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

Redacted for anonymous submission

February 6, 2016

Abstract

We propose Mini-MAC, a new message authentication protocol that works in existing automotive computer networks without delaying any message or increasing network traffic.

Deployed in many vehicles, the CAN bus is a low-speed network connecting electronic control units, including those that control critical functionality such as braking and acceleration. The CAN bus is extremely vulnerable to malicious actors with bus access, including wireless 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 or increase bus traffic, severely limit how well this network can be protected.

Mini-MAC is based on a counter-seeded keyed-Hash MAC (HMAC), augmented with message history and truncated to fit available message space. It does not increase 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. The message history feature protects against all transient attackers, even if they know the keys. 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 [MV15] gained full control of a new Jeep, including its engine, brakes, and steering, by exploiting vulnerabilities in its computer network and Wi-Fi implementation and by rewriting firmware on a controller connected to the car's entertainment system. This demonstration, and other similar projects [RMM+10, KCR+10, CMK+11, WJL15, FBZ+08], 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 payload sizes without delaying messages. Built 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; the message history feature protects against all transient attackers (attackers who can access the CAN bus only for a limited period of time). To avoid sending separate messages to different recipients, Mini-MAC applies authentication keys shared among groups of communicating Electronic Control 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 digital 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 prototype implementation of Mini-MAC and preliminary timing studies of Mini-MAC for three component hash functions (MD5, SHA-1, SHA-2). In comparison with a straightforward HMAC, our Mini-MAC implementations show on average a mean code size increase of 5.99% and a mean execution time increase of only 2.58%. For fastest speeds, we recommend using mini-MAC-MD5, for which our implementation running on an 8 MHz processor takes on average 7.5 milliseconds (ms) to compute a tag.

Our contributions include:

- Mini-MAC, an authentication protocol suitable for many 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.

- Mini-MAC's use of message history protects against all transient attackers, even if they know the keys.

- Experimental demonstration of Mini-Mac, including execution times of Mini-MAC with each our three HMAC implementations using the MD5, SHA-1, and SHA-2 hash functions. On average our implementation of Mini-MAC-MD5 takes 7.5 ms to compute a tag.

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 [Sti06] and NIST [Nat15].

2.1 ECUs

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 Controller Area Network (CAN) bus is a simple, low-speed bus designed to network simple nodes. It typically runs at 500 kbps in automobiles [fS15].[1]

Transmissions are organized in frames. As shown in Figure 1, a frame contains an 11-bit identifier field and a data payload, as well as some control bits. Figure 1 shows the allocated data payload space as 8 bits, but it can be 8 to 64 bits and (based on our observations) is typically 64 bits. The payload of up to 8 bytes is the most important element, as any MAC tag must fit into this frame or use a more complex multi-frame data transmission protocol that may or

[1]kilobits per second (kbps).

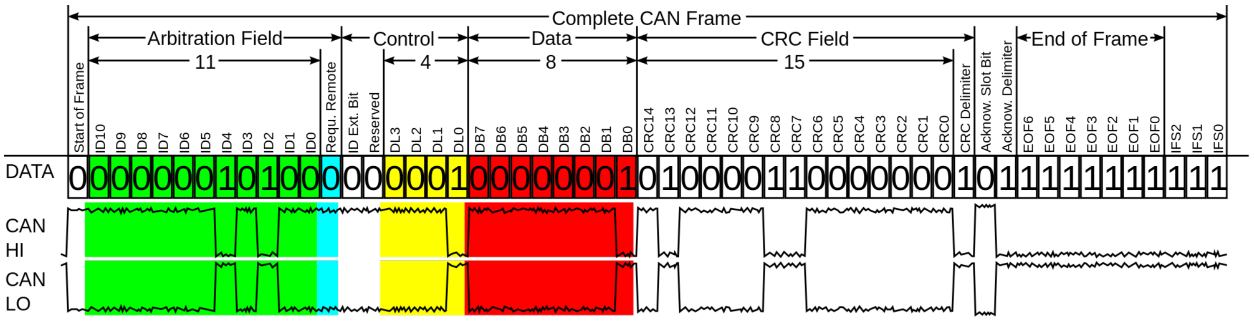

Figure 1: The CAN frame [Ern13]. Each transmission on the CAN bus is a structured sequence of 55–111 bits including 8–64 data bits.

may not be supported on all ECUs.

We use the term "message" to refer to the data payload of one logical transmission. Any payload longer than 64 bits must be sent as a multi-frame transmission.

To learn more about the characteristics of messages transmitted on the CAN bus, the authors captured 15,768 messages from a 2010 Toyota Prius during a 12.27-minute test drive around a parking lot on our university campus. The drive featured turns, gear changes, reverse driving, and switching on and off various instruments and tools including headlights, blinkers, and locks.

Ideally, and in the case of Mini-MAC, the MAC tag can fit into the payload together with the data, thus not increasing bus traffic. In our test data, approximately 61% of messages contain at most four data bytes (20% contain 4 bytes; 16% contain 3 bytes; 17% contain 2 bytes; and 8% contain 1 byte). Approximately 35% of messages contain a full 8 data bytes, and 4% contain 7 bytes.

Thus, for most message, there are at least four bytes of space available for a MAC tag. Inserting a tag in the unused allocated payload space does not increase the number of messages sent, delay any message, or increase the number of bits transmitted.

Messages have fixed, known sizes. The longest messages we witnessed seemed to be status reports describing the steering wheel angle and tire speed, with no apparent observable effect on any ECU. For these long messages with 8-byte payloads, Mini-MAC will not add a tag.

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 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, Checkoway et al. [CMK$^+$11] gained bus access by packing malware into a WMA audio file played on the car stereo. Rouf et al. [RMM$^+$10] demonstrated information leakage and message spoofing on RFID-based tire pressure monitoring systems.

Despite many demonstrated security flaws, automotive manufacturers have been unreasonably hesitant to acknowledge the vulnerabilities inherent to the CAN bus, sometimes wishfully claiming that undetected bus access is difficult. In 2013, Toyota [Kle13] offered the following position on automotive security: "Toyota has developed very strict and effective firewall technology against such remote and wireless services." "We believe our systems are robust and secure." "The presence of a laptop or other device connected to the OBD [on board diagnostics] II port would be apparent." These statements demonstrate a failure to acknowledge the reality of wireless vulnerability of modern vehicles demonstrated in several notable publications.

More recently, insurance companies have begun issuing OBD-II devices that track and report driving statistics via cellular modems, as part of a program to offer more competitive rates to customers. The Snapshot tool from Progressive[2] and the Drivewise tool from Allstate[3] are two examples of these dongles. Foster et al. [FPKS15] gained remote access to the CAN bus by hacking these devices.

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, who also knows the key, verifies the tag by recomputing it.

The Keyed-Hash Message Authentication Code (HMAC) [BCK96, Nat08, KBC97, Dan12] 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

[2]www.progressive.com/auto/snapshot/
[3]www.allstate.com/drive-wise.aspx

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

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. The symbols $\oplus$ and $\parallel$ represent exclusive-or (XOR) and concatenation, respectively.

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.

4 Previous Work

Previous proposals to add authentication to the CAN bus violate the engineering constraints described in Section 3, increasing bus traffic 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 per message, requiring 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 and Mayer [ZM14] propose a fixed-size, time-stamped MAC based on pairwise key distribution. Their tag overflows the CAN frame, increasing bus traffic, 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 pairwise key distribution. They unrealistically assume that the messages and MAC tag are short enough to fit into one frame. Also, by queuing messages into batches, their system delays messages.

Many automakers and parts manufacturers are now members of the Open Alliance,[4] 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 to any member of any group to which he holds the key.

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 accelerate the car. Goals of the attacker include spoofing or replaying messages on the CAN bus that will be accepted by an ECU as valid.

[4]http://www.opensig.org

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 second 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, yet a DOS attack on the CAN bus while the vehicle is operating could be dangerous (e.g., losing control on a high-speed turn). Guarding against DOS attacks is difficult and would require fundamental changes to the CAN bus.[5]

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.

[5]Another potential DOS attack, which more generally attacks all electronics in a vehicle, is to radiate the vehicle with an electro-magnetic pulse. Doing so might be useful to law enforcement in extreme circumstances to stop certain dangerous high-speed criminals, if the pulse could be directed in a controlled, focused, and well-targeted manner.

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. We also describe an enhanced message history option and comment on key distribution.

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 hypothetical precomputation attacks (even though we know of none). 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 tag to fit available space in the CAN frame (typically the resulting truncated 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.

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

Each ECU saves recent messages sent on the bus and concatenates 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 a message into any message history without successfully forging a message.

There are three reasons for the message history architectural element: First, it potentially 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 resynchronized (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 hypothetical precomputation attacks. Whereas counter values are predicable, message history is not.

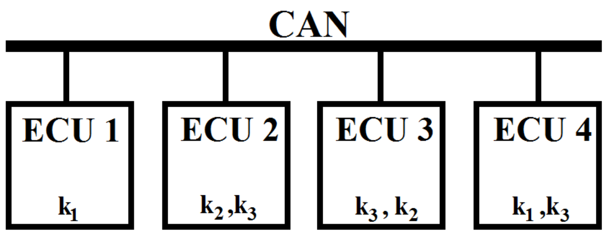

Figure 2: Mini-MAC key distribution. Each group of ECUs shares an authentication key.

6.2 Design

Figure 3 shows how to compute Mini-MAC($k, M_n, s, C,$ History), where k is the 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 = $(M_{n-\lambda}, \ldots, M_{n-1})$

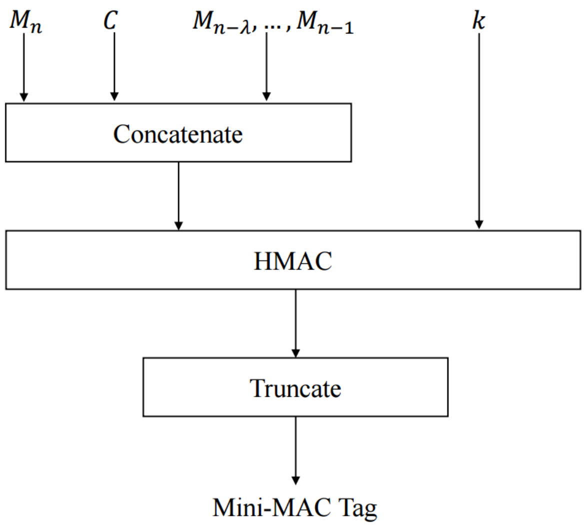

Figure 3: Mini-MAC. The Mini-MAC tag of a message M_n is a truncated keyed HMAC of the concatenation of M_n, a counter C, and the most recent λ messages.

is the sequence of the most recent valid λ messages for key k.

The Mini-MAC tag is computed as

$$\mathrm{trunc}(s, \mathrm{HMAC}(k, \mathrm{Input})), \tag{2}$$

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

$$\mathrm{Input} = M_n \parallel C \parallel (M_{n-\lambda} \parallel \cdots \parallel M_{n-1}). \tag{3}$$

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 $\mathrm{trunc}(s, \cdot)$ function extracts the s most significant bits of its input, which is the way recommended by NIST for truncating a hash tag [Dan12].

We recommend a 64-bit counter, which (assuming at most 40 message per second) ensures no repeated counter state for 20 years of continuous operation (32 bits would prevent counter roll-over for 20 years with four hours of driving a day).

In our implementation, we artibrarily used $\lambda = 5$, but we recommend a higher value (say, $\lambda = 16$) to reduce the chance of a repeated state. See Section 6.4 for an enhanced message history feature.

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.

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. Throughout, we shall refer to SHA-2-256 as SHA-2.

6.4 Enhanced Message History

An enhancement of the message history feature increases security following resynchronization (see Section 9.1) by ensuring that much older messages are incorporated.

By creating a message history sampling from a long enough (e.g., 30 min) stream of messages, no transient attacker will have enough time to observe the entire sample. Powerfully, this strategy defeats all transient attackers, even if they know the group keys. For example, this strategy defeats an attacker who learns the group keys by compromising the key distribution process or by physically probing the ECUs, but who cannot observe bus traffic for a long time.

There are many possible choices for realizing this strategy. For example, one might generate a cryptographic hash chain of all messages ever sent. Alternatively, for greater speed, one might use a faster non-cryptographic combining function (e.g., CRC). Another option would be to compute a hash chain from a limited sample of the stream.

Given the limited power of the ECUs, we recommend that the message history be defined in two parts from the recent stream of valid messages received under the given group key: one sparse, one consecutive. The first part comprises a spare sample of $\lambda/2$ messages selected according to a prescribed pattern in a first-in, first-out queue, over a specified section of the stream. For example, one might sample every 10,000th message. The section length should be chosen to exceed the time available to a transient attacker. For example, at most 72,000 messages can be sent in 30 minutes, even if messages are continuously sent using only one key.

The second part comprises the most recent $\lambda/2$ messages from the stream of valid messages received under the given group key. This consecutive sample helps ensure changed state after a resynchronization.

6.5 Key Establishment

We recommend that group keys be established as follows. Each ECU should generate its own key material, which should never leave the ECU, except possibly in encrypted form. Neither the ECU manufacturer, car manufacturer, nor car dealer should ever learn any keying material. The security manager (perhaps a dealer) should be capable of initiating a process for key establishment or rekeying.

To establish a group key, each group member should generate a key share of the same length as the group key. Using a key-encrypting-key (KEK) shared by all ECUs in the group and used only for this purpose, each member sends its share to the other group members. Each member computes the group key as the XOR of the shares. In the absense of public-key cryptography, we recommend that this KEK be loaded into the ECUs during car manufacturing, using a trusted process that prevents anyone from learning the KEK.

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 implementations, 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 traffic (number of messages and bits sent), 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 known at compile time because there is no dynamic memory allocation.

Running the MSP430 at 8 MHz, we measured execution time using one of its counter registers. The 32 kHz external clock on the test board provided timing values to approximate millisecond (ms) 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

Table 1: Additional bus traffic for various authentication mechanisms, for one key group with n recipients. Mini-MAC adds no additional bus traffic

Algorithm	**Additional Traffic (bits)**
HMAC-MD5 (Group)	128
HMAC-MD5 (Pairwise)	$128n$
Lin-MAC	$128n$
Mini-MAC	0

Table 2: Additional code size and mean additional time to compute Mini-MAC implementations over HMAC. For each implementation, the additional RAM usage is 5 bytes plus message history.

Hash	**Code Size (bytes)**	**Execution Time (ms)**
MD5	835	0.38
SHA-1	850	0.42
SHA-2	766	0.68

based on an Arduino Uno platform and connected via an OBD-II CAN transceiver shield (see Section 2).

8 Results and Analysis

We analyze Mini-MAC for performance and authentication strength. Mini-MAC must be able to au-

Table 3: Execution time (observed mean ± standard deviation) of Mini-MAC implementations. Only the MD5 implementation meets our engineering requirement of at most 25 ms per message.

Hash	**Time (ms)**		
	μ	$\pm$	σ
MD5	7.5	$\pm$	0.07
SHA-1	28.0	$\pm$	0.06
SHA-2	69.6	$\pm$	0.08

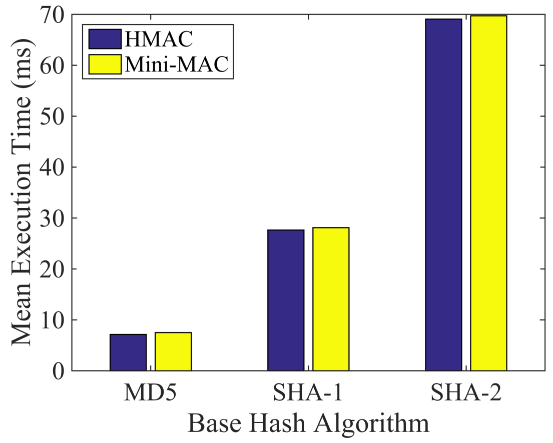

Figure 4: Mean execution time for our Mini-MAC implementations running on an 8 MHz processor, from 1000 messages. The observed standard deviations were small.

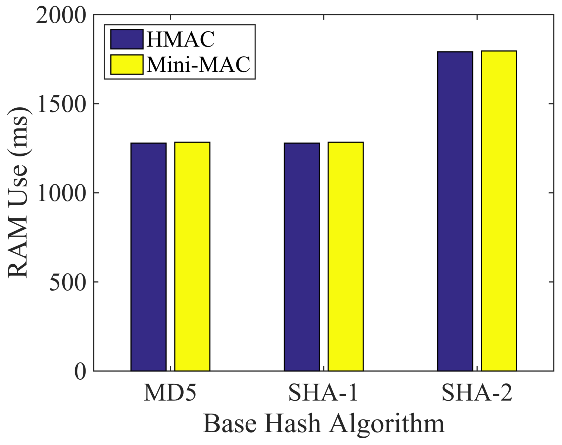

Figure 5: RAM usage for our Mini-MAC implementations, from 1000 messages with $\lambda = 5$.

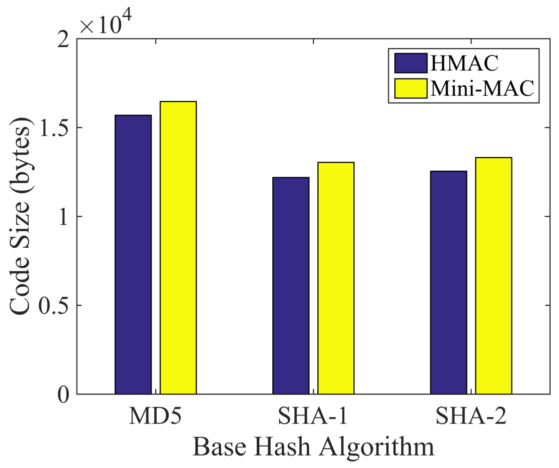

Figure 6: Code size of our Mini-MAC implementations.

thenticate messages without compromising the performance of the real-time, safety-critical systems.

8.1 Performance

Tables 1–3 and Figures 4–6 show the message traffic, execution time, code size, and RAM usage of our unoptimized implementations of Mini-MAC and their underlying HMACs. Time and space are dominated by the HMAC computation.

Table 1 shows that Mini-MAC adds no additional bus traffic. By contrast, HMAC-MD5 adds 128 bits (two CAN frames) per authentication, due to the 128-bit tag. Similarly, the pairwise-keyed Lin-MAC generates two additional CAN frames per recipient ECU in each group communication.

Table 3 shows the mean observed execution time and standard deviation for each of our implementations of Mini-MAC. Using MD5 is much faster than using SHA-1 or SHA-2. As shown in Table 2, the overhead in time to compute Mini-MAC beyond HMAC is very little (approximately 0.38–0.68 ms). Similarly, Figure 4 shows the mean observed execution times for Mini-MAC running on our 8 MHz processor.

Importantly, Table 3 shows that only our MD5 implementation of Mini-MAC runs fast enough to satisfy our requirement of authenticating at least 40 messages per second (approximately 25 ms between messages). While highly optimized code will likely run faster, on the basis of our timing measurements, we recommend implementing Mini-MAC using MD5.

Figure 5 shows the RAM usage for our implementations of Mini-MAC. As shown in Table 2, the overhead in RAM usage to compute Mini-MAC beyond HMAC is very low (5 bytes).

Figure 6 shows the code size of our Mini-MAC implementations. As shown in Table 2, the additional code size for Mini-MAC beyond HMAC is very small (approximately 800 bytes).

8.2 Authentication Strength

Mini-MAC detects spoofed messages by using a keyed HMAC. It detects replay attacks by incorporating a

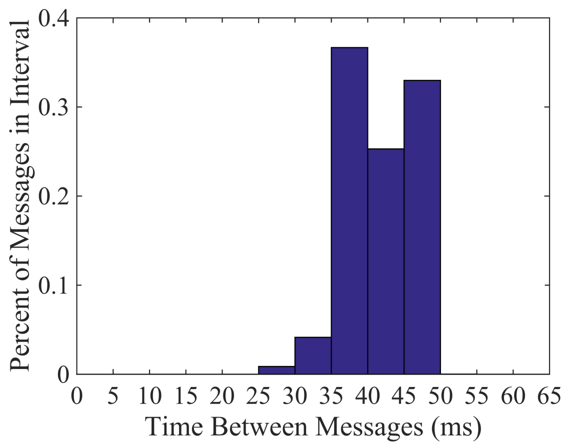

Figure 7: Histogram of message interarrival times observed by the authors from 15,768 messages on a 2010 Toyota Prius.

counter and recent message history into the HMAC input. Using message history also complicates a transient attacker who may be unable to observe a sufficient number of recent messages.

The 128-bit keys are sufficiently long to withstand exhaustive key-search attacks. The 64-bit counter is sufficient to prevent counter rollover within the lifetime of the car.

With the enhanced message history feature, the adversary must learn a sample of messages taken from a specified period of recent message history. This period can be set (e.g., 30 mins) to exceed the amount of time available for any transient attacker.

A security-enhancing feature of the CAN bus is, ironically, its slow speed of at most approximately 40 messages per second. Figure 7 shows a histogram of message arrival times that we collected from a 2010 Toyota Prius. The slow speed of the CAN bus limits the rate at which an attacker can inject messages into the bus.

Another defensive feature is that there is no simple fast way for an adversary to test if a candidate tag is valid. The only way we are aware of is to inject a message into the bus and observe if the receiving ECU accepted the message.

Limitations imposed by Mini-MAC include its use of group keys and truncated HMAC tags. Using group keys means that a single compromised ECU learns all of the keys to which groups it belongs. We consider this limitation an acceptable design tradeoff given that Mini-MAC does not increase bus traffic and the compromise of any critical ECU is a devastating security failure.

Mini-MAC does not authenticate messages with long 8-byte payloads. This limitation is an intentional tradeoff to ensure that Mini-MAC does not delay any message. The only 8-byte messages that we observed seemed to be status reports with no apparent effect on any ECU. It may be possible to reduce the length of such status data slightly by deleting some of the low-order data bits.

Because all message types have fixed, known message lengths, the adversary cannot force arbitrary messages to have artificially long 8-byte payloads.

One potential attack is for the adversary to inject mes-

Table 4: Probability and expected time for a spoofed message to be accepted by an ECU for various tag lengths for two attack scenarios. In random guessing, the adversary guesses a tag. In the hypothetical collision search, the adversary seeks a Mini-MAC collision. Probability denotes the probability of success for one trial. Expected time denotes the expected time for the cumulative success probability to be 0.5. All calculations assume the adversary injects 40 forged messages per second.

Tag Length (bits)	**Random Guessing**			**Collision Search**	
	Probability	**Time**		**Time**	
8	0.0039	3.20	sec	0.40	sec
16	1.53E-05	13.65	min	6.40	sec
24	5.96E-08	58.25	hour	1.70	min
32	2.33E-10	621.38	day	27.30	min
40	9.09E-13	435.82	year	7.28	hour
48	3.55E-15	111,568.9	year	4.85	day

sages into the bus with the hopes of trying a tag that verifies correctly. We shall call this attack the "trial injection attack." A downside of this attack is that it would be easy to detect such an attack involving many messages.

Table 4 lists for various tag lengths L two reference times that are useful in assessing the effectiveness of this and other hypothetical attacks. The "random guessing" columns refer to the strategy of randomly guessing tags. Probability denotes the probability that one guess will be accepted as valid; this probability is $1/2^L$, where L is the tag length. Expected time denotes the expected time for the cumulative success probability to be 0.5, assuming the adversary is limited to trying 40 tags per second. This value is the expected time for a straight-forward implementation of a trial injection attack to succeed.

For example, with a four-byte tag, the expected time for a straight-forward message injection attack to succeed is over 621 days.

The "collision search" column refers to an optimistic hypothetical attack whose success is based on finding a Mini-MAC collision,[6] exploiting the Birthday Paradox. We do not see how an adversary could mount such an attack; we include this column simply as a conservative reference.

Another potential attack is to observe bus traffic with the hopes of finding statistical regularities that would significantly improve the chances of success of the aforementioned attack. We conjecture that this attempt is unlikely to yield significant advantage, given the strong properties of HMAC and some desirable (albeit imperfect) characteristics of the component hash functions.

Given that the CAN bus does not authenticate messages, we conclude that Mini-MAC meaningfully raises the bar of vehicular security.

[6]A collision is any pair of different inputs that produces the same output.

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 synchronized 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).

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, by using counters and message history to authenticate messages, mini-MAC increases the opportunities for possible synchronization failures, and this opportunity increases with larger values of λ.

We do not know the typical frequency of synchronization failures, and we did not observe any.

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 does 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 valid Mini-MAC tag by exhaustive search. Table 4 gives the expected time for an adversary to send a forged message that will be accepted by another ECU, for various tag lengths. For example, increasing the tag from 8 to 32 bits increases this time from approximately 3.2 seconds to over 621 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

It would be interesting to implement and test Mini-MAC in an actual vehicle.

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 [WWP06].

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) Segregate nodes on the bus into task-defined groups. (3) Protect access to the bus by physically separating critical and non-critical systems. In particular, it should not be physically possible for malware or faults in entertainment or Bluetooth systems to affect braking, steering, or acceleration. (4) Physically protect the ECUs.

Additional ideas, suggested by REDACTED include: (5) Physically secure the bus to prevent any access, except through a read-only diagnostic port. (6) At pseudorandom time intervals, pseudorandomly change the ECU ID numbers, based on a key shared by all ECUs used only for this purpose.

A separate related problem is to detect vehicular network intrusions [OIOS14]. A challenge of such work is that there is no good response 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 message authentication protocol for the CAN bus that neither increases the number of messages sent nor delays any message, allowing it to be used in vehicular systems with time-sensitive messages. The truncated keyed HMAC protects against message injection by adversaries who do not know the ECU keys. The counter and message history protect against replay attacks. Message history also protects against all transient attackers, even if they know the authentication keys.

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.

Objects and systems in the emerging Internet of Things and Car-to-X network will face similar authentication challenges to those faced by vehicular ECUs. We hope, for example, that the designers of toaster ovens will not separately create ad-hoc security mechanisms for networked communications with the refrigerator, home electrical system, home owner, and toaster manufacturer. Instead, networked objects of the future need to be built with control units that can execute resilient general-purpose standard cryptographic functions and protocols suitable for the Internet of Things. We hope that some of the ideas from Mini-MAC will be helpful toward that objective.

11 Acknowledgments

Redacted.

References

[BCK96] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. Keying hash functions for message authentication. In Neal Koblitz, editor, *Proceedings of the 16th Annual International Cryptology Conference on Advances in Cryptology*, volume 1109 of *LNCS*, 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, and Tadayoshi Kohno. Comprehensive experimental analyses of automotive attack surfaces. In *USENIX Security Symposium*. San Francisco, 2011.

[Con06a] Brad Conte. Implementation of SHA-1 in C, 2006. http://bradconte.com/sha1_c.

[Con06b] Brad Conte. Implementation of SHA-256 in C, 2006. http://bradconte.com/sha256_c.

[Dan12] Dang, Quynh. *Recommendation for Applications Using Approved Hash Algorithms:*. NIST Special Publication 800-107, Revision 1. National Institute of Standards and Technology, August 2012.

[Ern13] Erniotti. CAN bus frame in base format without stuffbits and electrical bus levels, 2013. https://commons.wikimedia.org/wiki/File:CAN-Bus-frame_in_base_format_without_stuffbits.png.

[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.

[FPKS15] Ian Foster, Andrew Prudhomme, Karl Koscher, and Stefan Savage. Fast and vulnerable: A story of telematic failures. In *9th USENIX Workshop on Offensive Technologies (WOOT 15)*, Washington, D.C., August 2015. USENIX Association.

[fS15] International Organization for Standards. Iso 11989. Technical report, International Organization for Standards, 2015.

[KBC97] Hugo Krawczyk, Mihir Bellare, and Ran Canetti. HMAC: Keyed-hashing for message authentication. RFC 2104, RFC Editor, February 1997. `http://www.rfc-editor.org/rfc/rfc2104.txt`.

[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.

[Kle13] Zoe Kleinman. Car hackers use laptop to control standard car, 2013. http://www.bbc.co.uk/news/technology-23443215.

[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.

[MV15] Charlie Miller and Chris Valasek. Remote exploitation of an unaltered passenger vehicle. *Black Hat USA*, 2015.

[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.

[OIOS14] S. Otsuka, T. Ishigooka, Y. Oishi, and K. Sasazawa. CAN security: Cost-effective intrusion detection for real-time control systems. 2014.

[Pes09] Alexander Peslyak. A portable, fast, and free implementation of the MD5 message-digest algorithm (RFC 1321), 2009. http://openwall.info/wiki/people/solar/software/public-domain-source-code/md5.

[Riv92] Ronald L. Rivest. The MD5 message-digest algorithm. RFC 1321, RFC Editor, April 1992. `http://www.rfc-editor.org/rfc/rfc1321.txt`.

[RMM+10] Ishtiaq Rouf, Rob Miller, Hossen Mustafa, Travis Taylor, Sangho Oh, Wenyuan Xu, Marco Gruteser, Wade Trappe, and Ivan Seskar. Security and privacy vulnerabilities of in-car wireless networks: A tire pressure monitoring system case study. In *Proceedings of the 19th USENIX Conference on Security*, USENIX Security'10, pages 21–21, Berkeley, CA, USA, 2010. USENIX Association.

[Sti06] Stinson, Douglas R. *Cryptography: Theory and Practice*. Chapman & Hall/CRC, Boca Raton, FL, third edition, 2006.

[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.

[WWP06] Marko Wolf, André Weimerskirch, and Christof Paar. *Embedded Security in Cars: Securing Current and Future Automotive IT Applications*, chapter Secure In-Vehicle Communication, pages 95–109. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.

[WY05] Xiaoyun Wang and Hongbo Yu. How to break MD5 and other hash functions. In Ronald Cramer, editor, *Proceedings of the 24th Annual International Conference on Theory and Applications of Cryptographic Techniques*, volume 3494 of *LNCS*, pages 19–35, Berlin, Heidelberg, 2005. Springer.

[WYY05] Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu. Finding collisions in the full SHA-1. In Victor Shoup, editor, *Advances in Cryptology CRYPTO 2005*, volume 3621 of *LNCS*, 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.

Submitted to *Cryptologia*. February 6, 2016.