fast.html>.
- [Daemen, Clapp] J. Daemen and C. Clapp, *The Panama Cryptographic Function*, December 1998, Dr. Dobb's Journal.
- [Dai] W. Dai, *Benchmarks*, <http://www.eskimo.com/~weidai/benchmarks.txt>.
- [Dobbertin] H. Dobbertin, *Cryptanalysis of MD4*, Fast Software Encryption, LNCS 1039, 1996 by Springer-Verlag, pp. 53-69.
- [Gladman] B. Gladman, *AES Algorithm Efficiency*, <http://www.seven77.demon.co.uk/aes.htm>.
- [Halevi and Krawczyk] S. Halevi and H. Krawczyk, *MMH: Software Message Authentication in the Gbit/Second Rates*, Fast Software Encryption, 1997 by Springer, pp. 172-188.
- [Krawczyk] H. Krawczyk, *LFSR-based Hashing and Authentication*, Advances In Cryptography – CRYPTO '94, 1994 by Springer-Verlag, p. 129.
- [Menezes, van Oorschot, Vanstone] A. Menezes, P. van Oorschot, S. Vanstone, *Handbook of Applied Cryptography*, 1997 by CRC Press, Inc., p. 354.
- [Naftulin, Fishkin] H. Naftulin and A. Fishkin, message authentication research, <http://wwwcsif.cs.ucdavis.edu/~fishkin/md5>.
- [Nahum, et.al] E. Nahum, S. O'Malley, H. Orman, and R. Schroepel, *Towards High Performance Cryptographic Software*, Technical Report TR 95 04, Dept. Computer Science, University of Arizona, 1995.
- [Rogaway⁹⁵] P. Rogaway, "Bucket hashing and its application to fast message authentication", Advances in Cryptology – CRYPTO '95 Proceedings, Lecture Notes in Computer Science Vol. 963, D. Coppersmith ed., Springer Verlag, 1995.
- [Rogaway⁹⁶] P. Rogaway, *Design and Analysis of Message Authentication Codes*, The 1996 RSA Data Security Conference, January 19, 1996.
- [Rogaway⁹⁷] P. Rogaway, *Bucket Hashing and its Application to Fast Message Authentication*, October 13 1997.
- [RSA 3.3.6] *What Are MD2, MD4, and MD5?*, <http://www.rsa.com/rsalabs/faq/html/3-6-6.html>.
- [Schenier] B. Schenier, *Applied Cryptography*, Second Edition, 1996, John Wiley & Sons, Inc., p. 442.
- [Shoup] V. Shoup, *On Fast and Provably Secure Message Authentication Based on Universal Hashing*, Bellcore, 445 South St., Morristown, NJ, December 4 1996.

<http://www.cs.wisc.edu/~shoup/papers/macsp.ps.Z>. A preliminary version of this paper appears in Proc. Crypto '96 pp. 313-328.

[Wegman, Carter] M. Wegman and L. Carter, *New Hash Functions And Their Use in Authentication and Set Equality*, Journal of Computer and System Sciences 22 1981, pp. 265-279.