Mini-MAC: Raising the Bar for Vehicular Security with a Lightweght Message Authentication Protocol

Jackson Schmandt[‡], Alan T. Sherman[*], Nilanjan Banerjee[†]
CSEE Department, University of Maryland, Baltimore County (UMBC)
{schmandt, sherman, nilanb}@umbc.edu

January 26, 2016

Note: Footnotes not fixed yet. Cryptologia style not done yet.

Abstract

We propose Mini-MAC, a new message authentication protocol that works in existing automotive computer networks without adding any bus overhead.

Deployed in many vehicles, the CAN bus is a low-speed network connecting electronic control units (ECUs), including those that control critical functionality such as braking and acceleration. The CAN bus is extremely vulnerable to malicious actors with bus access. Traditionally, Message Authentication Codes (MACs) help authenticate the sender of a message, and variants prevent message replay attacks; however, standard MACs are unsuitable for use on the CAN bus because of small payload sizes. Restrictions of the CAN bus, including the need not to delay messages, severely limit how well this network can be protected.

Mini-MAC is based on a counter-seeded HMAC, augmented with message history and truncated to fit available message space. It causes no increase in bus traffic and incurs a very small performance penalty relative to the provably secure HMAC. It is the first proposal to combine these two tenets for vehicle networks. Even though the CAN bus cannot be properly secured against a dedicated attacker, Mini-MAC meaningfully raises the bar of vehicular security, enhancing the safety of drivers and others.

Index terms— CAN bus, automotive security, message authentication code, Mini-MAC, vehicle security, applied cryptography.

1 Introduction

At the 2015 Black Hat conference, Miller and Valasek [REF] gained full control of a new Jeep, including its engine, brakes, steering, and entertainment system, by exploiting vulnerabilities in its computer network and WiFi implementation and by rewriting firmware on a controller connected to the car's entertainment system. This demonstration, and other similar projects [add refs on car insecurity], highlight the egregious state of vehicular security, including the lack of authentication of messages sent on the Controller Area Network (CAN).

To strengthen vehicular security in a simple and practical yet meaningful way—without replacing the CAN bus—we propose Mini-MAC, a new variable-length Message Authentication Code (MAC) for the CAN bus that works with small message sizes without delaying messages. Based on the provably-secure HMAC, Mini-MAC protects against masquerade attacks. Mini-MAC also incorporates a counter and message history to protect against replay attacks. To avoid sending separate messages to different recipients, Mini-MAC applies authentication keys shared among groups of communicating Electronic Control

[1]Cyber Defense Lab
[2]Mobile Pervasive Sensor System Lab

Units (ECUs). It is the first proposal to authenticate messages on the CAN bus without increasing bus traffic or delaying messages.

Traditional authentication protocols (including digitial signatures or full-length MACs) are unsuited for the CAN bus due to small packet size, limited computational power of the ECUs, and the need not to delay messages (e.g., by time-consuming computations or by increasing bus traffic).

Mini-MAC improves on previous proposals, including Lin-MAC [LSV12], by not increasing bus traffic. Furthermore, Mini-MAC is easy to implement, requires no fundamental change to the underlying functionality of the ECUs, and requires no special hardware.

Our work includes a protoype implementation of Mini-MAC and preliminary timing studies of Mini-MAC for three component hash functions (MD5, SHA-1, SHA-2). For fastest speeds, we recommend using MD5.

Our contributions include:

- Mini-MAC, an authentication protocol suitable for vehicular systems, including the CAN bus, that require short message sizes and no message delays.
- Mini-MAC meaningfully raises the bar on authentication strength for the CAN bus, protecting against masquerade and replay attacks.
- Experimental demonstration of Mini-Mac, including execution times for an HMAC construction using the MD5, SHA-1, and SHA-2 hash functions.

2 Background

This section briefly reviews essential background on vehicular security and message authentication codes. The experienced reader may wish to skip to Section 3. We assume the reader is familiar with cryptographic hash functions as explained, for example, by Stinson [**?**] and NIST [Nat15]. Note: You mentioned Stinson on the last draft notes – this is the book Cryptography - Theory and Practice?—YES.

2.1 ECUs

The Electronic Control Units (ECUs) found in an automotive computer network are low-power, single-purpose devices. ECUs on the CAN bus control many components in a modern automobile, from headlights and window controls, to brakes and engine. They are not typically designed with security in mind and frequently comprise a basic CAN bus transceiver, basic message processor, and an actuator. The message processor identifies whether or not a message being broadcast is interesting to the ECU and arbitrates bus rights with the other ECUs.

2.2 The CAN Bus

The CAN bus is a simple, low-speed bus designed to network simple nodes. In an automotive environment, it typically runs at 500 kbps.[1] As shown in Figure 1, a message contains an 11-bit identifier field and a data payload, as well as some control bits. Figure 1 shows the data payload as 8 bits, but it is typically 8 to 64 bits [ref?]. The payload of up to 8 bytes is the most important element, as any MAC must fit into this frame or use a more complex multi-frame data transmission protocol that may or may not be supported on all ECUs.

Ideally, and in the case of Mini-MAC, the MAC can fit into the payload with the data, thus not increasing bus utilization. Data captured by the authors from a 2010 Toyota Prius show that a large percent (61%) of messages use no more than four bytes of data in the payload. We observed approximately 25 messages sent per second on average (40 maximum per second).

2.3 Bus Access

To spoof or replay messages on the CAN bus, the attacker must have access to it. There are several ways to access the CAN bus: (1) There is a physical connection through the On-Board Diagnostic (OBD-II) port, typically located underneath the steering wheel. An attacker might hide access to this port by splicing into unexposed wires. (2) An attacker might corrupt an ECU by rewriting its firmware. An attacker might

[1] kilobits per second (kbs).

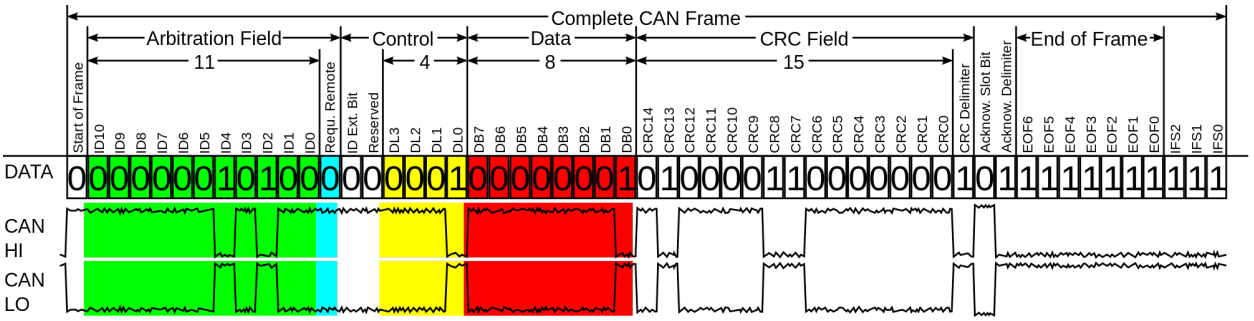

Figure 1: The CAN frame [] [need to cite fig]. Each message on the CAN bus is a structured sequence of 55–111 bits including 8–64 data bits.

do so while the car is being serviced or by entering the car while it is parked. (3) An attacker might gain access to the CAN bus by exploiting or corrupting a peripheral device connected to it, such as a cellular phone, audio system, or Bluetooth radio. For example, Checkowa [CMK+11] gained bus access by packing malware into a WMA audio file played on the car stereo. [give another example of wireless access from drive-by attacker]

Despite many demonstrated security flaws, automotive manufacturers have been unreasomnably hesitant to acknowledge the vulnerabilities inherent to the CAN bus, sometimes wishfully claiming that undetected bus access is difficult. [do you have a cite? for example, a quote from a car manufacturer saying something ridiculous?]

2.4 Message Authentication Codes

Given a message and optionally a key, a Message Authentication Code (MAC) computes a short string (called a tag) that a recipient can use, together with the message, to verify the authenticity of the message. The recipient verifies the tag by recomputing it.

The Keyed-Hash Message Authentication Code (HMAC) [BCK96, Nat08] is a well-known MAC construction that keys an underlying component hash function. Breaking it is as hard as breaking the component hash function.

HMAC is computed as

$$H((k \oplus \text{opad}) \parallel H((k \oplus \text{ipad}) \parallel \text{message}), \quad (1)$$

where k is the key, and opad and ipad are the outer and inner hash padding strings, respectively. These strings are constant strings defined by 0x5C and 0x36, repeated until the hash input is of the appropriate length. H is the component hash function used. The symbols $\oplus$ and $\parallel$ represent XOR and concatenation, respectively.[2]

3 Problem Statement

Our task is to create an authentication mechanism suitable for use on the CAN bus in vehicular environments. The mechanism must improve security over the standard non-secured bus, while not increasing bus traffic nor delaying messages. The mechanism must protect against replay and masquerade attacks as defined in Section 5. It must function in the highly constrained vehicular environment, which includes slow, low-power ECUs incapable of complex cryptographic functions.

In test data captured by the authors from a 2010 Toyota Prius, approximately 61% of messages contain at most four data bytes (20% contain 4 bytes; 16% contain 3 bytes; 17Approximately 35% of messages contain a full 8 data bytes, and 4% contain 7 bytes.

[2] Exclusive-or (XOR).

Thus, for most messages, there are at least four bytes of space available for a MAC tag.

4 Previous Work

Previous proposals to add authentication to the CAN bus violate the engineering constraints described in Section 3, increasing bus utilization and delaying messages. For example, pairwise key distribution among the ECUs and data that overflow CAN frame boundaries cause additional messages to be sent, delaying messages.

For example, Lin and Sangiovanni [LSV12] propose Lin-MAC, a keyed MAC with counter based on pairwise key distribution. Encrypting the same message to n different ECUs requires n messages to be sent. Using the full HMAC-MD5 requires 128 bits to be sent, requring two CAN frames per message.

Other recent CAN security projects suffer from similar limitations. Woo et al. [WJL15] propose a keyed MAC based on pairwise key distribution, packing the tag into the extended ID field (not used by all ECUs) and the CRC field in the CAN trailer (for a related proposal of ours, see Section 9.2). These bits fit only if the ECUs use the extended ID field. Their proposal requires a hardware redesign of CAN transceivers or rewriting a layer of message transmission firmware. Care should be taken when comparing their computation times because they assume a much more powerful message processor than we do.

Zalman et al. [ZM14] propose a fixed-size, time-stamped MAC based on pairwise key distribution. Their tag overflows the CAN frame, increasing bus utilization, and as the authors acknowledge, delaying messages.

Xie et al. [XLL+15] propose packing multiple messages into one CAN frame using a keyed MAC with based on pairwise key distribution. They unrealistically assume that the messages and MAC tag are short enough to fit into one frame. By queing messages into batches, their system delays messages.

[do we want to cite any other works on improving car security? If so, do it here.]

Many automakers and parts manufacturers are now members of the Open Alliance [need ref or URL], a non-profit group researching and encouraging the use of an Ethernet-based high-speed physical layer for use in vehicles. This approach would enable the use of established network security mechanisms in vehicle networks.

5 Adversarial Model

We consider three classes of adversaries:

Type 1 (*Strongest adversary*): A permanent entity on the CAN bus with a valid key for the MAC it wishes to generate.

Type 2 (*Strong adversary*): A permanent entity on the CAN bus without a valid key for the MAC it wishes to generate.

Type 3 (*Weak adversary*): A transient entity on the CAN bus without a valid key for the MAC it wishes to generate.

For example, a Type 1 adversary might be a compromised ECU on the CAN bus. A Type 2 adversary might be a malicious piece of hardware attached to the CAN bus. A Type 3 adversary might be a criminal who has gained temporary access to the CAN bus, perhaps via a wireless channel from another nearby car. For a Type 3 attacker, we assume the adversary's access to the CAN bus is limited to minutes (not hours). The differences among these attackers are what keys they know and for how long they have access to the CAN bus. This project aims to defend against Type 2 and Type 3 adversaries. Our techniques do not protect against a Type 1 adversary, who can spoof any message.

Motivation of the attacker includes criminal mischief (e.g., crashing car, destroying property) and theft. For example, spoofing messages on the CAN bus can unlock doors, disable brakes, and accerate the car. Goals of the attacker include spoofing or replaying messages on the CAN bus that will be accepted by an ECU as valid.

We assume the attacker posses complete knowledge of the CAN bus and ECUs, including all protocols, message IDs, and formats. We assume the attacker

has substantial computing power, reliable access to the CAN bus, and is able to monitor and inject messages on the bus. We assume, however, that the adversary cannot inject more than 40 messages per minute without detection.

We assume the ECUs are trustworthy and that the adversary cannot break standard cryptographic functions including encryption and hash functions.

This work does not address Denial-of-Service (DOS) attacks aimed at preventing a driver from using her vehicle. For example, our techniques do not prevent an attacker from flooding the CAN bus with messages. There are many simple physical DOS attacks, such as slashing a tire, cutting wiring, or draining fuel.

Although we assume the adversary has complete knowledge of the target technology, in practice the adversary must deal with the straightforward yet cumbersome task of learning this technology, which may include new ECUs and message formats.

6 Mini-MAC

Mini-MAC is a group-keyed lightweight variable-length truncated HMAC that depends on a counter and on recent message history. It does not increase bus traffic or delay messages. We explain Mini-MAC in three layers of abstraction: architecture, design, and implementation.

6.1 Architecture

Four core architectural elements characterize Mini-MAC: (1) Variable-size output to fit available space in the CAN packet. (2) Shared keys among groups of ECUs to avoid increasing bus traffic. (3) A counter to mitigate replay attacks. (4) Message history to defeat transient attackers, to mitigate replay attacks after counter resynchronization, and to serve as "salt" against possible unknown precomputation attacks. Elements (1) and (2) trade authentication strength for performance; see Section 8.2 for a discussion of this tradeoff.

To avoid sending long tags using additional messages, Mini-MAC truncates the HMAC to fit available space in the CAN frame (typically the resulting tag is approximately four bytes).

To avoid increasing bus traffic by separately authenticating the same message to different recipients, Mini-MAC uses long-term shared group keys instead of pairwise keys. For example, in Figure 2, ECUs 2, 3, and 4 share the group key k_3. Group key distribution is possible because, at system design, the designer knows which ECUs communicate with which others. There is no need for a dynamic group key update capability because group membership will never change. Details about how these keys are initially set, and possibly later updated, are beyond the scope of this paper.

Using a counter is a simple, standard, and effective way to mitigate replay attacks.

Each ECU saves recent messages sent on the bus and XORs them into the HMAC input. More specifically, each ECU saves a separate message history for the successfully authenticated messages received for each group key. Thus, the adversary cannot insert messages into any message history without successfully forging messages.

There are three reasons for the message history architectual element: First, it adds protection against any transient attacker (even one who knows the group keys) who cannot deduce enough of the message history. Second, it adds an additional layer of protection against replay attacks. This layer is useful if the counters need to be resychronized (see Section 9.1). Consider what happens if a group of ECUs reset their counters and message history to a previously stored synchronized state. Without message history, the network is potentially vulnerable to a replay attack of messages sent from a time the reset counter state was previously used. With message history, it is extremely likely that the unfolding message history will rapidly differ. Third, the history adds an unpredictable "salt" that might mitigate some possible precomputation attacks. Whereas counter values are predicable, message history is not.

6.2 Design

Figure 3 shows how to compute $Mini\text{-}MAC(k, M_n, s, C, \text{History})$, where k is the

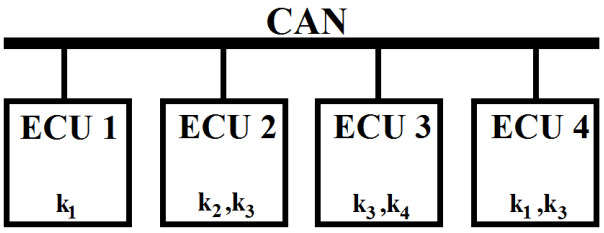

Figure 2: Mini-MAC key distribution. Each group of ECUs shares an authentication key.

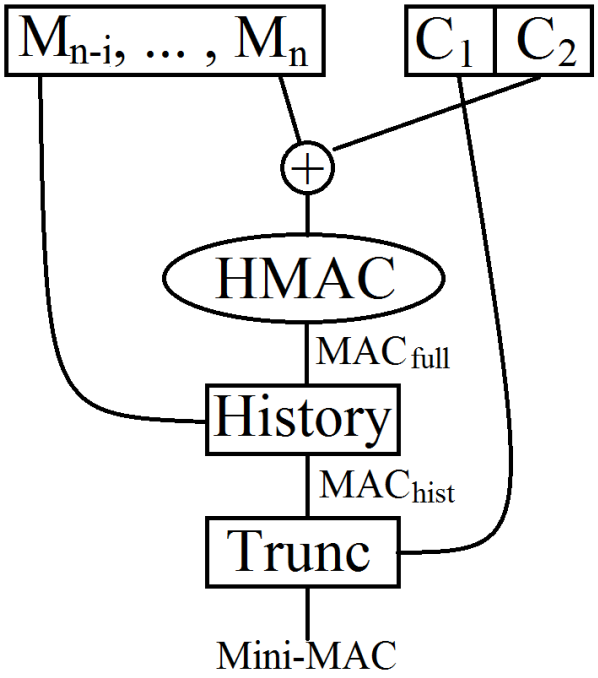

Figure 3: Mini-MAC. The Mini-MAC of a message M_n is a truncated HMAC of the XOR of M_n, a counter, and the most recent i messages.

group key, M_n is the current message, s is the number of available bits in the current CAN frame, C is the message counter, and History $= (M_{n-\lambda}, \ldots, M_{n-1})$ is the sequence of the most recent λ messages.

The Mini-MAC tag is computed as

$$\text{MAC}_{\text{mini}} = \text{trunc}(s, \text{HMAC}(k, \text{Input})), \qquad (2)$$

where HMAC is the underlying HMAC and $\text{trunc}(s, \cdot)$ extracts s bits from its input. The input to HMAC is computed as

$$\text{Input} = M_n \oplus C \oplus (M_{n-\lambda} \oplus \cdots \oplus M_{n-1}). \qquad (3)$$

6.3 Implementation

We implemented Mini-MAC using three different component hash functions (MD5, SHA-1, and SHA-2) to compare the resulting running times. Each group key is 128 bits. The $\text{trunc}(s, \cdot)$ function extracts the first s bits of its input. We used an [X]-bit counter, which (assuming at most 40 message per second) ensures no repeated counter state for 20 years.

For **HMAC-MD5**, we adapted Peslyak's [Pes09] implementation of MD5 [Riv92] for the MSP430 platform, producing a 128-bit output. Despite known collision attacks on MD5 [WY05], we consider MD5 for Mini-MAC for its very fast speed. [the relevant attack here is known pre-image, not collision finding. more in security section]

For **HMAC-SHA-1**, we adapted Conte's [Con06a] SHA-1 [Nat15] implementation for the MSP430 platform, producing a 160-bit output. As for MD5, despite known security vulnerabilities in SHA-1 [WYY05], we consider SHA1 as a potential candidate.

For **HMAC-SHA-2**, we also adapted Conte's [Con06b] SHA-2-256 implementation for the MSP430 platform, producing a 256-bit output. A member of the SHA-2 family of hash functions, SHA-2 is still in use and is recommended by NIST as a cryptographic hash function, though SHA-3 will soon replace it [Nat15]. We did not use SHA-3 because we did not find an implementation of it for the MSP430.

7 Testing

We measured the execution time and RAM usage of our three Mini-MAC implementations running on the Texas Instruments MSP430F5529 microcontroller. In speed and power this device is representative of vehicular ECUs.

7.1 Purpose

The purpose of our tests is to measure the time and space usage of our Mini-MAC, and more generally, to evaluate the performance suitability of Mini-MAC for authenticating messages on the CAN bus.

7.2 Methods

For each implementation we measured code size, memory usage, execution time, and bus utilization, collecting for each implementation metrics from 1000 inputs of various typical sizes (1 byte, 2 bytes, 4 bytes).

We measured code size and RAM usage at compile time using the Texas Instruments Code Composer v6. The RAM usage is know at compile time because there is no dynamic memory allocation.

We measured execution time using a counter register on the MSP430, whose 32kHz clock provides timing values to approximate millisecond accuracy. As a very minor point we note that, due to a limitation in the hardware's support for timing measurements, there may be a +/-0.03 ms inaccuracy in each reading due to the time it takes to read the counter.

We collected statistics on message traffic from a 2010 Toyota Prius with a CAN-bus sniffer program based on an Arudiuno Uno platform and connected via an OBD-II CAN transceiver shield.

7.3 Results

Tables 1, 2, 3 and Figures 4, 5, 6 summarize our results. Throughout, "B" denotes bytes, "b" denotes bits, and "ms" denotes milliseconds.

[need to enhance fig captions]

Table 1: MAC Bus Traffic Addition

Function	**Add. Traffic (b)**
HMAC-MD5 (Group)	128
HMAC-MD5 (Pairwise)	128*n
Lin-MAC	128*n
Mini-MAC	0

Table 3: Approximate Execution Time of Mini-MAC Construction

Hash	**Exec. Time (ms)**
MD5	7.5
SHA1	28.0
SHA2	69.6

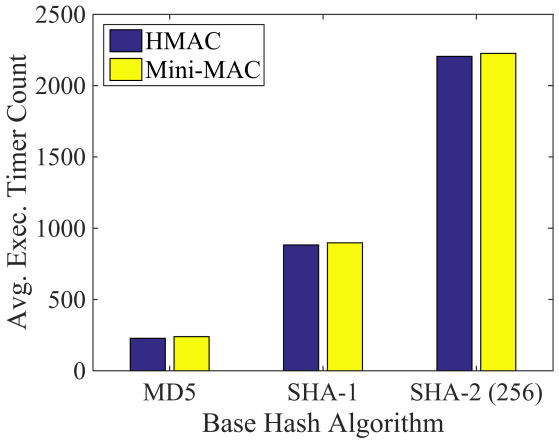

Figure 4: Execution Time Comparison of Mini-MAC Construction

Table 2: Mini-MAC Overhead Relative to HMAC

Hash	Code Size (B)	RAM Use (B)	Execution Time (ms)
MD5	835	5	0.38
SHA-1	850	5	0.42
SHA-256	766	5	0.68

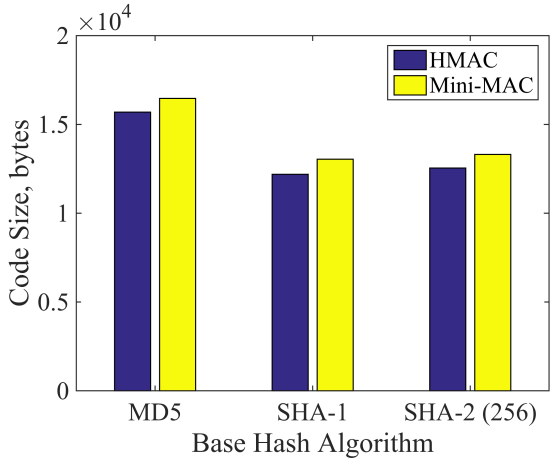

Figure 5: Code Size Comparison of Mini-MAC Code

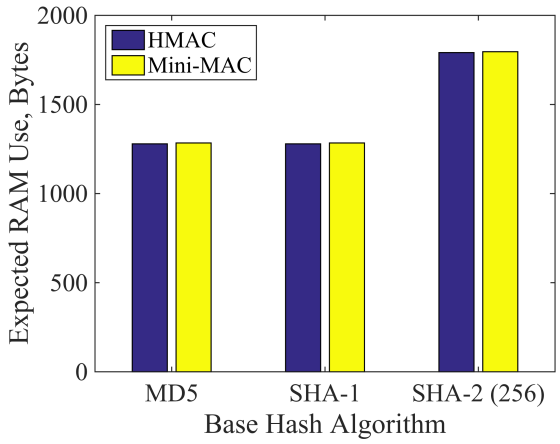

Figure 6: RAM Usage Comparison of Mini-MAC Code

8 Analysis

We analyze Mini-MAC for performance and security. Mini-MAC must be able to add security without compromising the performance of the real-time, safety-critical systems found in vehicles.

8.1 Performance

[from results:] Table 1 shows that even for a group key protocol, traditional HMAC adds at least two extra CAN messages for every data message sent. LinMAC MD5 sends an additional two messages per every user. Mini-MAC sends no additional messages.

Figure 3 shows the execution time comparison for the three HMAC constructions as well as Mini-MAC based on those HMACs. There is very little (approximately 0.68 ms) delay for Mini-MAC relative to HMAC.

Figure 4 shows a comparison of the code size for the three HMACs components as well as Mini-MAC constructions based on those HMACs. The average overhead is roughly 800 B.

Figure 5 shows a RAM usage comparison for the three HMAC constructions as well as the Mini-MAC constructions based on those HMACs. For each hash, the overhead is 5B.

Figures 3-5 show that the overhead is fairly minimal. There is less than 1 kB additional code required and only 5 B extra RAM relative to HMAC. Execution time increases by about 0.4 ms for MD5 and SHA-1, and only by 0.68 ms for SHA-256. For the environment (which typically sees 40 ms between messages) these timings are well within required limits.

Table 3 shows the approximate execution time as calculated from a cycle counter based on a 32kHz clock. Based these data, Mini-MAC-MD5 is the

only hash base that is fast enough for all observed cases. Mini-MAC-SHA-1 would be fast enough in most cases, but would not work for during startup and in the lowest delay cases. Mini-MAC-SHA-256 is too slow for almost every message delay case. The hash implementations used, however, are non-optimized and are designed to be flexible and platform-insensitive. After optimization for the MSP430 platform, it may be possible to reduce execution time.

The overall results are split. The memory and RAM usage numbers are very low, but the execution time is outside required limits in the case of Mini-MAC-SHA-1 and Mini-MAC-SHA-256. Captured data shows that messages are sent approximately every 40 ms. With this in mind, a node must be able to verify the authenticity of a message and respond in that window. This suggests that only the Mini-MAC-MD5 is fast enough to be used in this application.

8.2 Security

Mini-MAC counters replay and masquerade attacks by forcing an attacker to wait a relatively long period of time before a MAC is repeated. Table 5 shows a comparision of the time-to-defeat (how long before the MACs are re-used) of the various hash bases and counter sizes. R is the rollover counter size in bits, M is the message counter size in bits, "Msg BR" is the number of messages before repeat and "Min BR" is the time in minutes before repeat at a data rate of 40 messages/second. This table (and Table 5) show the time-to-defeat for various configurations of Mini-MAC. This is the number of messages required before a MAC repeats multiplied by the maximum message rate per second.

The time to guess a 32-bit MAC correctly is much shorter than the time to repeat for most cases—only 27.3 minutes, on average, for an exhaustive search (brute force) guessing attack. This means that the most efficient use of resources is the smallest counter combination that withstands the time of a brute force attack. Therefore the 16-bit message counter is the best choice from the above because it will ensure a replay attack takes at least longer than the average time to execute a brute force attack on the MAC but will not consume more resources than is necessary.

The message data captured suggests an average message rate of approximately 25 messages/second. Some messages occur, however, at up to 40 messages/second, although this is unlikely. In the event that an attacker floods the system with a higher rate to cycle through the counters more quickly, ECUs on the bus could easily identify an illegitimate user. Figure 5 shows the probability density function of the time between messages. This relates the number of messages to defeat to a time-to-defeat figure by approximating how many messages per second an attacker can send.

A key to Mini-MAC is that it uses the slow message rate of the CAN bus as an advantage. Malicious attackers must wait a long time (Table 4) to gain enough information to repeat a MAC. The use of message history in Mini-MAC ensures that even if an attacker has a long time to watch the bus, they will not be able to simply replay a message. It is worth noting as well that Table 4 shows the times for a stream of repeated messages, a case which is unlikely to repeat for the duration required to gain enough bits to repeat a MAC. Attackers cannot simply flood the network with messages designed to acquire responses more quickly because ECUs should be able to easily identify this behavior based on message delay statistics.

There are some attacks that will defeat Mini-MAC. Perhaps the easiest way to defeat any CAN security mechanism is by flooding the bus. The attacker does not need to try and break any security, but by preventing ECUs from talking to each other, the attack succeeds as the car will not be able to function properly. Similarly, if an ECU is flashed with corrupted firmware, it does not need the correct group keys to launch an attack. It needs only to wait long enough to see the counters roll over. Most people keep the same automobile for many years, so even if this attack requires several years to gather enough information, it will still eventually succeed. Mini-MAC is useful against a resourceful attacker, but not a patient one.

Table 4: Time-to-Defeat for Various Configurations

Hash	$\text{Counter}_{\text{R}}$ (b)	$\text{Counter}_{\text{M}}$ (b)	Msg. Before Repeat	Min. Before Repeat
MD5	7	8	24480	10.2
MD5	7	16	6291360	2621.4
MD5	7	32	4.12317E+11	171798691.8
SHA1	8	8	32640	13.6
SHA1	8	16	8388480	3495.2
SHA1	8	32	5.49756E+11	229064922.4
SHA256	8	8	57120	23.8
SHA256	8	16	14679840	6116.6
SHA257	8	32	9.62073E+11	400863614.2

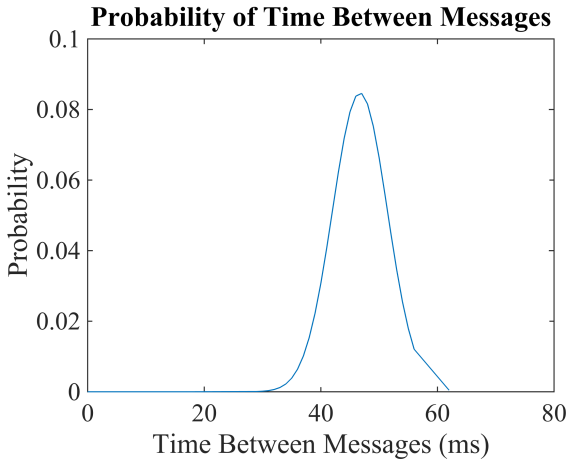

Figure 7: Probability Density Function of Message Delay – ECUs can easily be able to identify nodes spamming messages

9 Discussion

In this section we discuss how to resynchronize ECUs, how to lengthen the mini-MAC tag as an optional improvement, and we list some open problems.

9.1 Resynchronization

Our mini-MAC proposal requires the ECUs to have synchronized counters and message-history states. Therefore, a mechanism is needed to resynchronize the ECUs in case they ever lose synchronization, as might happen, for example, by a fault in the ECU or a disruption in message transmission.

Two common solutions are to reset the state to a specified initial state, or for one ECU to select a new state and communicate that state to the other ECUs (encrypted by a shared secondary communication encryption key). [do you have a ref?]

Instead, for enhanced security we propose that each ECU periodically save its state in persistent memory. In the initial attempt to resynchronize, each ECU loads its most recent state. If that fails, then the aforementioned mechanisms could be applied. Section 6.1 explains how message history helps guard against replay attacks upon resynchronization.

A limitation of the CAN bus is that it provides no mechanism for detecting when ECUs are out of synchronization. In some cases, by monitoring observable conditions on the bus, an ECU might detect that another ECU is not responding to a message, which might be the result of a synchronization failure. A tradeoff in our design is that,

Table 5: Expected time for some spoofed message to be accepted by an ECU for various tag lengths, assuming the adversary injects 40 forged messages per second.

Tag Length (b)	Time to Find Collision
16	6.40s
24	1.70m
32	27.30m
40	7.28h
48	4.85d

by using counters and message history to athenticate messages, mini-MAC increases the opportunities for possible synchronization failures.

9.2 Lengthening the Mini-MAC Tag

Optionally, it is possible to lengthen the Mini-MAC tag by using the two bytes of space allocated for the CRC field in the CAM frame (see Figure 1), as suggested by Woo et al. [WJL15] in a related proposal. Because a MAC detects transmission errors (in fact, better than a simpler CRC), there is no need for a CRC in addition to a MAC.

Increasing the tag length greatly increases the time required for an adversary to forge a valid tag by finding a collision in the Mini-MAC by exhaustive search. Table 5 gives the expected time for an adversary to send a forged message that will be accepted by another ECU, for various tag lengths. Here, the main limiting assumption is the frequency with which messages can be sent on the CAN bus (approximately 25 messages per second. [in table, change b to bits] For example, increasing the tag from 32 to 48 bits increases this time from approximately 27.3 minutes to over four days.

To implement this strategy one could modify the lower-level code in the CAN network stack, either to perform the MAC calculation there or to open the CRC field to the application level to calculate the MAC.

9.3 Open Problems

Our engineering decisions are driven by a desire to improve vehicular security by adding authentication to the CAN bus, without increasing bus traffic or delaying messages, and without making any disruptive changes. The egregious state of vehicular security, however, demands a radical disruptive redesign of vehicular computer networks carried out including security as a foundational design requirement. [should we cite any discussion like this from someone else? are we the first to say so?]

Design ideas for a replacement network to the CAN bus include the following: (1) Use a well-established high-speed network (such as 802.3 Ethernet) on which standard security mechanisms (such as IPsec) can be deployed. (2) Segegrate nodes on the bus into task-defined groups. (3) Protect access to the bus by physically separating critcal and non-critical systems. In particular, it should not be possible for maleware or faults in entertainment or Bluetooth systems to affect braking, steering, or acceleration.

A separate related problem is to detect vehicular network intrusions. [ref?] A challenge of such work is that there is no good reponse of what to do if an intrusion is detected other than to shut down the vehicle safely.

The Car-to-X network [FBZ+08] is an emerging interconnected collection of vehicles, buildings, signs, and road infrastructure to reduce congestion and enable more efficient traffic control. Cars of the future will have to be able to communicate securely with objects on such networks, requiring authentication and key management beyond Mini-MAC.

10 Conclusion

We propose Mini-MAC, the first variable-length MAC protocol for the CAN bus that adds no bus traffic overhead, allowing it to be used in vehicular systems with time-sensitive messages. The truncated HMAC protects against message injection by adversaries who do not know the ECU keys. The counter and message history protect against replay attacks.

Limited message size, the need not to delay messages, the limited computational power of the ECUs, and the relative ease of gaining access to the bus severely restrict how well the CAN bus can be protected. Mini-MAC meaningfully raises the bar on vehicular security, approaching (we conjecture) the limits of what is possible for authentication strength in this highly constrained environment.

11 Acknowledgments

Schmandt and Sherman were supported in part by the National Science Foundation (NSF) under SFS grant 1241576. Sherman was also supported under a subcontract of NSF INSuRE grant 1344369, and by the Department of Defense under CAE-R grant H98230-15-10294.

I don't have Dr. Banerjee's support information yet

References

[BCK96] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. Keying hash functions for message authentication. In *Proceedings of the 16th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '96, pages 1–15, London, UK, UK, 1996. Springer-Verlag.

[CMK+11] Stephen Checkoway, Damon McCoy, Brian Kantor, Danny Anderson, Hovav Shacham, Stefan Savage, Karl Koscher, Alexei Czeskis, Franziska Roesner, Tadayoshi Kohno, et al. Comprehensive experimental analyses of automotive attack surfaces. In *USENIX Security Symposium*. San Francisco, 2011.

[Con06a] Brad Conte. Implementation of SHA-1 in C, 2006.

[Con06b] Brad Conte. Implementation of SHA-256 in C, 2006.

[FBZ+08] Anreas Festag, Roberto Baldessari, Wenhui Zhang, Long Le, Amardeo Sarma, and Fukukawa Masatoshi. Car-2-X communications for safety and infotainment in Europe. *NEC Technical Journal*, 3(1):21–26, 2008.

[LSV12] Chung-Wei Lin and A. Sangiovanni-Vincentelli. Cyber-Security for the controller area network (CAN) communication protocol. In *Proceedings of the 2012 International Conference on Cyber Security (CyberSecurity)*, pages 1–7, Dec 2012.

[Nat08] National Institute of Standards and Technology. *FIPS PUB 198-1: The Keyed-Hash Message Authentication Code (HMAC)*. July 2008.

[Nat15] National Institute of Standards and Technology. *FIPS PUB 180-4: Secure Hash Standard*. August 2015.

[Pes09] Alexander Peslyak. A portable, fast, and free implementation of the MD5 message-digest algorithm (RFC 1321), 2009.

[Riv92] Ronald Rivest. IETF RFC 1321: The MD5 message-digest algorithm, April 1992.

[WJL15] S. Woo, Hyo Jin Jo, and Dong Hoon Lee. A practical wireless attack on the connected car and security protocol for in-vehicle CAN. *IEEE Transactions on Intelligent Transportation Systems*, 16(2):993–1006, April 2015.

[WY05] Xiaoyun Wang and Hongbo Yu. How to break MD5 and other hash functions. In *Proceedings of the 24th Annual International Conference on Theory and Applications of Cryptographic Techniques*, EUROCRYPT '05, pages 19–35, Berlin, Heidelberg, 2005. Springer.

[WYY05] Xiaoyun Wang, YiqunLisa Yin, and Hongbo Yu. Finding collisions in the full sha-1. In Victor Shoup, editor, *Advances in Cryptology CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*, pages 17–36. Springer Berlin Heidelberg, 2005.

[XLL+15] Yong Xie, Liangjiao Liu, Renfa Li, Jianqiang Hu, Yong Han, and Xin Peng. Security-aware signal packing algorithm for CAN-based automotive cyber-physical systems. *Automatica Sinica, IEEE/CAA Journal of*, 2(4):422–430, October 2015.

[ZM14] R. Zalman and A. Mayer. A secure but still safe and low cost automotive communication technique. In *Design Automation Conference*

(DAC), 2014 51st ACM/EDAC/IEEE, pages 1–5, June 2014.

Preliminary draft to be submitted to *Cryptologia*. January 26, 2016.