

# Adaptively Trading Off Strength and Performance in Network Authentication

RSA Conference 2000  
January 19, 2000

David W. Carman  
Principal Cryptographic Engineer  
Cryptographic Technologies Group  
NAI Labs, The Security Research Division  
Network Associates, Inc.

This work funded under the  
Adaptive Cryptographically Synchronized Authentication (ACSA) project  
by the DARPA/ITO High Confidence Networks (HCN) Program  
Air Force Research Laboratory (AFRL) Contract No. F30602-98-C-0215

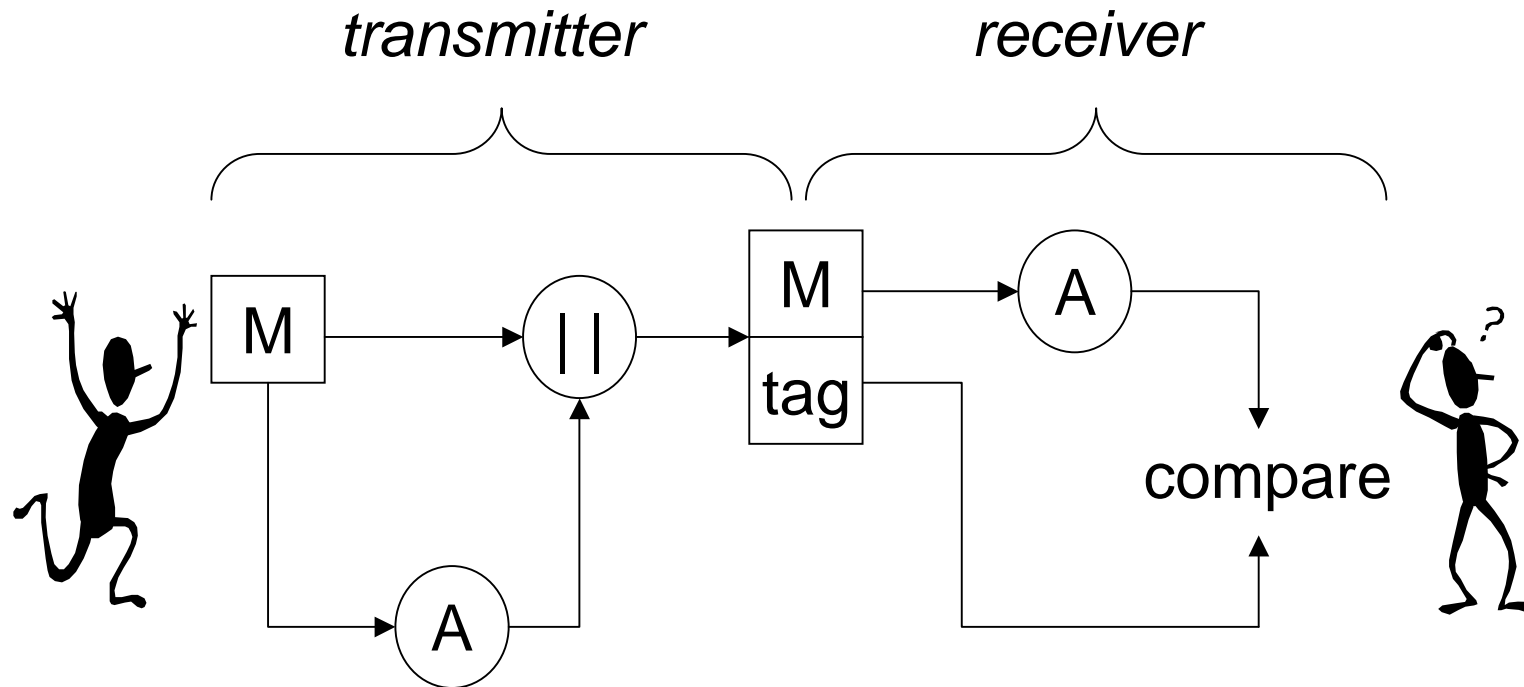
# Presentation Outline

- Background and Motivation
- Model
- Candidate Authentication Mechanisms
- Integration with IKE/IPsec
- Prototype
- Performance Measurements
- Strength/Performance Tradeoff
- Summary

# Terminology

- “Network authentication”
  - Connectionless integrity and data origin authentication for IP datagrams [IPsec definition by Kent, Atkinson]
- “Strength”
  - refers to cryptographic strength
  - generally expressed in terms of the probability of an adversary successfully forging a message
- “Performance”
  - refers to the amount of CPU processing required to generate or verify an authentication tag for a given message
  - directly related to the network throughput that can be supported when a link is processor-limited, not bandwidth-limited
  - expressed as the number of processor clock cycles per message byte processed by the authentication function

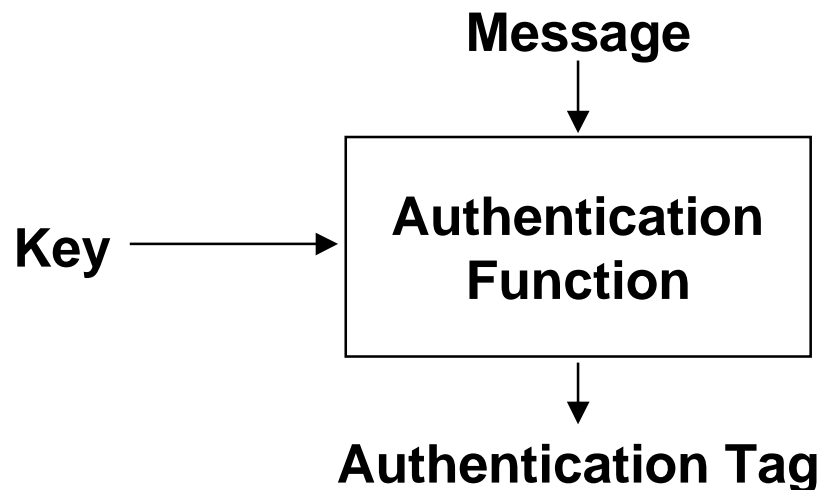
# How is network authentication provided?



- M = message
- A = authentication tag generation/verification function
- || = concatenation function
- tag = authentication tag

# What is a MAC?

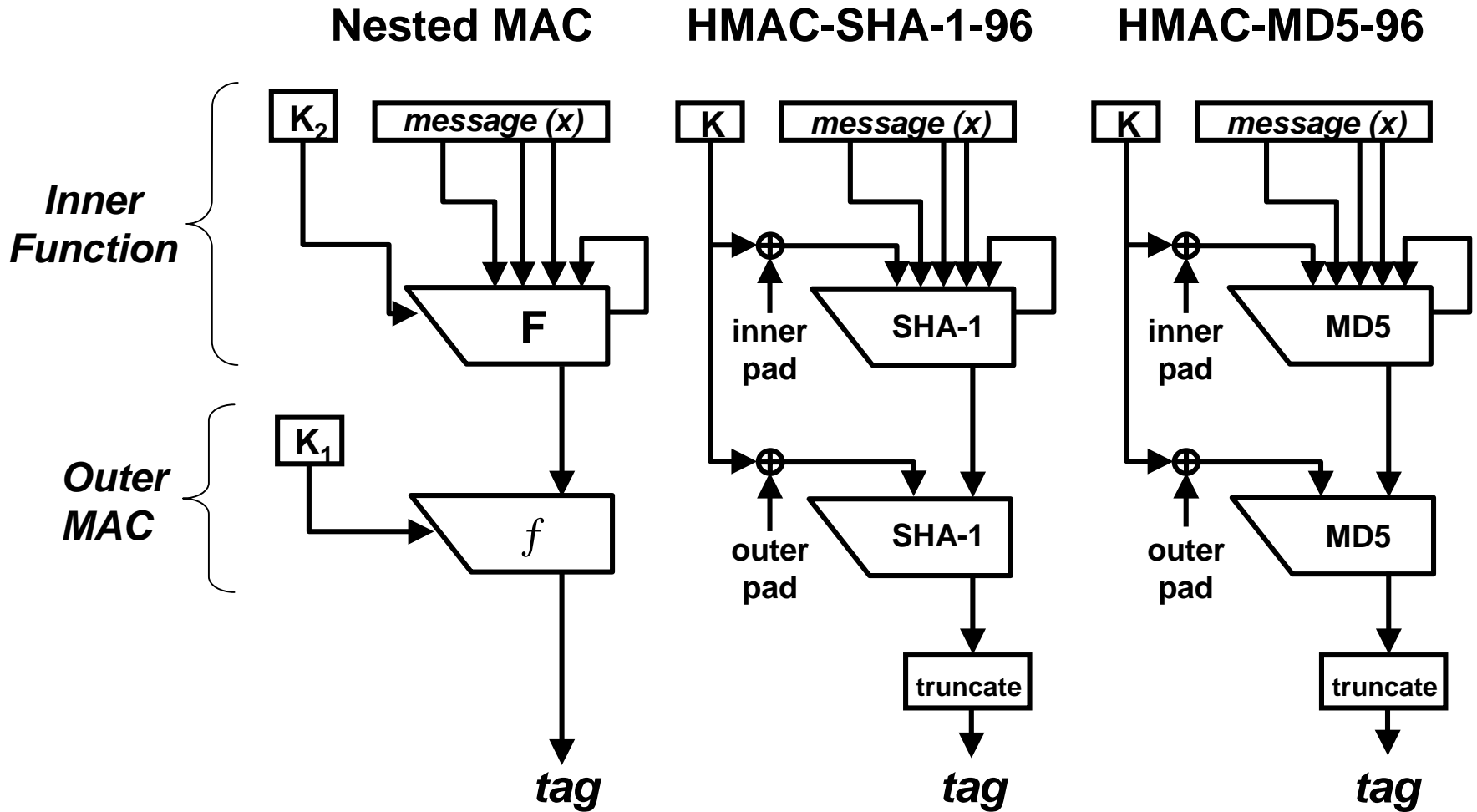
- Message Authentication Code (MAC)
  - can be used to provide network authentication
  - an authentication tag (also called a checksum) derived by applying an authentication scheme, together with a secret key, to a message
  - unlike digital signatures, MACs are computed and verified using a single shared secret key



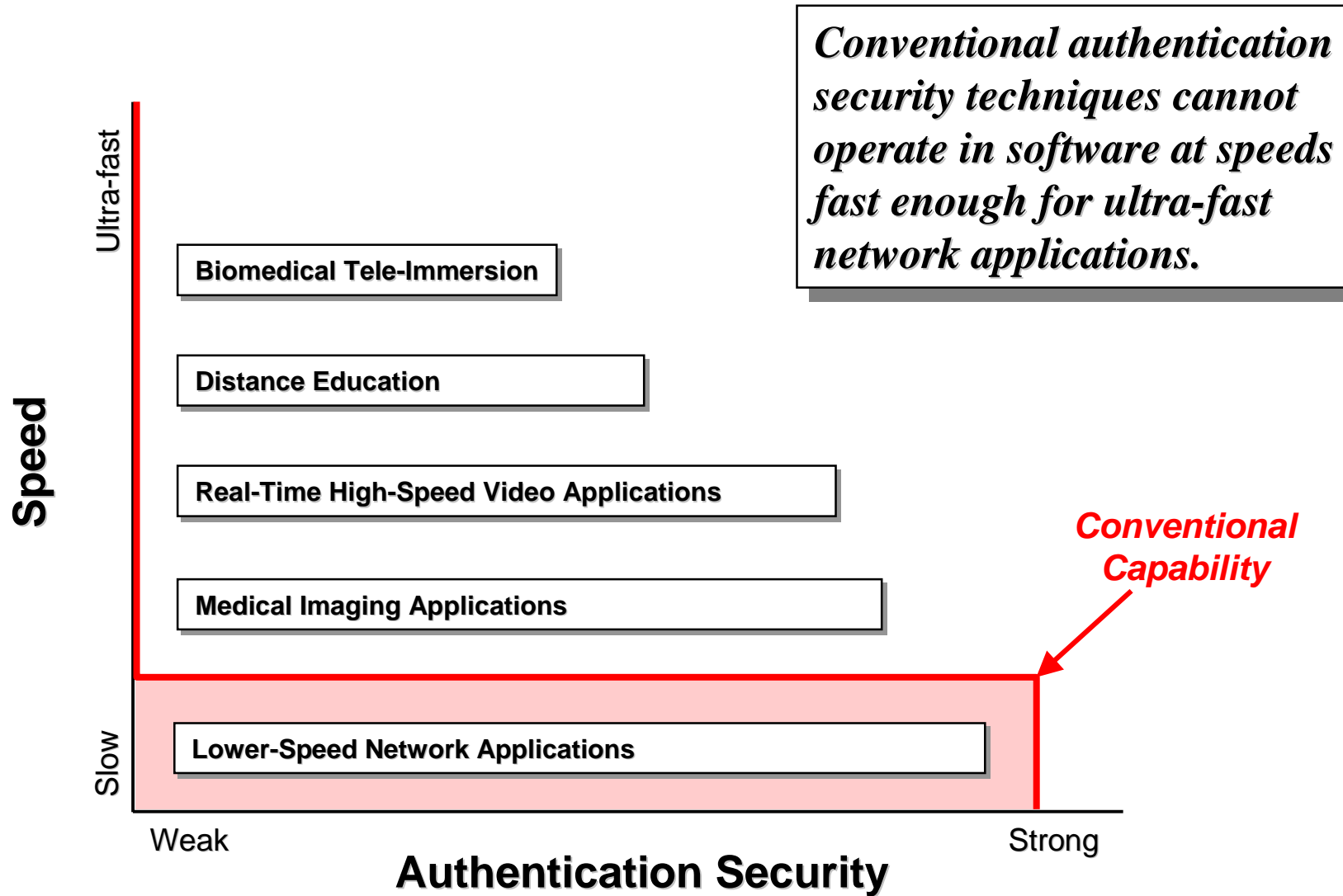
# Conventional MACs

- Block cipher-based MACs
  - DES-CBC MAC [FIPS Pub 113, ANSI X9.9]
  - MDC-2
- Stream cipher-based MACs
- Hash-based MACs
  - Keyed-MD5
  - HMACs [Bellare, Canetti, Krawczyk]
    - Hash-based Nested MACs
    - HMAC-MD5
    - HMAC-SHA-1

# Nested MACs and HMACs



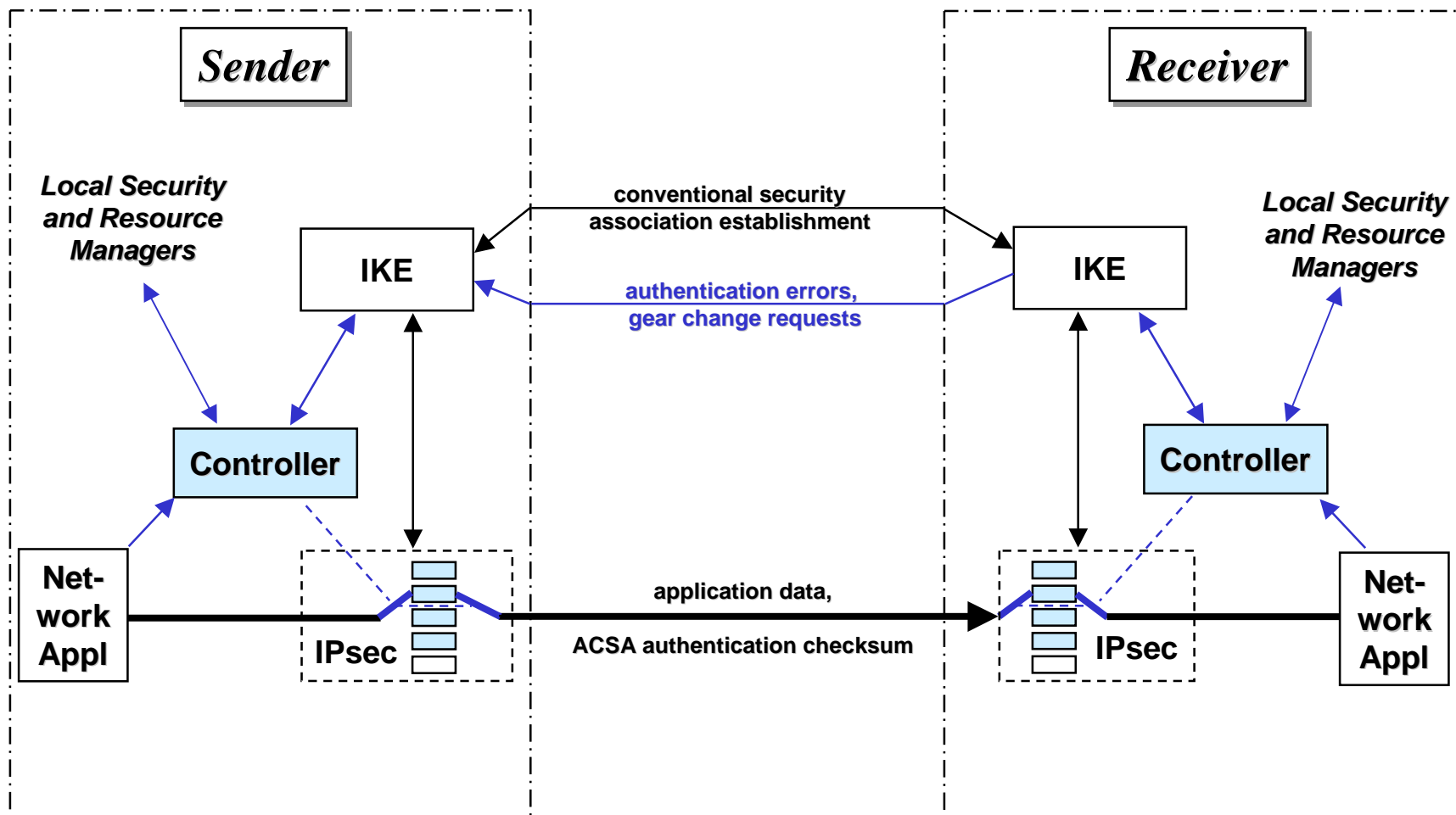
# The Need for Speed



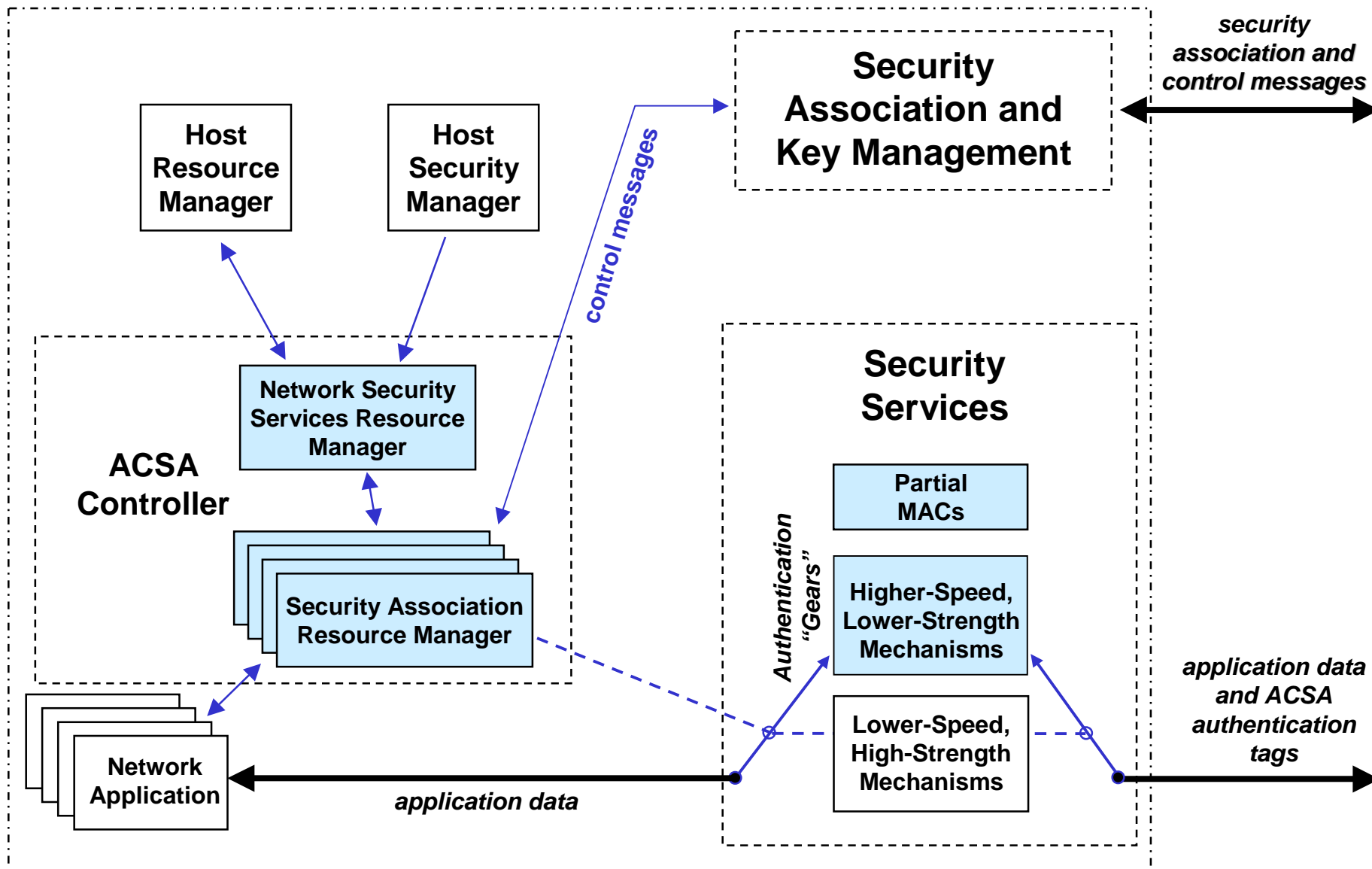
# Applicable Platform/Networks

- Conventional Platforms w/ High Speed Network Applications
  - Support for 100MBit, Gigabit Ethernet, or ATM networks on Pentium+ class machines without hardware crypto
- Platforms w/ Ultra-Fast Network Applications
  - Conventional cryptographic hardware is too slow for some ultra-fast networks
- Computationally-Limited Single Processor Devices
  - Personal Digital Assistants (PDAs)
  - Tokens
  - Mobile devices

# ACSA Model - Network View



# ACSA Model - Local View



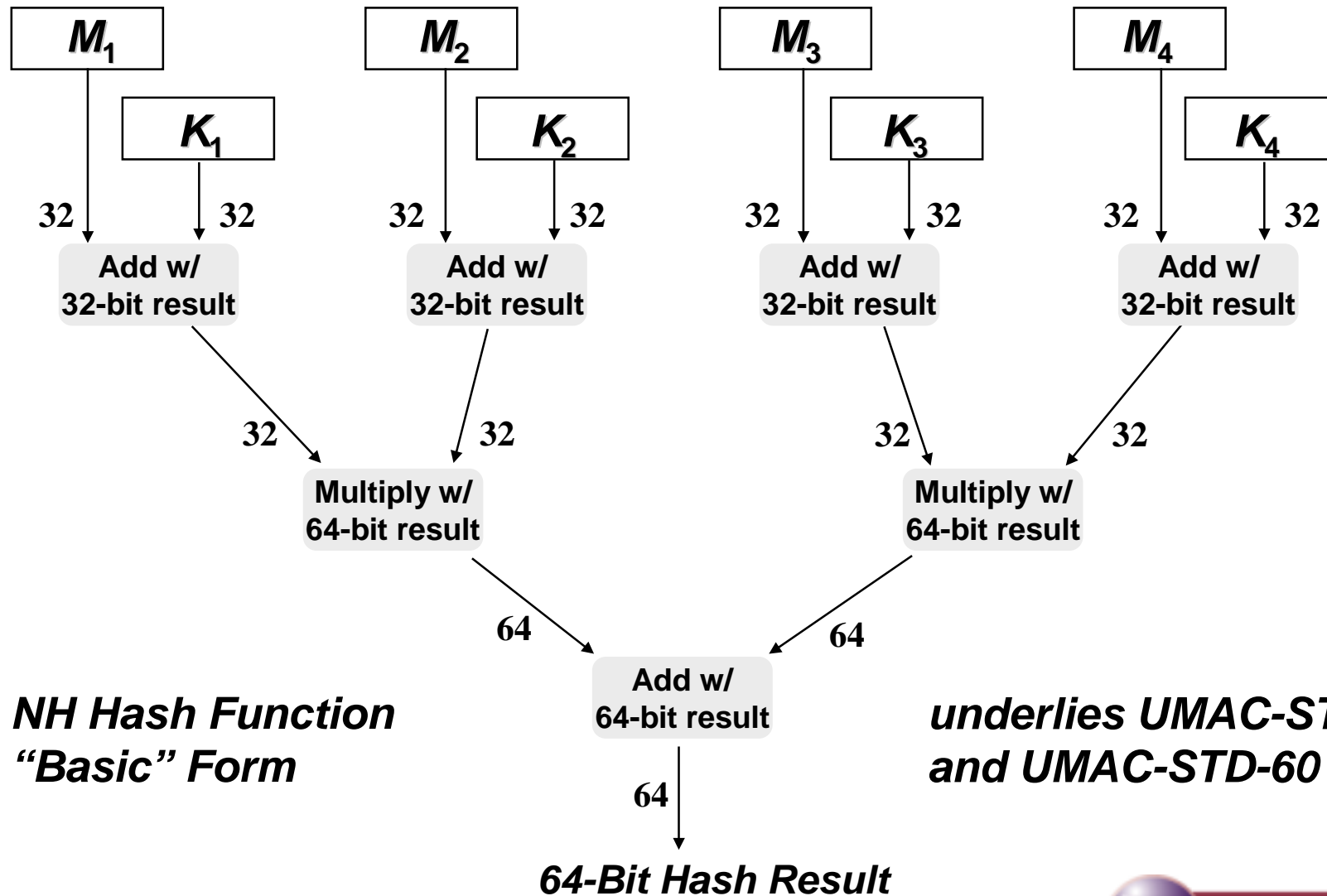
# Authentication Mechanism Classes

- Lower-speed, high-strength mechanisms
  - IPsec specified mechanisms
    - Mandatory: HMAC-SHA-1-96, HMAC-MD5-96 [Krawczyk]
    - Optional: HMAC-TIGER-96 [Anderson, Biham], HMAC-RIPEMD-160-96 [Keromytis, Provos]
- Higher-speed, lower-strength mechanisms
  - UMACs [Black, Halevi, Krawczyk, Krovetz, Rogaway]
    - First presented at Crypto '99
    - Will be presented at RSA Conf 2000 on Thursday at 8AM
    - Achieves high performance on SIMD architectures -- specifically MMX-enabled microprocessors
  - Inner Function Groups w/ bit scattering
- Partial MACs
  - Compute a MAC over only some of the message data

# UMAC-Based Authentication

- Subkey generation (performed once per SA)
  - Using a PRNG (e.g., RC4), map  $Key$  to  $K$  and  $A$ 
    - Generate  $K = 4096$  byte subkey for computing  $HM$
    - Generate  $A = 512$  bit subkey for computing  $Tag$
- Hashing the message  $Msg$  to  $HM = NHX_{Key}(Msg)$ 
  - $Len = |Msg| \bmod 4096$ , encoded as a 2-byte string
  - make  $|Msg|$  divisible by 64
  - $Msg = Msg_1 || Msg_2 || \dots || Msg_t$
  - $HM = NH_K(Msg_1) || NH_K(Msg_2) || \dots || NH_K(Msg_t) || Len$
- Computing the authentication tag:
  - $Tag = HMAC\text{-}SHA1_A(Nonce || HM)$

# UMAC - NH Function (Basic Form)



**NH Hash Function  
"Basic" Form**

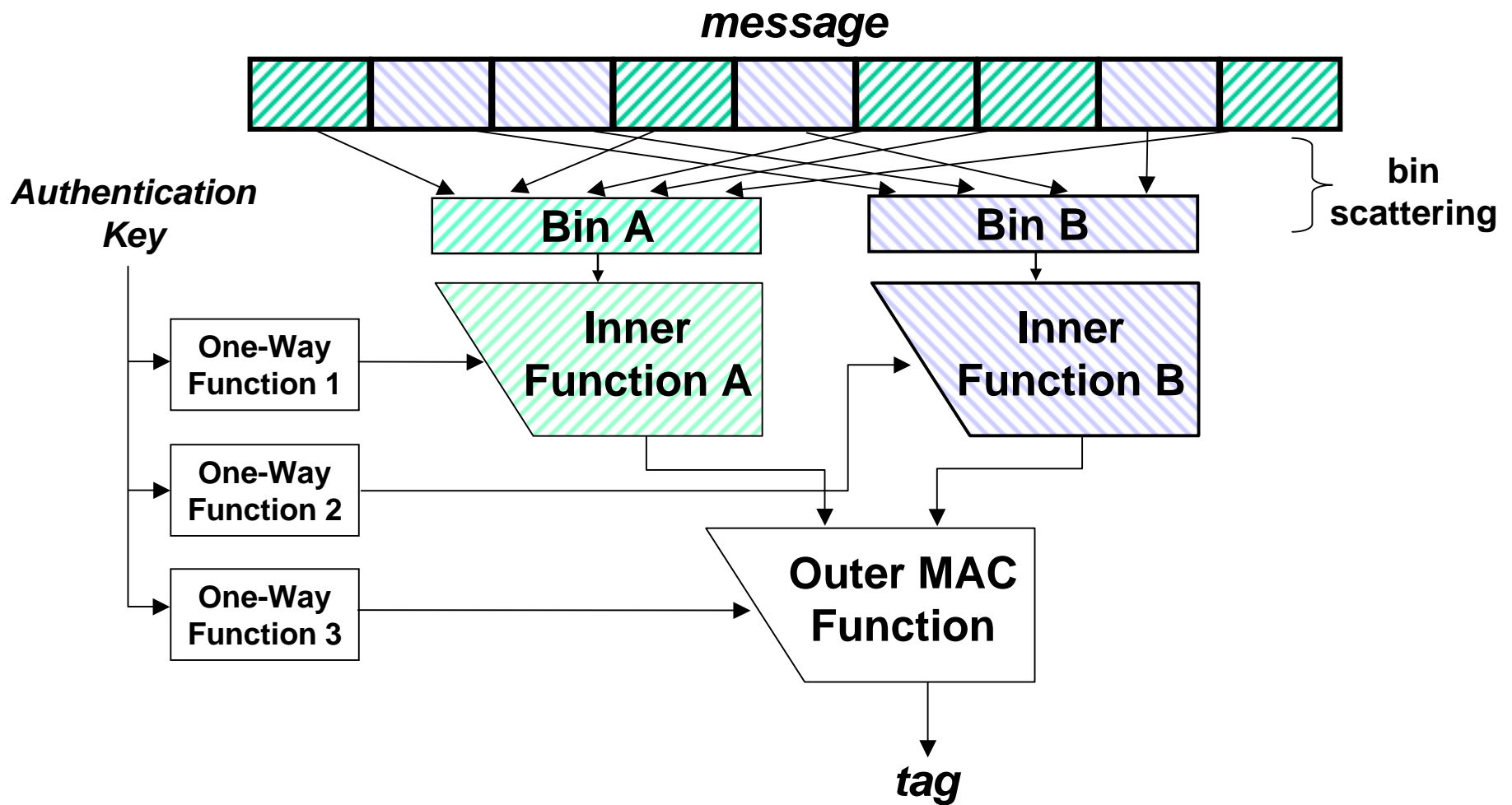
**underlies UMAC-STD-30  
and UMAC-STD-60**

# UMAC Parameter Sets

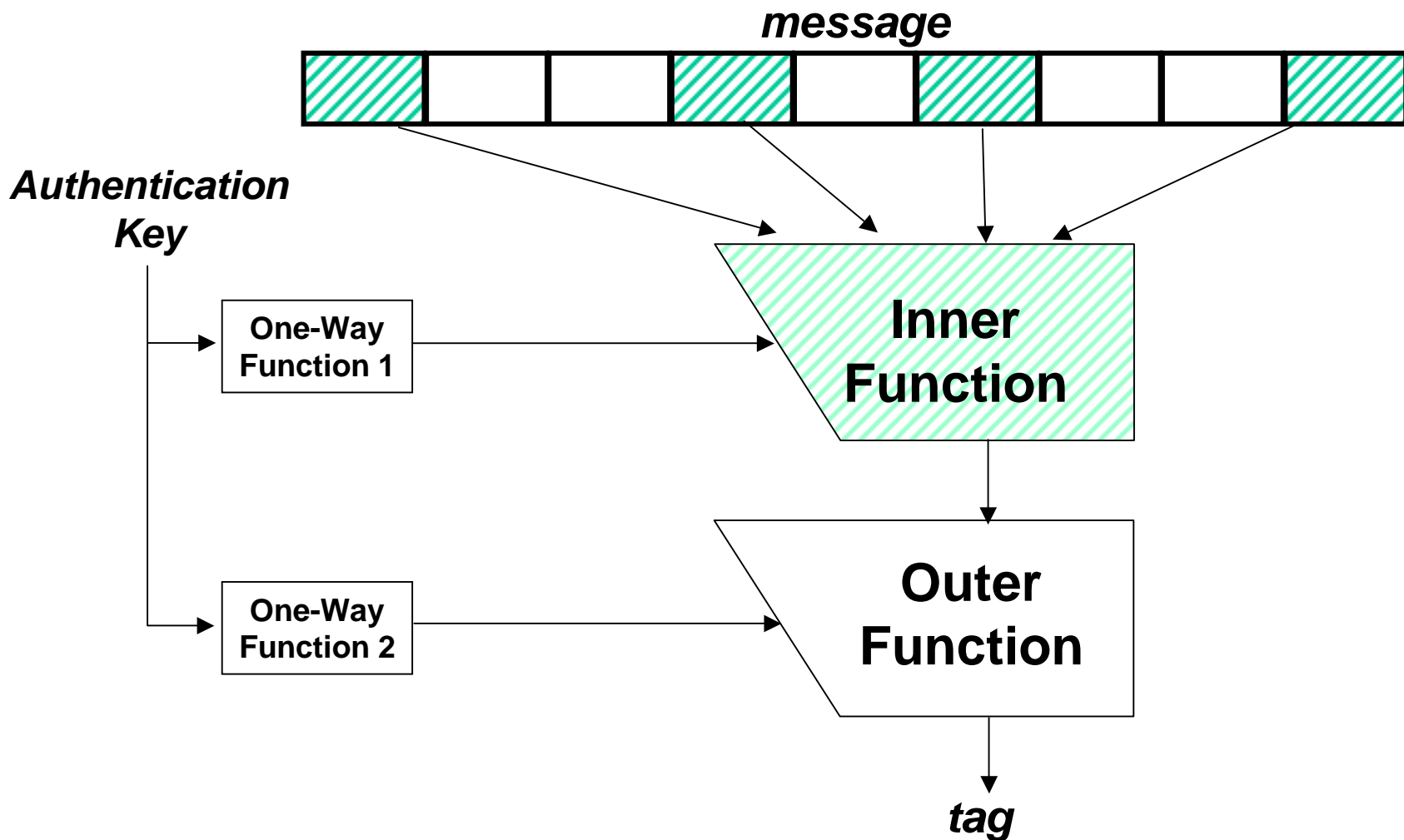
- Two forms: “basic” and “strided”
  - Basic: UMAC-STD-xx (can be computed on any processor)
  - Strided: UMAC-MMX-xx (designed for MMX processors)
- Inherently provides a strength/performance tradeoff among four algorithm parameter sets

UMAC Parameter Sets	Collision Probability	Peak Performance
UMAC-MMX-60	$2^{-60}$	0.98 cycles / byte
UMAC-MMX-45	$2^{-45}$	0.75 cycles / byte
UMAC-MMX-30	$2^{-30}$	0.51 cycles / byte
UMAC-MMX-15	$2^{-15}$	0.32 cycles / byte

# Inner Function Groups



# Partial MACs



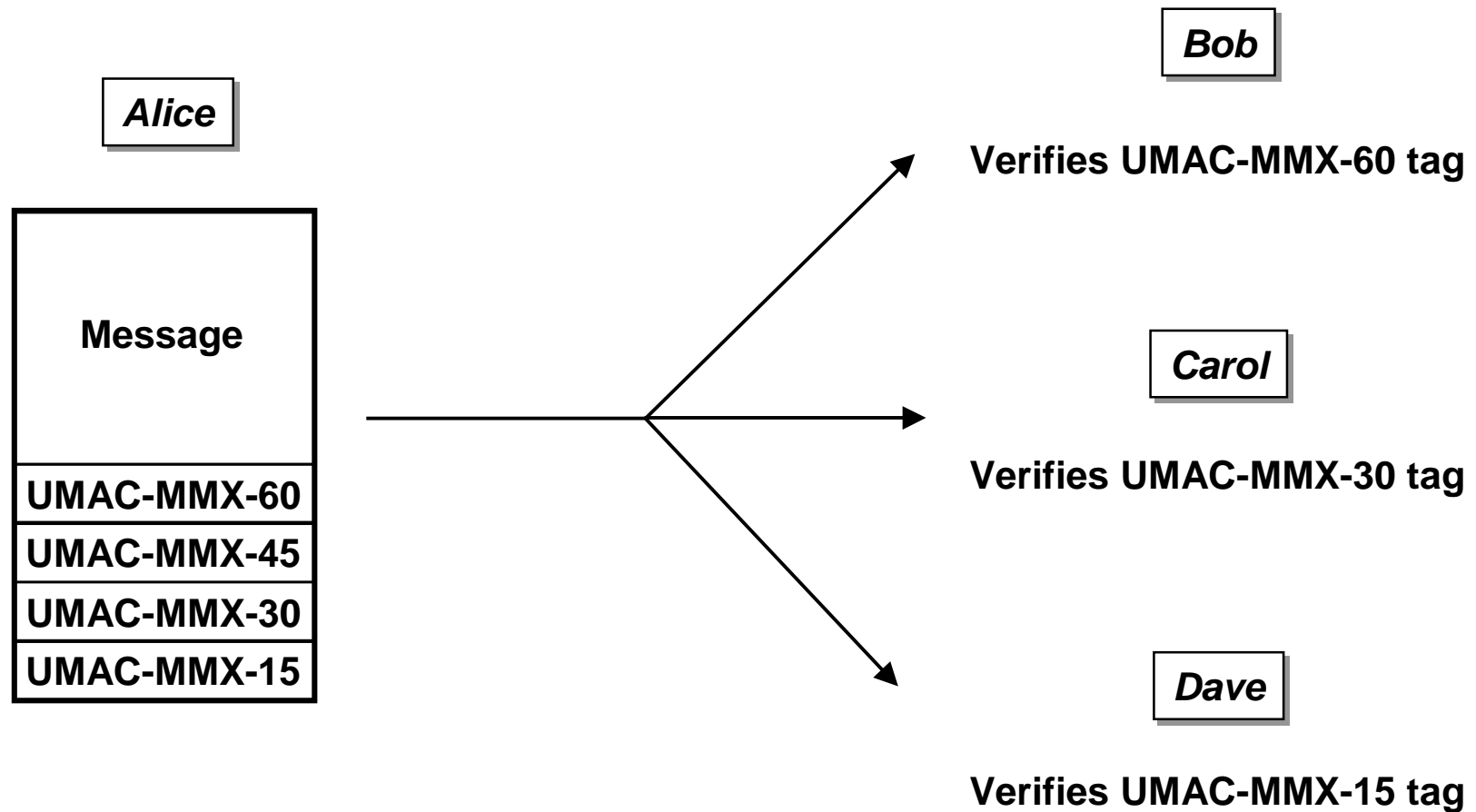
# Multiple Authentication Tags - Concept

- Sender generates multiple authentication tags for a single message packet
  - each authentication tag is produced with a different cryptographic strength-performance pair
  - the performance “cost” of providing multiple authentication tags is only slightly more than the cost of computing the “strongest” tag
    - Additionally generating UMAC-MMX-45, UMAC-MMX-30, and UMAC-MMX-15 tags when computing UMAC-MMX-60 costs less than 20%
- Receiver verifies only one authentication tag
  - receiver trades off authentication strength and performance cost to determine which authentication tag to verify
- Applicability
  - varying receiver processor load
  - multicast where receivers have varying capabilities, CPU loads

# Multiple Authentication Tags in Practice

- UMAC Multiple Authentication Tags
  - When generating UMAC-MMX-60, also generate UMAC-MMX-45, UMAC-MMX-30, and UMAC-MMX-15 tags
  - When generating UMAC-MMX-45, also generate UMAC-MMX-30 and UMAC-MMX-15 tags
  - When generating UMAC-MMX-30, also generate a UMAC-MMX-15 tag
- Inner Function Group Multiple Authentication Tags
  - For an inner function group comprised of two inner functions, generate three authentication tags
    - one for Inner Function A
    - one for Inner Function B
    - one for both Inner Functions A and B
  - Verifying only one inner function is roughly equivalent to the strength of a Partial MAC

# Multicast and Multiple Authentication Tags



# ACSA Prototype

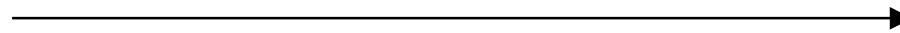
- Integration with IPsec/IKE
- Third-Party Network Security Implementations
- Prototype Toolkit
- Target Demonstration Platforms

# Integration with IKE/IPsec

- Use IKE to negotiate suite of security associations (SAs), one for each gear
- Synchronize sender and receiver by using the SPI in the ESP or AH header to indicate the gear
  - Each SPI corresponds to only one gear
- Issue: number of SAs generated per connection
  - increasing the number of SAs to be maintained could be a problem for memory-limited processors/platforms

# Security Association Negotiation Example

- |                  |               |
|------------------|---------------|
| 1. HMAC-SHA-1-96 | 7. PMAC-8     |
| 2. HMAC-MD5-96   | 8. PMAC-16    |
| 3. UMAC-MMX-60*  | 9. PMAC-32    |
| 4. UMAC-MMX-45*  | 10. PMAC-64   |
| 5. UMAC-MMX-30*  | :             |
| 6. UMAC-MMX-15   | 14. PMAC-1024 |



***IKE SA Proposals***

***Initiator***

***Responder***

1. HMAC-SHA-1-96
2. HMAC-MD5-96
3. UMAC-MMX-60\*
5. UMAC-MMX-30\*
6. UMAC-MMX-15
8. PMAC-16
9. PMAC-32

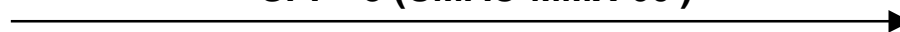


***IKE SA Proposal Responses***

***Sender***

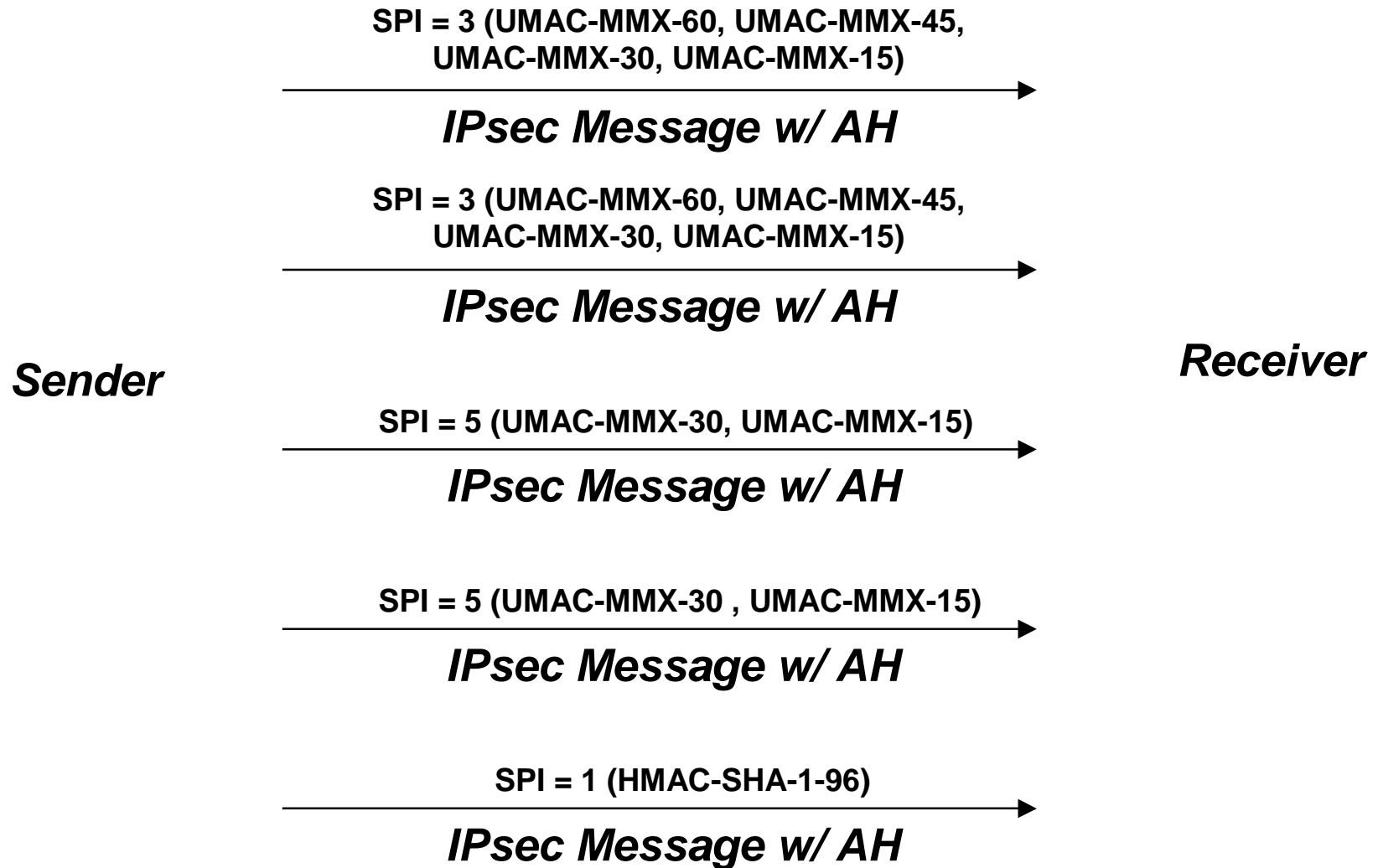
***Receiver***

SPI = 3 (UMAC-MMX-60\*)



***IPsec Message w/ AH***

# Sender Gear Switch Example



# Receiver Gear Switch Example

SPI = 3 (UMAC-MMX-60, UMAC-MMX-45,  
UMAC-MMX-30, UMAC-MMX-15)

→  
***IPsec Message w/ AH***

Receiver Computes:

UMAC-MMX-60

SPI = 3 (UMAC-MMX-60, UMAC-MMX-45,  
UMAC-MMX-30, UMAC-MMX-15)

→  
***IPsec Message w/ AH***

UMAC-MMX-30

SPI = 5 (UMAC-MMX-30, UMAC-MMX-15)

→  
***IPsec Message w/ AH***

UMAC-MMX-30

SPI = 5 (UMAC-MMX-30 , UMAC-MMX-15)

→  
***IPsec Message w/ AH***

UMAC-MMX-15

SPI = 1 (HMAC-SHA-1-96)

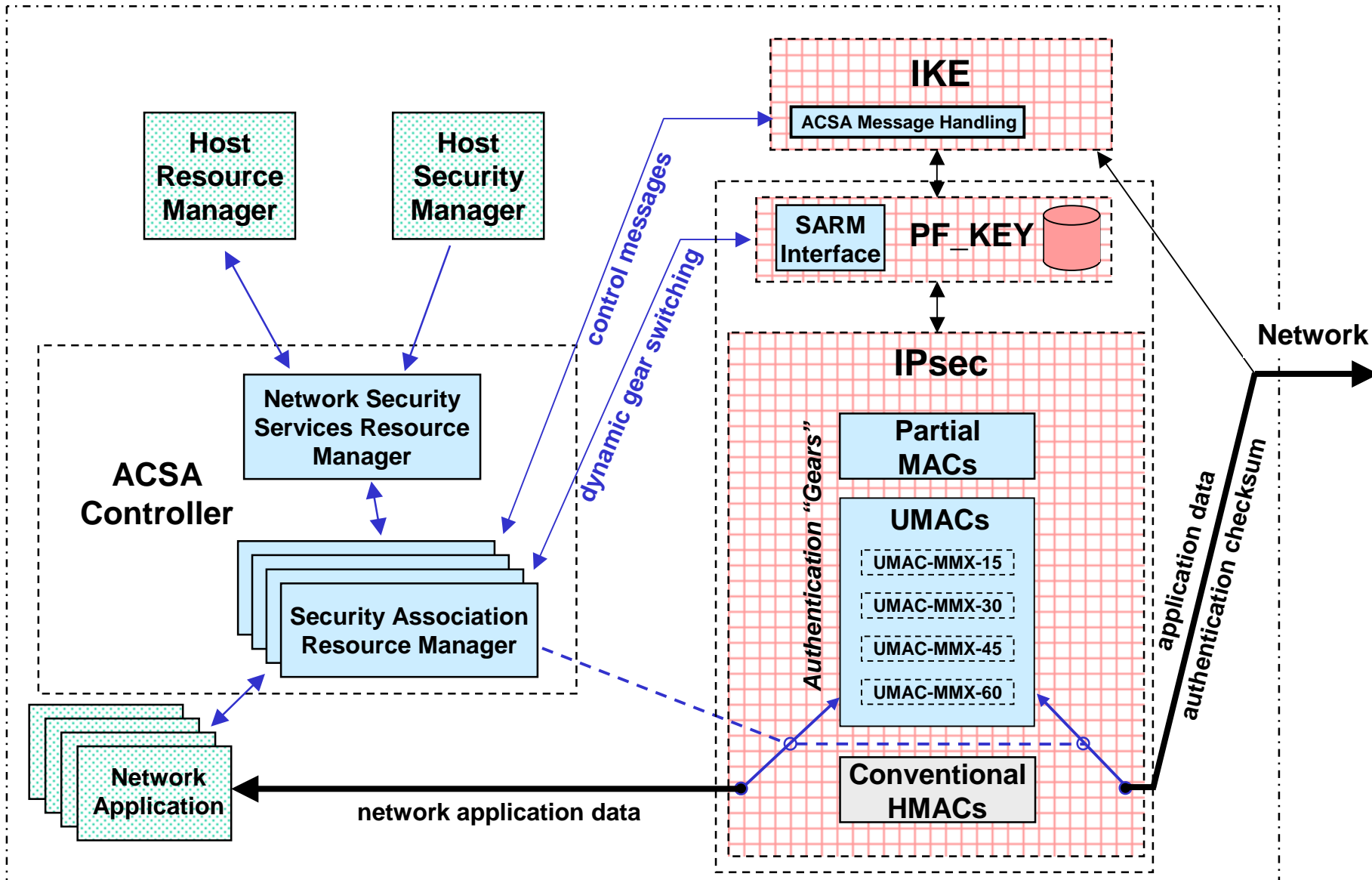
→  
***IPsec Message w/ AH***

HMAC-SHA-1-96

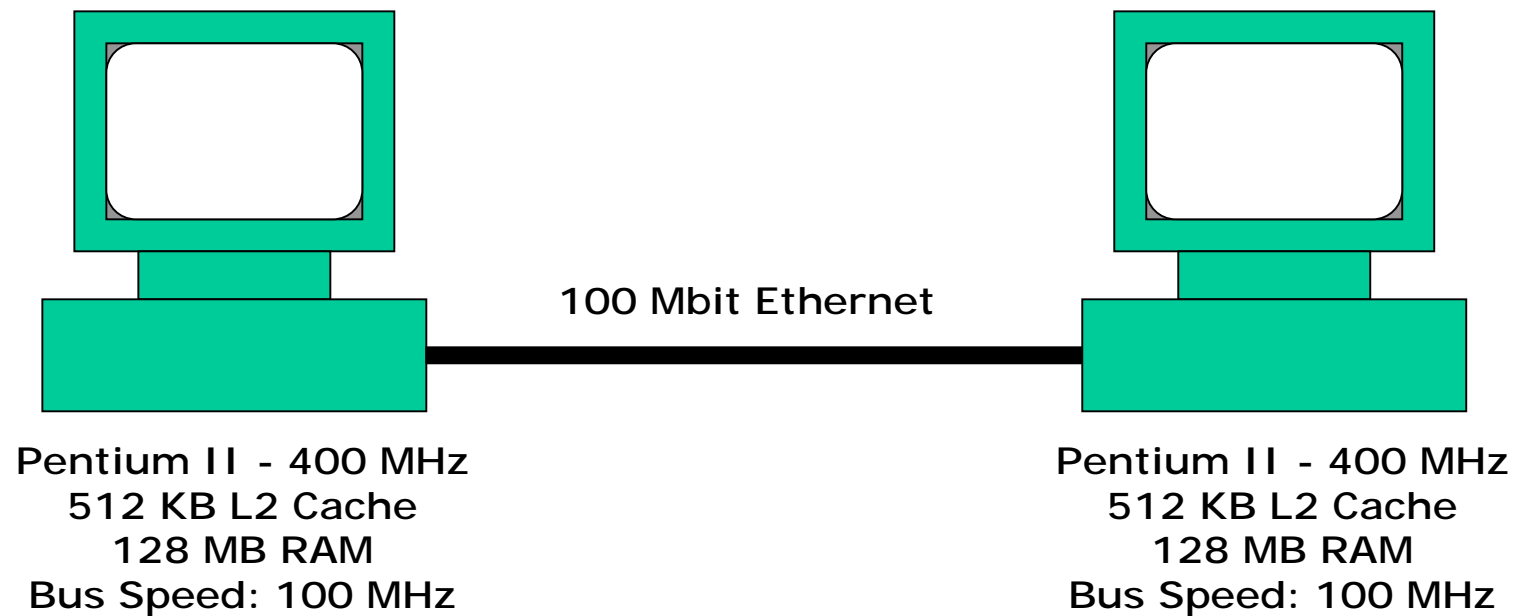
# Third-Party Network Security Implementations

- Criteria
  - IKE and IPsec compliant
  - Currency - Active development
  - Redistributable source code
  - Quality and Reputation
  - Future Integration
  - Performance
- Candidates
  - **NRL**
  - **OpenBSD**
  - **FreeSWAN** - most active open development, performance
  - **NIST Cerberus/PlutoPlus**

# ACSA Prototype Toolkit



# Target Demonstration Platforms



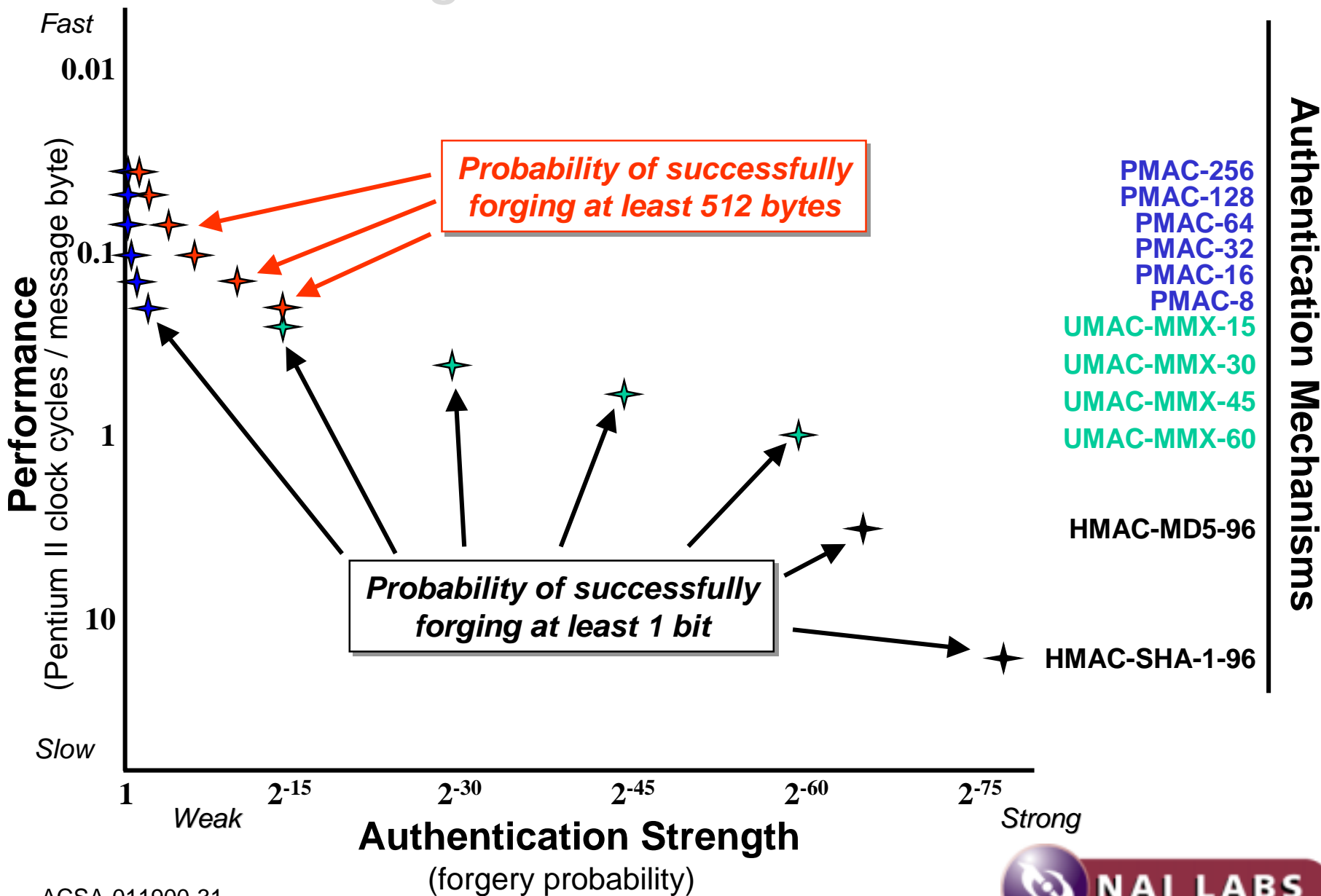
# Performance Measurements

Algorithm	Clock cycles per message byte (4096 byte packets)	Clock cycles per message byte (65536 byte packets)
HMAC-SHA-1-96	13.0	13.0
HMAC-MD5-96	6.4	6.3
UMAC-MMX-60	1.42	1.35
UMAC-MMX-45	1.23	1.13
UMAC-MMX-30	1.01	0.76
UMAC-MMX-15	0.80	0.53
PMAC-8	0.77	0.44
PMAC-16	0.68	0.20
PMAC-32	0.62	0.10
PMAC-64	0.60	0.076
PMAC-128	0.58	0.055
PMAC-256	0.57	0.047
PMAC-512	0.57	0.044

# Performance - Real World Issues

- At very high speeds, other characteristics affect performance
  - packet size
    - smaller packet sizes incur a large overhead, benefits of fast inner functions largely negated
  - processor architecture
    - caching
      - cache sizes, multi-level caches
    - instruction execution
      - memory structure
      - dynamic execution

# Strength vs. Performance



# Acknowledgements

- NAI Labs Team - Jamison Adcock, David Balenson, David Carman, Michael Heyman, Alan Sherman
- Dennis Branstad for proposing the initial concepts of trading off authentication strength and performance
- Phillip Rogaway contributed very helpful detailed comments on our preliminary work
- Hugo Krawczyk for constructive feedback
- Ted Krovetz provided us with assistance in implementing UMAC
- DARPA Program Managers Hilarie Orman and Doug Maughan for supporting this research

# Summary

- Framework for taking advantage of strength-performance tradeoffs in network authentication
- New authentication mechanism approaches
  - Inner Function Groups with bit scattering
  - Partial MACs
  - Multiple message authentication tags
- Prototype toolkit to be made available
- Applicability
  - High-speed network applications on processor-limited platforms
  - Not appropriate for applications that cannot tolerate single bit forgeries (e.g. financial applications, control information)