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
{schmandt, sherman, nilanb}@umbc.edu
November 21, 2015

Abstract

We propose Mini-MAC, a new message authentication protocol that is designed to work 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) are used to verify the identity of the sender of a message, and variants are used to prevent message replay attacks; however, standard or time-seeded MACs are unsuitable for use on the CAN bus because of data payload size restrictions. This work presents a smaller footprint alternative to the traditional MAC which aims to raise the bar of automotive network security. The key elements in Mini-MAC are 1) It causes no increase in bus traffic 2) It incurs a very small performance penalty relative to HMAC. It is the first work to combine these tenets for automotive computer networks. Mini-MAC is based on a time-seeded HMAC, but is augmented with message history and then pared down to fit available message space.

Index Terms

CAN bus, automotive security, message authentication, vehicle security, applied cryptography.

Note: Are units like B(bytes), b(bits), and ms(milliseconds) normally explicitly defined? I thought these were ubiquitous enough to not need that.

Figure and Table numbers may be off – I moved a few things around, but I think I caught where I had to change a reference

1. Introduction

Note to ed: I haven't re-done the intro yet. I want to get the rest of the paper in good shape before working on it.

The Controller Area Network (CAN) bus is typically used in automotive networks to connect various ECUs in the vehicle. These ECUs can control anything from electronic door locks to the brakes or engine. With the advent of "drive-by-wire" systems in vehicles that offload tasks that previously required physical driver interaction (for example, ignition, steering, or braking) to semi-autonomous systems, the ECUs in a vehicle are tasked with actually driving the vehicle more and more.

This new paradigm in automotive control presents an obvious target to malicious actors who wish to gain control over a car. Such access poses a direct threat to the safety of the people in the car and other drivers on the road. The CAN bus provides no security layer and is not suitable for use with traditional security measures [5] [2]. To this end, this project presents a security solution designed for message authentication in a constrained environment such the CAN bus, which is the first solution designed to authenticate bus messages without adding any message traffic overhead.

The threat to the network stems from the broadcast nature of the CAN bus – any message put on the bus can be sent from any ECU, and only a non-verified ID field is used to discriminate between senders. The lack of authentication allows malicious actors to cause invalid messages to be accepted by ECUs simply by spoofing their ID or by replaying messages seen earlier.

A traditional solution to this vulnerability is message authentication which requires the use of a secret key to create a verification value the recipient can use to verify the sender did indeed generate the message. Simple MACs do not prevent replay attacks as a malicious actor can simply resend a message seen before if they wish to recreate the behavior (for example, the use of brakes). Time-seeded MACs use some time-based token to prevent replay attacks by issuing a new MAC for the same message at a different time.

Traditional cryptography is also unsuitable for use in automotive networks for several reasons. Primarily, the computing resources required for full encrypt/decrypt operations are unrealistic for the type of embedded computer found in automobiles. Additionally, the messages generated by traditional symmetric ciphers (asymmetric cryptography is not considered, being much more computationally instensive than symmetric cryptography) such as AES will not fit into a single CAN frame and would therefore significantly increase bus traffic.

Another concern with the CAN bus is that many of the messages concern the real-time operation of ECUs which control critical car components such as the brakes or engine. Not only must a security measure be suitable for execution on computationally weak devices, but it must not incur a bus utilization penalty for fear of violating any real-time constraints which could have dire safety implications for the driver or other people on the road.

The attacks concerned are more specifically called replay and masquerade attacks. In a replay attack, an illegitimate node propagates a previously observed message on the bus in hopes of re-creating a desired response, also previously observed. In a masquerade attack, the illegitimate node propagates either a new or previously observed message with the intent of appearing to be a legitimate sender.

Lin-MAC suggests a MAC computed with a counter, which is incremented on every message. This solves the replay attack problem, but it does not address some of the other issues with a MAC, namely that 1) the sender must generate a new MAC for each recipient and B) the sending of a MAC along with a message takes far too much bus time to be useful in a time-constrained system [6].

To address these concerns, this project presents Mini-MAC, which is designed not to increase bus utilization at all. Mini-MAC adaptively sizes the MAC result depending on which message is being sent. It also operates on the premise of grouped keys, rather than a pairwise key system. It also incorporates message history into the construction of the MAC which adds a layer of confusion to the resulting authentication value. It is the first protocol to authenticate automotive network messages without adding bus utilization overhead.

Specifically, this project presents several contributions:

- Mini-MAC, a small-size authentication protocol suitable for systems like the CAN bus
- A partial redesign of the CAN protcol which would enable stronger authentication Note to ed.: This isn't necessary for Mini-MAC to work but it would make it stronger - I don't want the reader to think that the system won't work without modifying the underlying CAN protocol
- A discussion on context-dependent security for automotives Note to ed.: what I want to mention is the idea that security is not a binary option, and that for a context like this, it may be suitable to have a system that will prevent attacks for a short time period. I'm not sure the best way to summarize that.

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.

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.

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 things 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,

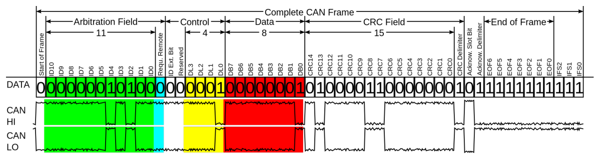

Figure 1. CAN Frame

it typically runs at 500 kbps. A message contains an 11-bit identifier field and an 8-byte data payload as well as some control bits. Figure 1 illustrates a CAN packet. 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 show that a large amount of messages (61%) use no more than 4-byte of data in the payload.

2.3. Bus Access

Despite many demonstrated examples, automotive manufacturers are hesitant to acknowledge the vulnerabilities inherent to the CAN bus because of the difficulty in gaining access to it. 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. A driver might notice some kind of hardware module sticking out from underneath the wheel. It has been shown that it is possible, however, 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. Researchers have even demonstrated gaining bus access by packing code into a WMA audio file played on the car stereo [2].

I saw your note on this subsection, but I think it's important to keep a mention of the work done in remote access here. I don't want to discuss that work on its own as it isn't related except that it helps show there are multiple paths to getting access to the CAN bus. What do you think?

2.4. Message Authentication Codes

Message Authentication Codes (MACs) are bit strings used to verify the identity of the sender in order 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) is by the far the most famous and most popular MAC construction protocol. It is so widely used for 2 key reasons. 1) The security of HMAC is mathematically related to the security of the underlying hash function, which makes it relatively easy to trust 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 [1] [7].

The equation for calculating the HMAC is as follows:

$$\text{HMAC} = H((K_0 \oplus \text{opad}) \vee H((K_0 \oplus \text{ipad}) \vee \text{message})$$

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 $\vee$ represent the XOR operation and concatenation respectively. [7]

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.

3. Problem Statement

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

In captured test data for a 2010 Toyota Prius, over 60% of messages contain no more than four data bytes. This leaves 4 bytes of space for a MAC in the most common case.

Note: It's hard to list a requirement for performance or security here. I approached this work more with the idea of "how can I increase security at all, and how will it perform" rather than "here is how it must perform and the security it must have."

4. Previous Work

The most complete work specifically discussing the usage of MACs in the automotive CAN network produced the Lin-MAC construction [6]. 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 also proposes 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.

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.

Note to ed: I recently found a few other papers that I want to mention here but I don't have it written up yet.

5. Adversaries

We consider three classes of adversaries for this problem.

- Type 1: The strongest adversary, a corrupted ECU which has access to a valid key for the MAC it wishes to generate.
- Type 2: A strong adversary, a corrupted ECU without access to valid keys.
- Type 3: A weak adversary, a party with access to the CAN bus but without any valid keys.

Note to ed: I saw you made parenthesis marks around the adversary list on draft 3 - I don't know what this means

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 (that is, 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.

Adversaries are not without some difficulties - there are more ECUs in vehicles every year, and each ECU has a unique (and not publicly known) message ID and data format. These both must be known in order to orchestrate a targeted attack, and 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 this work is designed to defend against are 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. The 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, 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.

As discussed in section 2.3, access to the CAN bus can be obtained via a physical device connected to the CAN bus (typically, but not necessarily, through the OBD-II port), via a wireless link such as cellular or Bluetooth, or even through a corrupted ECU programmed to execute certain commands.

For this work, we assume attackers have all information about the bus – they know the correct commands and ECU IDs to trigger any action possible with the bus. They also have unlimited computing power.

Each ECU on the CAN bus is thought of as trusted, meaning that every message is treated as a legitimate message. This work considers no node trusted, and must verify the authenticity of every message. We do not consider that a legitimate message could be sent from a malcious node (such as a Type 1 attacker).

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

$$\text{HMAC} = H((K_0 \oplus \text{opad}) \vee H((K_0 \oplus \text{ipad}) \vee \text{message})$$

A MAC such as the HMAC will defeat the masquerade attack. Legitimate nodes will share some secret key that is used to seed the HMAC and any node without such a key will not be able to create a valid MAC for a given message. However, a standard HMAC 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 seeding of the HMAC. The resulting MAC 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 in order to guarantee against repeated MACs.

The core principal 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

Note: I don't think I have a good understanding of what you mean by architecture vs design. I thought that the third draft had a good balance of high level description vs how the mini-mac is actually constructed. what's left in this draft is a cut-down version of draft 3 as you noted it was not concise

Mini-MAC uses group-shared keys (Figure 2) instead of pairwise keys. 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.

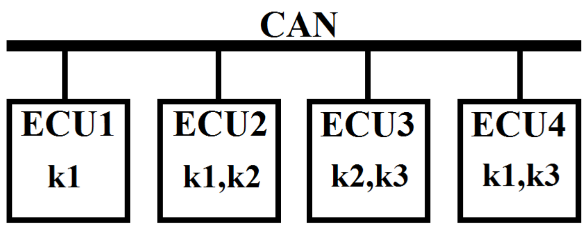

Figure 2. Mini-MAC Key Distribution

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 depicts the key group key distribution model used for Mini-MAC. Note that as multiple ECUs share the same key, messages may be sent to multiple nodes without re-calculating a new MAC for each recipient. This is a key bus overhead reduction compared to Lin-MAC.

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

The calculation of Mini-MAC is only slightly more involved than in the HMAC construction. It is shown below:

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

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

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

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

Note: One comment was that this section needs to be more precise, but I am unsure how to be more precise than the preceeding formulae – are they poorly formed? To me, it is quite clear but I understand if they don't appear that way from a different perspective.

You note that h, l and s are undefined, but they are defined below:

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 pull the Mini-MAC from, 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.

6.3. Implementation

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. HMAC with three common hash functions were used to compute the Mini-MAC for this work. 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.

Note: Adapting the original code for the MSP430 is mostly a matter of type conversion- the MSP430 is a 16 bit device, but most of the code is written for 32 bit systems. I don't want to spend more time discussing the hash implementations than necessary as they aren't really important to the end system – what we do with the HMAC after it's computed is the important part. I added the paragraph above to explain why I used various hashes

HMAC-MD5 The MD5 implementation tested for this project was originally written by Alexander Peslyak [9] and adapted for the MSP430 platform by the authors.

——-

Alexander Peslyak [9] wrote the original MD5 implementation used in this project, and it has been adapted for use on the MSP430 platform by the authors.

MD5 produces a 128-bit output value from a variable length message. Since 2004, the security of MD5 has been severely compromised [12] – 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 [10].

HMAC-SHA1 The HMAC-SHA-1 implementation used for this project was originally written by Brad Conte [3] 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 [11], but it is still used in many applications and will be for the immediate future [8].

HMAC-SHA256 The HMAC-SHA-256 implementation used in this project was originally written by Brad Conte [4] 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 [8].

7. Testing

Testing of the three Mini-MAC implementations noted previously was conducted on the Texas Instruments MSP430F5529 microcontroller. As previously discussed, the speed and power of this device makes it a good test platform for CAN security software.

7.1. Purpose

The purpose of the tests performed here is to evaluate the suitability of HMAC-based message authentication for nodes on the CAN bus. Small memory, RAM and performance overhead is ideal. The three hash functions selected are compared for two reasons 1) That performance and security is a function of the hash function selected as a base 2) Overlaying Mini-MAC onto HMAC incurs a minimal performance penalty regardless of the hash selected.

7.2. Methods

A counter register on the MSP430 generates execution time values. A 32kHz clock increments this counter.

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

Table 1. Mini-MAC Overhead Relative to HMAC

Hash	Exec. Time(ms)
MD5	7.5
SHA1	28
SHA2	69.6

Table 2. Approximate Execution Time of Mini-MAC Construction

/textcolorredNote: I think the sentence from the 3rd draft (below) is less awkward than the above Execution time values are generated by reading a counter on the MSP430 which is incremented by a 32kHz clock.

The approximate millisecond execution time values are extracted from this counter. There may be a +/- 0.03ms inaccuracy in this value depending on the time it takes to read the counter. The results shown are averaged from 1000 runs.

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

The metrics recorded are code size, memory usage, execution speed, and bus utilization.

Table 2 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, while ms represents a value in milliseconds.

Note to ed: You had a comment ”for tables only” on table 2 – what is this referring to?

8. Analysis

The analysis must be conducted from two perspectives - Mini-MAC must be secure, but it must also be able to run on ECUs.

8.1. Performance

Figure 3 shows the execution time comparison for the three HMAC constructions as well as Mini-MAC constructions based on those HMACs. There is very

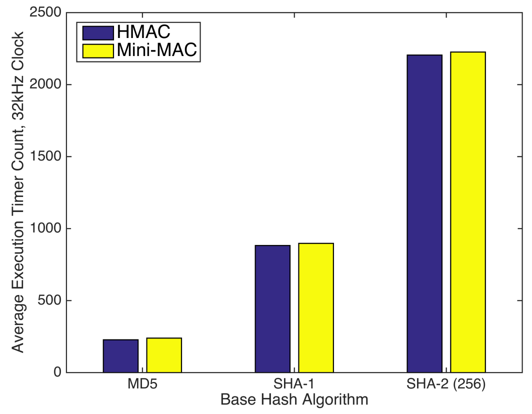

Figure 3. Execution Time Comparison of Mini-MAC Construction

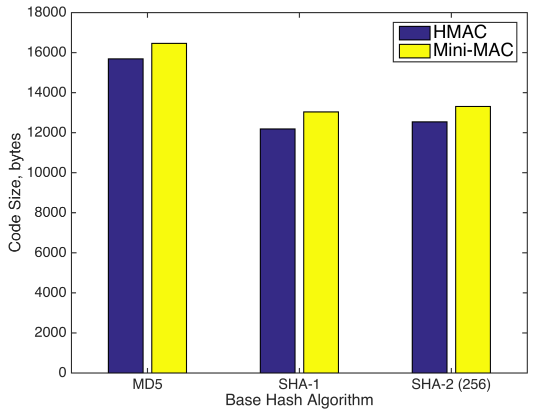

Figure 4. Code Size Comparison of Mini-MAC Code

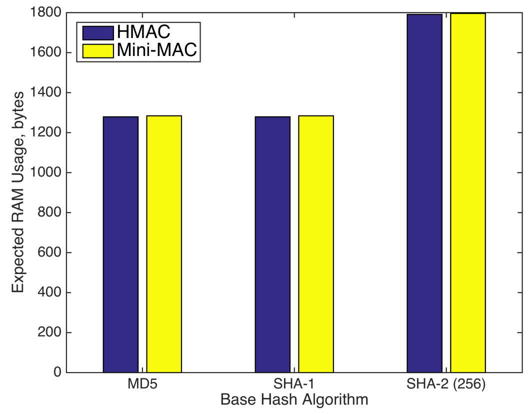

Figure 5. RAM Usage Comparison of Mini-MAC Code

Hash	R(b)	M(b)	Msg BR	Min BR
MD5	7	8b	24480	10.2
MD5	7	16b	6291360	2621.4
MD5	7	32b	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

Table 3. Time-to-Defeat for Various Configurations

little (approximately 0.68ms) delay for Mini-MAC relative to HMAC.

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

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 1kB additional code required and only 5B extra RAM relative to HMAC. Execution time increases by about 0.4ms for MD5 and SHA-1, and only by 0.68ms for SHA-256. For the environment (which typically sees 40ms between messages) this is well within the required time limits.

Table 3 shows the approximate execution time as calculated from a cycle counter based on a 32kHz clock. Based on this 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. However, the hash implementations used are non-optimized and are designed to be flexible and platform-insensitive. After optimization specifically for the MSP430 platform, it may be possible to reduce execution time.

The results are mixed. The memory and RAM usage numbers are very low, but the execution time is worrying. Captured data shows that messages are sent approximately every 40ms – 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

Table 3 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.

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 a brute force guessing attack. This means that the most efficient usage 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.

These times are based on real-world data captured from a 2010 Toyota Prius, which suggests an average message rate of around 25 messages/second. Some messages occur, however at up to 40 messages/second, although it 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.

The 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 3) to get enough information to repeat a MAC. The usage 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 3 shows the times for a stream of repeated messages, a case which is unlikely to repeat for the duration required to get enough bits to repeat a MAC. Attackers cannot simply flood the network with messages designed to get 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

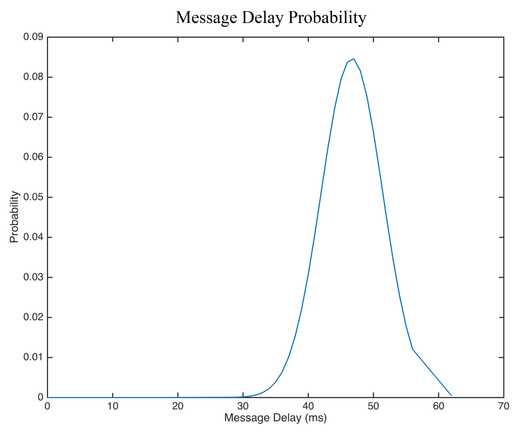

Figure 6. Probability Density Function of Message Delay – ECUs can easily be able to identify nodes spamming messages

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.

Note to ed: security analysis seems brief. still thinking on this.

9. Discussion

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

9.1. Adding Bits to Mini-MAC

Although 27.3 minutes average time-to-defeat for a brute force attack may be entirely 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.

It is possible to add two more bytes to the transmitted MAC by replacing the CRC field (2 bytes, 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.

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

Table 4. Time-to-Defeat for Various MAC Lengths

The difficulty here is in rewriting lower-level code such as firmware or drivers (depending on ECU) 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 4 shows how the brute force time-to-beat changes for different lengths of the MAC. Bits are represented by b in Table 4. 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.

9.3. Open Problems

There is a relevant open problem related to this work which can be framed by asking the question "How effective are hash functions over small messages?" As previously noted, the messages on the CAN bus are 8B, and before any practical use of a solution like Mini-MAC could be made, this question must be answered to a satisfying degree. Many practical 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 severly compromised from a hash which is weak under heavy padding as the message is so small compared to the input block size.

10. Conclusion

Automotive computer systems are extemely 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.

Mini-MAC is the first variable-length MAC protocol for the CAN bus which adds no bus traffic overhead. Using 4B 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 highly-efficient way. The world in which Mini-MAC may be useful is one in which an adversary might be another vehicle driving past you on the highway; in this scenario, being secure enough to withstand 27 minutes worth of attacks may be secure enough.

11. Acknowledgements

Schmandt and Sherman were supported in part by the National Science Foundation (NSF) under SFS grant No. 1241576.

Sherman was also supported in by a subcontract of INSuRE grant No. 1344369, and by the Department of Defense under grant No. H98230-15-10294.

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

References

[1] 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.

[2] 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.

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

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

[5] 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.

[6] 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.

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

[8] National Institute of Standards and Technology. *FIPS PUB 180-4: Secure Hash Standard*. March 2012.

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

[10] Ronald Rivest. IETF RFC1321: The MD5 message-digest algorithm, April 1992.

[11] 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.

[12] 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.