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 23, 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.

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 do so while the car is being serviced or by entering

[1] kilobits per second (kbs).

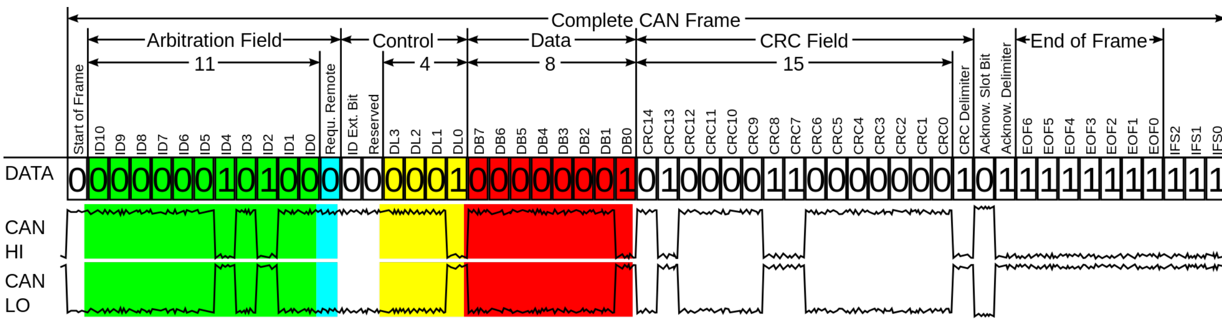

Figure 1: The CAN frame [] [need to cite fig]. Each message on the CAN bus is a structured sequence of 55–111 bits including 8–64 data bits.

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

$$H((K_0 \oplus \text{opad}) \,\|\, H((K_0 \oplus \text{ipad}) \,\|\, \text{message}), \qquad (1)$$

where K_0 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 $\|$ 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, timestamped 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 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 can be broken into three conceptual tiers–archictecture, design and implementation. This section describes Mini-MAC at each of these levels. The architecture describes the high-level critical points that differentiate Mini-MAC from other MAC protocols, while the design level describes the algorithm used to generate the resulting MAC. Finally, this section describes how Mini-MAC is implemented with various hash functions.

While a MAC such as HMAC will defeat a masquerade attack, it will not be able to defeat a replay attack, as the previously recorded message will have a valid MAC. To address this, some time-based token, frequently a counter, can be added to the computation of the MAC. The resulting tag is unique to the message and the time it is sent, which prevents an illegitimate node from simply replaying a message seen in the past.

For the automotive environment, the time-based HMAC has one significant downside - the size of packets on the CAN bus is very small (64B) and the size of the HMAC bit string is, depending on the hash function used, at least twice that size. In order to use normal HMAC, bus traffic would go up some large factor determined by the size of the hash used.

Mini-MAC is a variable-length Message Authentication Code protocol based on HMAC. It selects an output size to match the available space in a given CAN message. It uses secret keys as well as variable-length counters to seed and condition the HMAC to guarantee against repeated MACs.

The core principle of Mini-MAC is that in the automotive environment attackers have a small window of time to break the network. With that in mind, it is possible to design a security mechanism that is capable of defending the network on that very short scale of time without breaking real-time or processing capability constraints.

6.1 Architecture

There are four key points that help define Mini-MAC:

- Variable-size output to fit the CAN packet
- Group-shared keys
- A split counter used for MAC seeding and truncation
- Message history to confuse MAC output.

Note: Do you like the itemization of these points, or would it be better organized in a paragraph?

Mini-MAC uses group-shared keys (Figure 2) instead of pairwise keys. For example, the group comprised of ECUs 1, 2, and 4 share key 1. The use of group keys means that a message need be sent only once and the entire group which needs the message can verify the sender rather than having a separate MAC sent for each recipient. This means Mini-MAC does not need to send any additional message in order to provide authentication, which meets the design requirements for bus traffic overhead. Group key distribution is possible because at the time of system design the engineers know exactly which ECUs will need to communicate with which others. There does not need to be a dynamic group update function as there are no circumstances in which a group should change.

The downside of using group keys is that if one member becomes compromised, that member has the

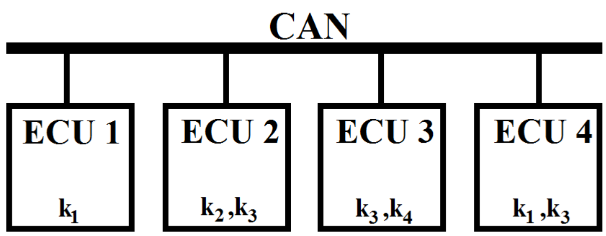

Figure 2: Mini-MAC Key Distribution

ability to send illegitimate messages to every other member of the group rather than just one other node as would be the case in a pair-wise key distribution. The choice to use a group key distribution was made for Mini-MAC because security must be balanced with efficiency, and the bus traffic increase resulting from pair-wise key distribution may be too high to meet real-time constraints. There may be many groups composed of only two nodes, in which case security is not reduced at all compared to a pair-wise protocol.

A split counter is used to alter and select the final MAC from the HMAC. The low-order bits are used to seed the HMAC, while the high-order bits are used to select the starting bit in the HMAC from which the Mini-MAC is drawn.

Message traffic history is used as a post-HMAC confusion step – the previous messages sent on the sender's ID are used to change the resulting HMAC. This prevents the same MAC being used after the counters roll over. Unless the same message (or message pattern) is repeated exactly, the same MAC will not be issued, even on identical counter values.

6.2 Design

Note: Is this a better way of incorporating the equations into the text?
I don't like the figure – do you have a recommendation for software to create good-looking figures? I've just been using MS Paint and would like to tidy up a few things before submitting.

One of the most important aspects of HMAC is that it allows the use of a wide range of iterative hash functions as its base. Mini-MAC, being based on

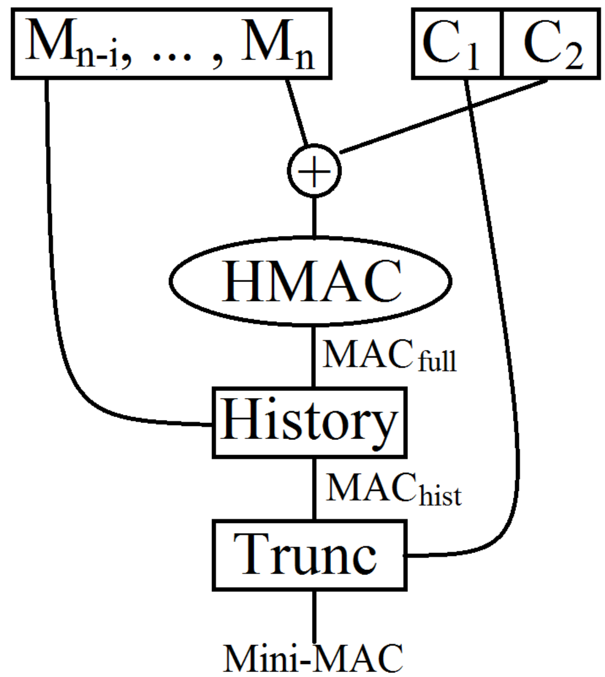

Figure 3: Mini-MAC Construction Diagram

HMAC itself, similarly allows various hash functions as a base.

The computation of the Mini-MAC begins with the creation of an input string,

$$\text{Input} = \text{Message}_n \oplus \text{Counter}_{\text{msg}} \tag{2}$$

which is used to generate an HMAC value. The HMAC calculation

$$\text{MAC}_{\text{full}} = \text{HMAC}(\text{Input}) \tag{3}$$

results in a full-size MAC without any message history input. For $h =$ the number of messages in history to use,

$$\text{For } i = 1 : h,\ \text{MAC}_{\text{hist}} = \text{MAC}_{\text{full}} \oplus \text{Message}_{n-i} \tag{4}$$

where MAC_{hist} is a full-size MAC with added message history information. This value is finally truncated according to the trunc function, defined as

$$\mathrm{MAC}_{\mathrm{mini}} = \mathrm{MAC}_{\mathrm{full}}(l, l + s) \quad (5)$$

where l is the rollover counter, which ticks when the message counter rolls over, and s is the available size in the CAN message for the Mini-MAC. The result of the two-counter system is that a new MAC is generated for each value of the message counter, and from that MAC, the bits starting with the bit addressed by the rollover counter are taken as the Mini-MAC. This way, when the message counter rolls over, the bit start location shifts so the same Mini-MAC is not re-used until the rollover counter rolls over.

6.3 Implementation

Mini-MAC was implemented with three hash functions–MD5, SHA1, and SHA2. The varying performance and security characteristics of these hash functions provide end users with a spectrum of solutions to choose from without needing to vary the Mini-MAC computation.

HMAC-MD5 We adapted Peslyak's implementation of MD5 for the MSP430 platform. MD5 produces a 128-bit output from a variable length message. Since 2004, the security of MD5 has been severely compromised [WY05] – however, tests showed that it was the fastest HMAC construction and for that reason only it was used as a basis for Mini-MAC. It should be noted that any other hash function could be used, but the test was interested in computation speed more than any other metric [Riv92]

HMAC-SHA1 We adapted Brad Conte's SHA-1 implementation for the MSP430 platform [Con06a]. SHA-1 produces a 160-bit output value from variable length inputs. Similar to MD5, security vulnerabilities have been found in SHA-1 [WYY05], but it is still used in many applications and will be for the immediate future [Nat15].

HMAC-SHA256 We also adapted Brad Conte's SHA-256 implementation for the MSP430 platform [Con06b]. SHA-256 is a member of the SHA-2 family of hash functions. This family produces fixed length output values from variable length input sequences. SHA-2 is still in use and is recommended by NIST as a secure hash function, but SHA-3 will soon replace it [Nat15].

7 Testing

We experimented with three Mini-MAC implementations on the Texas Instruments MSP430F5529 microcontroller. The speed and power of this device makes it an appropriate test platform for CAN security software.

7.1 Purpose

The purpose of our tests is to evaluate the suitability of HMAC-based message authentication for nodes on the CAN bus. Small memory, RAM and running-time performance overhead is ideal, on the basis that ECUs are low-resource devices. The three hash functions selected are compared for two reasons 1) Performance and security are functions of the hash function selected as a base 2) Overlaying Mini-MAC onto HMAC should incur a minimal performance penalty regardless of the hash selected.

7.2 Methods

The tests performed execute the Mini-MAC construction protocol over a variety of inputs, repeated 1000 times. This test characterizes the performance of the protocol. The metrics recorded are code size, memory usage, execution time, and bus utilization.

Statistics on message traffic were collected 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.

A counter register on the MSP430 generates execution time values. A 32kHz clock increments this counter, which provides approximate millisecond execution time values. There may be a +/-0.03ms inaccuracy in this value depending on the time it takes to read the counter.

RAM and memory usage figures are generated at compile-time. Texas Instruments Code Composer v6 provides values for both after code is compiled and flashed to the MSP430 hardware.

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

7.3 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. Lin-MAC MD5 sends an additional two messages per every user. Mini-MAC sends no additional messages. For this table, B represents a value in bytes, b is a value in bits, while ms represents a value in milliseconds.

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

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.

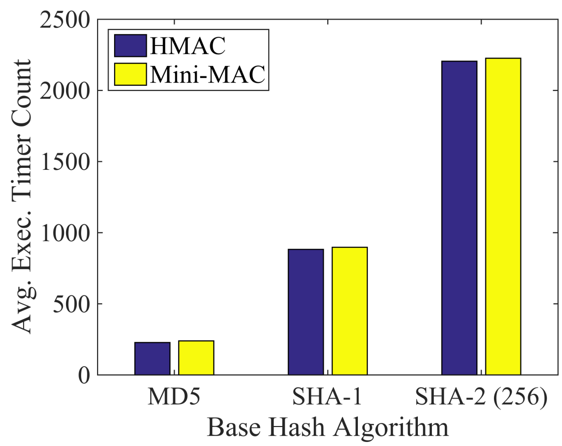

Figure 4: Execution Time Comparison of Mini-MAC Construction

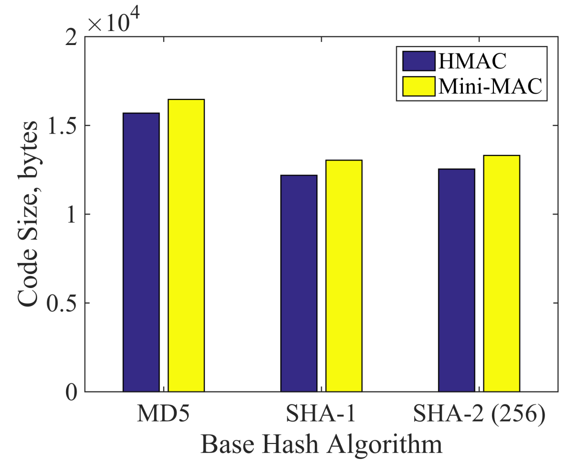

Figure 5: Code Size Comparison of Mini-MAC Code

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

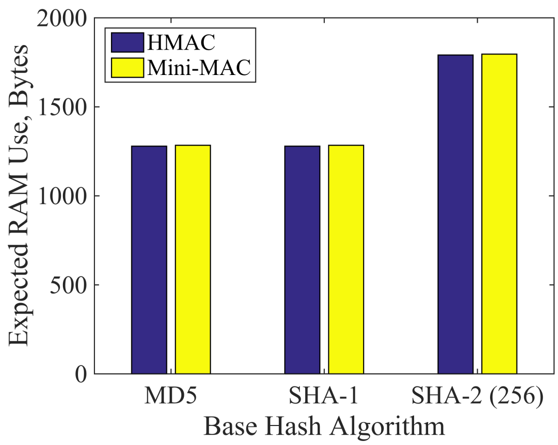

Figure 6: RAM Usage Comparison of Mini-MAC Code

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

Table 4: Time-to-Defeat for Various Configurations

Hash	**Counter$_\mathbf{R}$ (b)**	**Counter$_\mathbf{M}$ (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

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

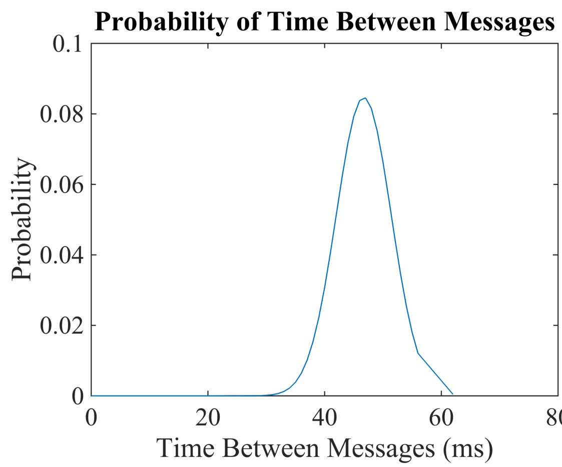

Figure 7: Probability Density Function of Message Delay – ECUs can easily be able to identify nodes spamming messages

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.

9 Discussion

In this section we discuss how to resynchronize ECUs, how to lengthen the mini-MAC tag, 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. [how do ECUs know that they are out of synch?]

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.

9.2 Lengthening the Mini-MAC Tag

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 time to find a Mini-MAC collision for various tag lengths. [under what assuptions?? algorithm, machine, etc?] For example, increasing the tag from 32 to 48 bits increases this time from approximately 27.3 minutes to over four days.

Table 5: Time to find collision for various tag lengths [explain]

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

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. [refs?]

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.

[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*, EURO-

CRYPT '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 23, 2016.