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

December 27, 2015

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.

The CAN bus is a low-speed network between electronic control units (ECUs). It is extremely vulnerable to malicious actors with bus access. Traditionally, Message Authentication Codes (MACs) verify the identity of the sender of a message, and variants prevent message replay attacks; however, standard or time-seeded MACs are unsuitable for use on the CAN bus because of short data payload size restrictions. Our work with a smaller footprint alternative to the traditional MAC raises the bar of vehicle network security. Mini-MAC causes no increase in bus traffic, and incurs a very small performance penalty relative to the provably secure HMAC. It is the first work to combine these two tenets for vehicle computer networks. Mini-MAC is based on a time-seeded HMAC augmented with message history and truncated to fit available message space. Mini-MAC is not suitable for long-term defense against dedicated attackers, but rather is designed to protect the physical safety of drivers against opportunistic attackers in the proposed Car-to-X environment.

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

1 Introduction

The 2015 Black Hat conference featured a presentation in car hacking by Charlie Miller and Chris Valasek [REF] which demonstarted how easy it is to gain access to a car's computer network. Miller and Valasek were able to gain full access to the engine, braking, steering, and entertainment systems (as well as other sub-systems) of a new Jeep by leveraging exploits in the secure WiFi implementation used onboard, a vulnerability in Sprint's cellular network, and finally by re-writing the firmware of a controller connected to the in-car entertainment unit.

Message authentication is a fundamental element of a secure communication link, but traditional protocols are unsuitable for the CAN bus due to limited resources and small packet size. This work shows that message authentication can be practically implemented in a resource constrained environment such as a vehicle computer network with Mini-MAC, a variable-size message authentication code protocol.

Mini-MAC improves on previous work in several ways. It improves on previously published work such as Lin-MAC [LSV12] by requiring no increase in bus traffic on the CAN bus. It also improves on previous work in that it is easily implementable and requires

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

no firmware modification or additional hardware.

The construction of Mini-MAC involves several factors designed to combat replay and masquerade attacks. Mini-MAC uses a buffer of past messages to compute the next MAC as well as a counter to ensure a different MAC even for repeated messages. Like the HMAC it is based on, Mini-MAC also uses a secret key to seed the underlying hash functions.

The end goal for security is usually to make it too expensive (in terms of time, computing resources, or otherwise) for an attacker to defeat the defensive measures in place, be it standard cryptography or a low-overhead MAC protocol such as this work. The key difference between this work and traditional network security is an attack budget dominated by the very short time available to the attacker. 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.

Specifically, this project presents several contributions:

- Mini-MAC, a small-size authentication protocol suitable for systems like the CAN bus
- A property of Mini-MAC which demonstrates a quantifiable improvement in CAN bus security

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 hash functions as explained by, for example, the National Institute of Standards and Technology FIPS Publication 180 [Nat15] and Stinson Note: You mentioned Stinson on the last draft notes – this is the book Cryptography - Theory and Practice?.

2.1 ECUs

The electronic control units 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 the 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 an 8-byte data payload as well as some control bits. The 8-byte payload is the most important number, 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 amount of messages (61%) use no more than 4 bytes of data in the payload.

2.3 Bus Access

Despite many demonstrated security flaws, automotive manufacturers are hesitant to acknowledge the vulnerabilities inherent to the CAN bus. One reason they cite is the difficulty in gaining access to the bus. It can be difficult—the most typical way to gain access is through a physical connection through the On-Board Diagnostic port (OBD-II), which is located underneath the steering wheel, physical connection to the CAN bus can be hidden in an unexposed part of the car's wiring. A driver might notice a hardware module sticking out from beneath the steering wheel, but may not notice a similar module attached inside part of the engine compartment. An attacker would have ample time to place such a device in a car parked in a garage overnight or when a vehicle goes to a shop for maintenance.

[1]kilobits per second

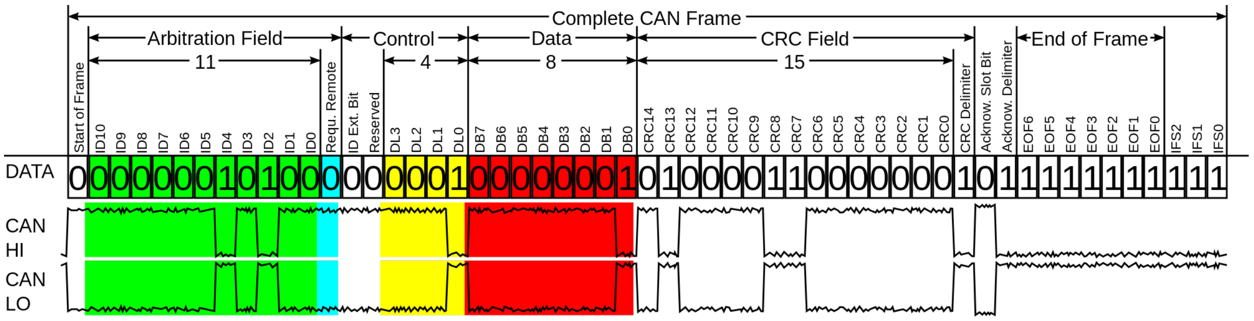

Figure 1: The CAN FrameTest Caption Goes Here

There are various ways to gain access to the bus without physically attaching a device as well. It has been shown that it is possible to gain remote access to these vehicles (or to corrupt / rewrite firmware on an ECU) through wireless channels such as the cellular modem or bluetooth radio. Checkowa demonstrated gaining bus access by packing code into a WMA audio file played on the car stereo [CMK+11].

2.4 Message Authentication Codes

Message Authentication Codes (MACs) are bit strings used to verify the identity of the sender to check the legitimacy of a message. These codes are usually relatively small compared to the length of the message.

The Keyed-Hash Message Authentication Code (HMAC) [BCK96] [Nat08] is by the far the most famous and most popular MAC construction protocol. It is so widely used for two reasons. 1) The security of HMAC is mathematically linked to the security of the underlying hash function, which makes it relatively easy to prove secure and 2) it is extremely easy to plug in various hash functions, making it easy to update to a new implementation if issues are found in the currently used hash.

The equation for calculating the HMAC is as follows:

$$H((K_0 \oplus \text{opad}) \parallel H((K_0 \oplus \text{ipad}) \parallel \text{message}) \tag{1}$$

Where K_0 represents the key, which is sized depending on the underlying hash, and opad and ipad are the outer and inner hash padding strings respectively. These strings are constant strings represented by 0x5C and 0x36 repeated until the hash input string is of the appropriate size. The term H represents the hash function used to generate the HMAC. The symbols $\oplus$ and $\parallel$ represent the XOR operation and concatenation respectively. [Nat08]

Note: I saw your note about the trust issues with shared-key MAC. I think a comment on this fits better in section 6. In previous work I talk about how the other work uses pair-wise keys and that this causes bus utilization problems, so I want to make my case for group keys in the context of security vs efficiency, rather than in background

2.5 Car-to-X

The Car-to-X network is the collection of vehicles, buildings, signs and road infrastructure that are connected to each other to enable more efficient traffic control and congestion relief. These networks are currently in fledgling states at best but will become more prevalent over time [FBZ+08].

3 Problem Statement

The task of this work is to create an authentication mechanism suitable for use on the CAN bus in vehicular environments. Given the context, this mechanism must provide some measurable increase in security over the standard non-secured bus. It must not increase bus traffic, and it must not violate message delay constraints.

In captured test data for a 2010 Toyota Prius, over 60% of messages contain no more than four data bytes. This leaves four bytes of space for a MAC in the most common case.

Given these limitations, the problem addressed is how to add as much security as possible, rather than to make the system secure by conventional definition.

4 Previous Work

The most complete work specifically discussing the usage of MACs in the automotive CAN network produced the Lin-MAC construction [LSV12]. Lin-MAC presents the idea that a MAC should be based not only on the secret key, but on a counter as well, as discussed in Section 3. It uses a pair-wise key distribution scheme where each node shares a secret key with each other node.

There are some issues with this protocol. Consider that an entity Bob wishes to send the same message to entities Alice and Charlie. He must send the message along with two MACs – one computed with the secret key he shares with Alice and one computed with the secret key he shares with Charlie. For every recipient, Bob must send a MAC. This will increase bus utilization, even in the case of one recipient, as the MAC must be transmitted in an additional CAN frame.

The other issue is that the tags must fit into 64 bits, the size of the data payload. HMAC produces a much larger output (depending on the underlying hash) and transmitting this MAC will greatly drive up bus utilization. For example, the smallest HMAC result used in this work is 128 bits, produced by HMAC-MD5. Transmitting the entire HMAC would require an additional two CAN frames per every message.

Other recent CAN security projects suffer from similar problems. Proposed protocols by Woo et al [WJL15], Zalman et al [ZM14], and Xie et al [XLL^{+}15] all use pair-wise key distribution which increases bus utilization for a messages sent to multiple recipients.

The work by Woo et al requires re-writing of firmware or hardware redesign of CAN transceivers. It packs the MAC into an extended ID field not used by all ECUs and the CRC field in the CAN trailer. Woo also assumes a much more powerful message processor than is assumed in this work.

Zalman et al uses a timestamp value to seed the MAC, but this technique is by the author's admission vulnerable to delay in the network. Zalman also uses a fixed-size MAC that may be too large to fit in one CAN frame, causing excessive bus overhead.

Xie et al proposes packing multiple messages into one CAN frame with a MAC in order to add security without increasing bus utilization, but this may breach message delay requirements. It also assumes messages are small enough to fit into a CAN frame with other messages and an appropriately-sized MAC.

5 Adversaries

We consider three classes of adversaries for this problem.

- Type 1 (*Strongest adversary*): A corrupted ECU that has access to a valid key for the MAC it wishes to generate.
- Type 2 (*Strong adversary*): A corrupted ECU without access to valid keys.
- Type 3 (*Weak adversary*): A party with access to the CAN bus but without any valid keys.

This project aims to defend against adversaries of Type 2 and Type 3. A Type 1 adversary with group keys would be able to spoof any message it wished. However, the process of an ECU being corrupted (e.g., firmware being flashed) may lose knowledge of keys. The existence of this strongest class of adversary is dependent upon some kind of external command interface which would allow firmware updates, or some kind of modified hardware (e.g., hardware trojan, counterfeit IC).

Adversaries are not without some difficulties - there are more ECUs in vehicles every year, and each ECU has a unique message ID and data format, which manufacturers share with automakers but not the public. These both must be known to orchestrate a targeted attack, and aside from getting the information from the ECU manufacturer or automaker, they can only be discovered with time consuming bus analysis or the costly measure of taking a vehicle apart and inspecting individual ECUs. However, for this work we assume that adversaries have full knowledge of all bus messages and the relevant transmission protocols.

Specifically, the adversaries we intend to defend against include those a driver may encounter on the road which may present an immediate threat to the safety of the driver or other road users. These attackers may be pieces of software in other automobiles that attempt to gain remote access via WiFi or Bluetooth, corrupted smart roadside technology with similar attack vectors, or, more generally, any malicious member of the Car-to-X network. A key feature of the attack vectors concerned is the range – typically, these attacks require close proximity, which for a moving vehicle, may not be possible for more than a short period of time.

Because of this attacker profile, for certain attacks it may be sufficient to defend the CAN network for a much shorter period of time than for traditional networks. While there are no data to suggest a hard number, it should be safe to say this attack time window is measured in minutes instead of hours or days, as an attack on a traditional network would be.

Attackers may be motivated for various reasons, but the most obvious are for theft of vehicle or to cause damage to the car or people in or around it. If an attacker can gain remote control of a car, he or she may be able to gain entry when the car is locked. Similarly, this remote control can give the attacker the ability to crash the car on purpose. Mischief is also a valid motivation for work like this—an attacker could simply wish to prevent a driver from using the vehicle, in which case a physical attack such as slashing a tire, cutting wiring, or draining fuel would be enough to accomplish their goal.

In addition to possessing all knowledge of ECU message IDs and formats, we assume that the attackers have reliable bus access and unlimited computing power.

In a typical CAN bus, each ECU is implicitly trusted, in that there are no explicit protocols or rules governing authentication or verification. In contrast to this, our system assumes every message may potentially be from an illegitimate source and must authenticate the sender before acting.

We assume attackers are capable of monitoring all bus traffic at any time and are also capable of injecting a message onto the bus at any time and with any frequency.

6 Mini-MAC

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

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 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.

Two counters are used to alter and select the final MAC from the HMAC. The message counter is used to seed the HMAC, while the rollover counter is 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.

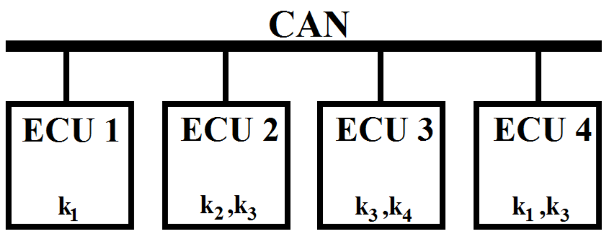

Figure 2: Mini-MAC Key Distribution

6.2 Design

Note: I don't have a solution to dealing with faults at this time. I believe there are some good ideas in previous work that I will take another look at
Update: The previous work tends not to mention dealing with faults. In similar systems, there is a mechanism for setting counters to specific values on a fault, which should allow re-sync between sender and recipient. In practice, end users more commonly reset counters to 0 and start from the beginning. Even though this is less secure the procedure is more simple. I'm not sure how I want to address it. Do you have any thoughts?

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

The calculation of Mini-MAC is only slightly more involved than in the HMAC construction—before computing the HMAC, the input is XORed with a counter, and after the HMAC is computed the result is XORed with previous messages and trimmed.

The full construction protocol is shown below in Equation 2 through Equation 5, where n is the number of the message in a sequence of messages, $n = 0$ being the most recent message, h is the number of messages from history to be used for MAC confusion, l the starting bit to select the Mini-MAC from within the full-size MAC, and s being the size of the Mini-MAC in bits.

The value of l is determined by a second counter designated the rollover counter, which ticks when the message counter rolls over. The result of this 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.

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

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

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

$$\text{MAC}_{\text{mini}} = \text{MAC}_{\text{full}}[l : l + s] \tag{5}$$

Equation 2 is a simple XOR operation of the message to be sent and the message counter. Equation 3 takes the result of Equation 2 and performs the HMAC on it as detailed in Equation 1. The HMAC is then XORed with previous messages, up to h messages in history, as shown in Equation 4. Finally, in Equation 5, the variable size Mini-MAC is extracted from the full-size, history-based MAC computed in Equation 4.

6.3 Implementation

Mini-MAC can be used with a variety of hash functions such as MD5, SHA1, or SHA2. This work uses these three common hash functions as a base for HMAC to compute the Mini-MAC. Their varying performance and security characteristics provide end users with a spectrum of solutions to choose from without needing to vary the Mini-MAC computation.

HMAC-MD5 The MD5 implementation tested for this project was originally written by Alexander Peslyak [Pes09] and adapted for the MSP430 platform by the authors.

——-

Alexander Peslyak [Pes09] wrote the original MD5 implementation used in this project, and it has been adapted for use on the MSP430 platform by the authors. Note: Which of the above two sounds better? I can change the SHA1 and SHA2 to match.

MD5 produces a 128-bit output value 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 The HMAC-SHA-1 implementation used for this project was originally written by Brad Conte [Con06a] and adapted for the MSP430 platform by the authors. 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 The HMAC-SHA-256 implementation used in this project was originally written by Brad Conte [Con06b] and was adapted for the MSP430 platform by the authors. 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 tested 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 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

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

incur a minimal performance penalty regardless of the hash selected.

7.2 Methods

The tests performed were executing the Mini-MAC construction protocol over a variety of inputs, repeated 1000 times. This test, although simple, is sufficient to characterize the performance of the protocol. The metrics recorded are code size, memory usage, execution speed, 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 generated.

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. B represents a value in bytes, b is a value in bits, while ms represents a value in milliseconds.

Table 3: Approximate Execution Time of Mini-MAC Construction

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

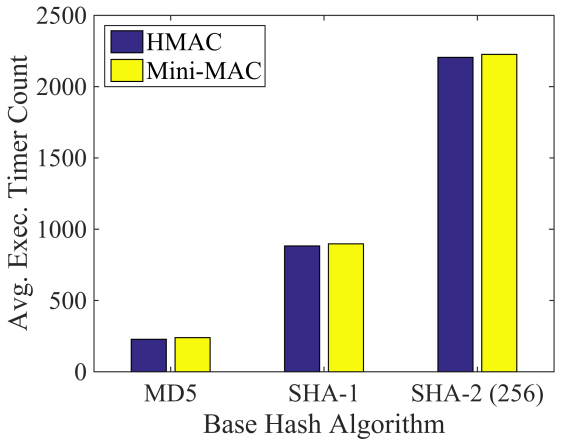

Figure 3: Execution Time Comparison of Mini-MAC Construction

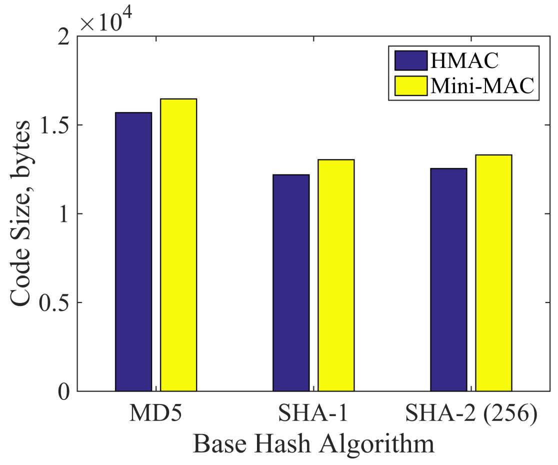

Figure 4: Code Size Comparison of Mini-MAC Code

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

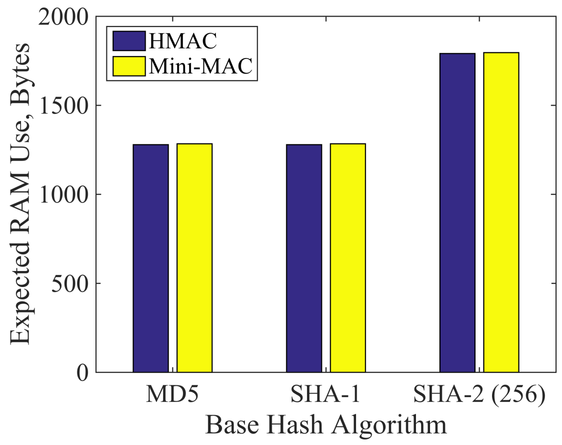

Figure 5: RAM Usage Comparison of Mini-MAC Code

8 Analysis

Not only must Mini-MAC add security, but it must be able to actually run on resource-limited ECUs operating under real-time constraints. It is from these two perspectives that the results have to be analyzed.

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. Note: Could you please clarify your comments on this paragraph?

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.

Table 4: Time-to-Defeat for Various Configurations

Hash	**Counter$_R$ (b)**	**Counter$_M$ (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

8.2 Security

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

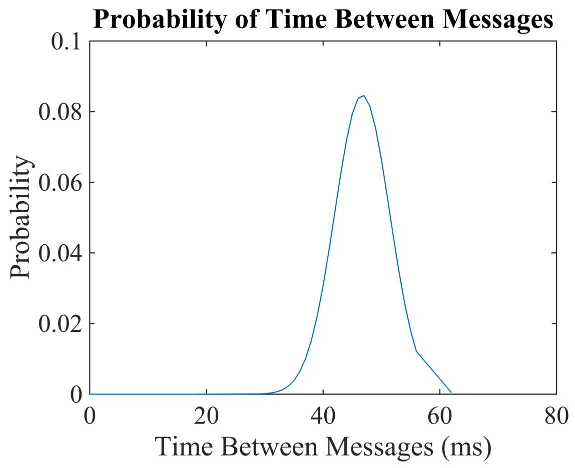

Figure 6: Probability Density Function of Message Delay – ECUs can easily be able to identify nodes spamming messages

against a resourceful attacker, but not a patient one.

9 Discussion

While Mini-MAC succeeds in securing the CAN bus against replay and masquerade attacks from time-limited attackers, it is possible to improve automotive security further through more involved protocol and network architecture changes. These changes are more difficult and more costly to implement but offer better protection.

Section 9.1 describes a proposed change to the network stack used in the CAN bus that will allow Mini-MAC to use more bits in the CAN frame. Section 9.2 describes a new network architecture that is inherently more secure than CAN is now, and Section 9.3 describes some open problems in the field of automotive security.

9.1 Adding Bits to Mini-MAC

By enlarging the Mini-MAC, it is possible to drastically increase CAN security. Although 27.3 minutes average time-to-defeat for a brute force attack may be sufficient for the context of a moving automobile, it is always worth exploring raising that number. Against a brute force attack the solution is simply more bits in the MAC, which as discussed, is not possible without adding bus overhead at the application level.

Table 5: Time-to-Defeat for Various MAC Lengths

MAC Length(b)	Time-to-Defeat
16	6.4s
24	1.7m
32	27.3m
40	7.28h
48	4.85d

It is possible to add two more bytes to the transmitted MAC by replacing the CRC field (2 B, Figure 1) with MAC bits. The MAC will provide the feature of checking for message error (so long as the underlying hash is collision-free) in addition to authentication.

The difficulty here is in rewriting lower-level code in the network stack to either perform the entire MAC calculation or to open the CRC field up to the application level which calculates the MAC. The defense time gains from the additional two bytes can be extremely significant. Table 5 shows how the time-to-defeat changes for different lengths of the MAC. Any MAC smaller than 4B is probably useless, but in the case of a 5B or 6B MAC, the brute force attacker will very likely run out of time. The 6B MAC, which accounts for the original 4B plus the 2B from the CRC field, is capable of holding out against even an attacker with an extended time budget (for example, if you leave your car on the street and leave for the weekend).

9.2 Design Alternatives

The capabilities of the CAN bus and the ECUs on it are the limiting factors in how secure it can be made. There are two major design changes to automotive computer networks that would significantly increase security: 1) Replace CAN with high-speed, well defined network stack elements such as 802.3 Ethernet and IP 2) Segregate nodes on the CAN bus into task-defined groups.

As info-tainment becomes a major task of the modern automotive computer network, these networks must be able to support not only control information but high rate audio and video content. The Ethernet/IP stack found in many home networks is a natural fit for this task. Additionally, these networks can utilize IPsec, which can handle data encryption and authentication.

That a car radio could potentially send a message to a brake control unit is a tremendous oversight in design. Although some vehicles do a better job of segregating vehicle control units from entertainment or environment control units than others, there is no standardization and these design decisions seem to stem more from spatial arrangment or bus availablilty than they do for any security-conscious reason. The solution should be to place high responsibility nodes (ECUs that control, for example, brakes and acceleration) on physically separate networks from low-responsibility nodes (the radio, blinkers, etc) and nodes designed to communicate to the Car-to-X network (navigation systems, in-car Internet).

9.3 Open Problems

The most relevant question to this work is the effectiveness of hash functions over small inputs. As previously noted, the messages on the CAN bus are 8 B, and before any practical use of a solution like Mini-MAC could be made, this question must be answered to a satisfying degree. Many hash implementations require padding of inputs to fit fixed size inputs, but what ratio of padding to message is acceptable before the hash loses effectiveness? These are not questions unique to Mini-MAC, but it is a protocol which could be severely compromised from a hash which is weak under heavy padding as the message is so small compared to the input block size.

10 Conclusion

We propose Mini-MAC, which is the first variable-length MAC protocol for the CAN bus that adds no bus traffic overhead. Using 4 B for a MAC is too small for many applications – the core idea of Mini-MAC is not to make the CAN bus impenetrable, but instead to raise the bar for security in a resource-conscious way.

Automotive computer systems are extremely limited in several ways. The computational power in nodes is limited and incapable of complex cryptography, and the bus is too slow to allow for overhead related to security protocols. Additionally, many of the messages are extremely time sensitive and must not be delayed or risk the safety of the driver or others on the road.

Given this context, we believe Mini-MAC is reaching the limits of what is possible for securing the CAN bus, but note that even this system is still vulnerable to attacks from determined attackers or those wishing to deny service.

Note: I don't like the last sentence very much. Still thinking on it

Response to Ed's note - the fault tolerance procedure I mentioned previously is directly from spacecraft. I've been thinking about how to re-sync on a fault and that is currently where I'm leaning. I'm not very familiar with any specific authentication mechanisms they use though.

11 Acknowledgments

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

[KCR+10] K. Koscher, A. Czeskis, F. Roesner, S. Patel, T. Kohno, S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, and S. Savage. Experimental security analysis of a modern automobile. In *Proceedings of the 2010 IEEE Symposium on Security and Privacy (SP)*, pages 447–462, May 2010.

[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 RFC1321: 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. *Intelligent Transportation Systems, IEEE Transactions on*, 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*. December 27, 2015.