A practical solution to achieve real-time performance in the automotive network by randomizing frame identifier

Kyusuk Han, André Weimerskirch, and Kang G. Shin

October 26, 2015

Keywords Security, Authentication, CAN, CAN-FD, Controller Area Network

1. Introduction

Recent research demonstrated that it is possible to hack a currently available car model[1] to introduce safety-critical behavior using a variety of available interfaces including Bluetooth, cellular modem, USB, CD (via manipulated MP3 files), and OBD2. Thus a large variety of countermeasures was suggested in the past and many research projects are on-going to design further security solutions. One of most crucial security technology is authenticated CAN. While authenticated communication should have a high priority in the list of security technologies (e.g. most of our laptop's communication to www.google.com and email servers are protected by TLS), it has been long neglected in vehicles. The limited performance features of CAN (at most 8 byte payload per CAN packet) and the highly restricting requirements in terms of real-time performance make it almost impossible to design and implement a solution that is suitable for deployment.

We previously presented a new solution named Identity-Anonymized CAN (IA-CAN) protocol at escar USA 2014. IA-CAN provides CAN sender authentication by including an anonymous ID in each CAN message. The anonymous ID is calculated in a pseudo-random fashion by sender and receiver, and it can only be predicted by parties that share a secret key. The anonymous ID is refreshed for each CAN message and parts of the anonymous ID can be included in the 11-bit or 29-bit CAN ID field. Note that IA-CAN does not affect the ability to integrate a message authentication code (MAC) in the 64-bit payload field such that IA-CAN provides sender authentication and a traditional MAC provides message authentication. In that case, we believe that a short MAC of 4 to 32 bits, depending on use-case, is sufficient for most cases.

In this work, we present a refined version of IA-CAN, an analysis of IA-CAN's security objectives, and results of a prototype implementation. We will demonstrate that IA-CAN is superior to previous secure CAN protocols in that it protects CAN communication while preventing attacks in which an attacker tries to overwhelm an ECU by sending too many CAN messages that are processed by an ECU (ECU starvation attacks). We will also demonstrate that IA-CAN is applicable to further in-vehicle communication buses such as CAN-FD.

2. Related Work

2.1 CAN features

CAN standard is designed to operate in an isolated environment, thus it is built on a weak security architecture.

CAN transmissions encode a 0-bit (dominant

[1] http://www.wired.com/2015/07/hackers-remotely-kill-jeep-highway/

bit) by putting voltage on the CAN bus, and a 1-bit (recessive bit) by not having voltage on the CAN bus. Arbitration is implemented by the CAN-ID, where a lower numbered CAN-ID has higher priority. If two ECUs try to send at the same time, they will start transmission with the CAN-ID. Once a sender detects that there is voltage on the CAN bus (1-bit) although that sender did not pull voltage (0-bit), that sender will stop its activity and the other sender continues sending due to the higher CAN-ID priority. Each CAN message has a payload of up to 8 bytes, and there is a two byte CRC. If a node detects that there is an error (e.g. CRC error), it broadcasts a CAN error message. Note that on a CAN bus segment, every ECU monitors the CAN bus and can receive all messages; there is no concept of a sender and receiver identification.

2.2 Security research on CAN

Several approaches have been demonstrated to provide CAN message authentication. Some of the research work focused on truncating the size of the message authentication code (MAC) tag [1, 2, 3, 4, 5], instead of using a full size MAC that would require several CAN frames to submit. There are also approaches [6] on transmitting a MAC across multiple frames, or deploying the model of wireless sensor networks [7, 8].

The authors of [9] proposed a CAN-specific authentication model that runs on CAN+ [10], which is an improved and backward compatible design of CAN allowing more data in a frame. Thus the authentication data can be transmitted out-of-band.

However, since there is no intermediate protection between the sender and the receiver ECUs, every receiving ECU verifies *all* frames upon their arrival, and any attacker can transmit fraudulent frames to targets as far as CAN has bandwidth for their transport.

Requiring extensive computation on an ECU can degrade or even cripple its normal operation, especially in view of its limited resources. The authors of [11, 12, 13] used this aspect to demonstrate the feasibility of a DoS attack on ECUs.

3. Attacker Model and Assumptions

It is reasonable to be mainly concerned about remote and local attacks that do not physically manipulate the vehicle and its hardware but that compromise software running on the vehicle's electronic control units (ECU) or on an attached OBD2 dongle. For instance, we are concerned about attacks that compromise the infotainment or telematics unit, and attacks that compromise an attached OBD2 insurance dongle. The attacker can manipulate the application layer, the CAN driver layer, and the attacker can possibly use an undocumented diagnostics code of the CAN controller. Hence an attacker can inject CAN messages, read CAN messages, and possibly manipulate messages and/or flip individual bits. If an attacker manipulates a message on CAN, every device attached to the CAN bus segment will detect the modification and assume an error. An attacker cannot halt a CAN message, and the attacker cannot manipulate a message without an error being detected.

For the remainder of this paper, we assume that there is a single CAN bus segment. Modern vehicles use central gateways and separated CAN bus segments. In this case, the following considerations apply at least to the segment on which an attacker is located.

We assume that ECUs were provisioned secret keys ahead of time, e.g. at the assembly line, and that sender and receiver ECUs share a secret key. We advise to define groups of ECUs and possibly define sub-groups of CAN messages, and to assign a shared secret key to each group and/or sub-group, thus logically separating the groups. The group sets could be defined to separate safety-critical from non-safety critical ECUs, or to separate all ECUs with user accessible interfaces (e.g. infotainment, telematics, tire pressure monitoring system, etc.) from the remaining ECUs.

Our protocol requires time synchronization between sender and receiver ECU. Sender and re-

ceiver need to adjust the anonymized ID with each CAN message. Periodic CAN messages are send up to a few hundred times per second such that we require precision in the millisecond range. The time synchronization can be achieved in different ways and will likely use a combination of an ECU that acts as time server (e.g. a central gateway or body control module) and that provides a regular time-signal, leveraging a periodic CAN message that is broadcast anyways, and ECU-local limited time synchronization. It seems reasonable to authenticate the regular time-signal with a MAC.

4. ID anonymization

We now present *ID anonymization protocol for CAN* (IA-CAN). We first introduce a novel idea of ID anonymization to overcome the drawbacks of the existing approaches, and then show the advantages of IA-CAN. We finally show the detailed operation of IA-CAN.

4.1 ID anonymization by randomizing CAN ID that only authorized entity identify

Most of the existing work focuses on adding a message authentication code (MAC) in the data field for CAN frame. The glaring drawback of this approach is that receivers must do cryptographic computations to verify the validity of *all* frames, which inevitably incurs a significant additional delay. These, in turn, enable DoS attacks on resource-constrained devices like ECUs.

To overcome these drawbacks, we introduce a new concept, *ID anonymization*, in which the CAN ID is randomized on a *per-frame basis*. The randomized CAN ID appears anonymous to unauthorized entities, but it is identifiable by authorized entities. The authorized receiver ECUs update their filters by pre-computing the frame ID and, upon receiving a frame, filter it. Since the filter is ideally located in the CAN transceiver, and the process requires only XOR bit comparison, the run-time delay is negligible. Each randomized ID is used only *once* and an attacker cannot reuse a captured anonamyized ID, i.e., replay attacks are prevented.

4.2 System model for IA-CAN

The system under consideration consists of ECUs connected to the CAN bus. Each ECU contains a re-configurable frame filter that decides whether to accept a frame or not, a set of CAN IDs and corresponding set of keys. Initially the filter contains the original CAN ID, and update to anonymous ID whenever it accepts a frame.

Since CAN uses priority-based arbitration to resolve bus contention, we must ensure that the frames get transmitted according to their priority specified in their frame IDs. We thus need to ensure the preservation of frame priorities even after the ID is replaced. To meet this requirement, we keep the priority bits in the ID intact and anonymize only the remaining bits as shown in Fig. 1. If the first few bits are used to specify priority, then the remaining a bits are used for ID anonymization. For example, the SAE J1939 standard assigns the first 3-bits of the 29-bit ID for specifying priority.

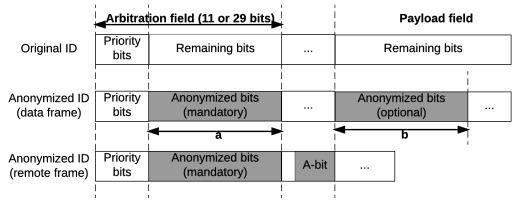

Figure 1: Priority bits in the anonymized ID are kept intact in the data frame.

To make brute-force attacks difficult on a small number of anonymized bits (i.e., $11 - a$ or $29 - a$ bits), an additional b bits are borrowed from the data field to expand the anonymization field as shown in Fig. 1.

Since the remote frame contains no data field, we cannot borrow b bits from the data field, but

we

A remote frame contains no data field. Hence to authenticate a remote frame, we cannot borrow b bits from the data field but we suggest assigning up to 5 bits in the control field (DLC and r0).

4.2.1 Key management

Each ECU maintains a set of secret keys where each key is associated with each CAN ID. The keys are seeded during manufacturing stage, and stored in a tamper-proof storage (e.g. in a SHE or HSM secure micro-controller) and can be used only by the node itself. Only ECUs who know CAN ID can send or receive a frame with an anonymized ID that has been secured with the key.

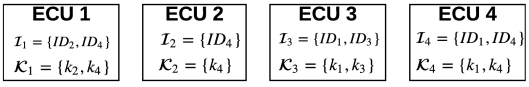

Figure 2: Each ECU stores factory-seeded long term keys allocated CAN IDs. Example shows that e_1 handles CAN ID ID_2 and ID_4 and stores factory-seeded keys k_2 and k_4.

4.3 Operation of IA-CAN

The protocol works in four phases; **session update**, **anonymous ID generation**, **payload authentication code generation** and **frame authentication**. Fig. 3 shows the brief overview of each operation.

As an example, we will consider that a sender ECU broadcasts frames associated with a CAN ID ID_j to a target ECU among the receiver group.

4.3.1 Session execution

This phase is initiated when a session is set up among ECUs for the first time, or when the session key needs to be refreshed. Let the sender

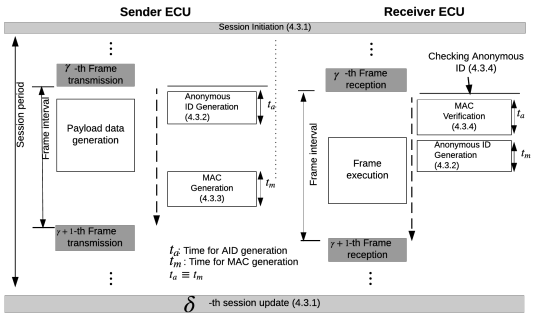

Figure 3: Overview of IA-CAN

and receivers begin a new session for frames with the original ID ID_j. They need to share a random nonce n to generate two session keys; $sk_j^{\delta,I}$ for anonymous **I**dentifier and $sk_j^{\delta,P}$ for **P**ayload authentication). δ denotes the session number. To avoid modification and replay attack by the attacker, n needs to be verifiable by the receiver.

Session initiation A session is initiated when a car is turned on. The sender selects a random nonce n and generate a hash output $INIT_j$ using a keyed hash function f_1 with k_j:

$$f_1(k_j) : \{ID_j, n\} \rightarrow INIT_j.$$

Then, the sender sends n and $INIT_j$ via CAN as shown in the format of Fig. 4. INIT is to indicate that the frame initiates a session refresh.

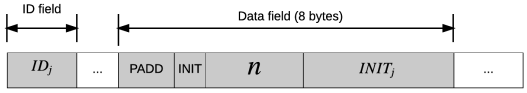

Figure 4: Data frame for a session refresh.

When a receiver accepts a frame, it verifies $INIT_j$ by generating $VERIFY_j$, where $f_1(k_j) : \{ID_j, n\} \rightarrow VERIFY_j$. If $INIT_j \equiv VERIFY_j$, the receiver accepts n to generate the session key.

Sender and receivers generate two first-session keys $sk_j^{1,I}$ and $sk_j^{1,P}$ ($\delta \equiv 1$) using k_j, where $sk_j^{1,I} = KDF\{k_j, 0, n\}$ and $sk_j^{1,P} = KDF\{k_j, 1, n\}$. KDF denotes key derivation function, which is a cryptographic one-way function.

Session refresh The session can be refreshed when the session is expiring (i.e. the counter reaches the maximum preset number), or needs to be updated with any reasons. The maximum numbers are defined by the vehicle manufacturer during the design of CAN. Considering general public driving patterns, it is reasonable to assume a session period of 24 hours.

To refresh session, the sender first randomly selects n and generates a message authentication code $UPDATE_j^\delta$:

$$f_1(k_j) : \{ID_j, n, REM_j^*\} \rightarrow UPDATE_j^\delta.$$

REM_j^* is the remainder of previous anonymous ID generation and is used to avoid replay attack. How to get REM_j^* is described in section 4.3.2. Then, the sender broadcasts n and $UPDATE_j^\delta$ to CAN bus as shown in the format of Fig. 5. $AD_j^{1,\gamma}$ is an anonymous ID in the arbitration field, and $AD_j^{2,\gamma}$ is b bits located in the data field.

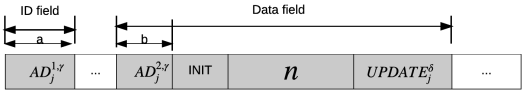

Figure 5: Data frame for a session refresh.

When e_2 receives a frame, it verifies $UPDATE_j^\delta$ by generating $VERIFY_j$, where $f_1(k_j) : \{ID_j, n, REM_j^*\} \rightarrow VERIFY_j$. If $UPDATE_j^\delta \equiv VERIFY_j$, a receiver accepts n to update the session key.

Sender and receivers generate $sk_j^{\delta,I}$ and $sk_j^{\delta,P}$ using k_j, where $sk_j^{\delta,I} = KDF\{k_j, 0, n\}$ and $sk_j^{\delta,P} = KDF\{k_j, 1, n\}$.

4.3.2 Anonymous ID generation

Both sender and receivers generate anonymous IDs using the session key $sk_j^{\delta,I}$.

Let, during δ-th session, γ-th anonymous IDs for data frame AD_j^γ and for remote frame RD_j^γ.

Sender side As soon as the sender finishes sending the (previous) frame with $AD_j^{\gamma-1}$, it generates next IDs using f_1 and f_2 with $AD_j^{\gamma-1}$, $RD_j^{\gamma-1}$, the remainder $REM_j^{\gamma-1}$, and $sk_j^{\delta,I}$ as shown in Fig. 6. Note $AD_j^0 \equiv RD_j^0 \equiv ID_j$ and REM_j^0 is a string of 0s, whose size is equivalent to other remainders. In addition to f_1 a segmentation function f_2 is used for this phase. There can be several ways to construct f_2, and we assume in this paper that f_2 to truncate the output from f_1 to match the size of AD_j^γ and RD_j^γ .

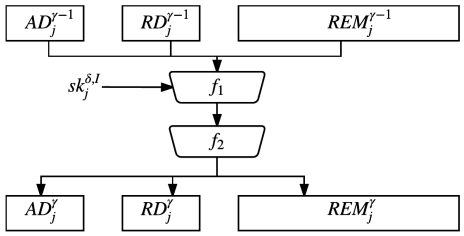

Figure 6: ID generation using f_1 and f_2

The sender stores AD_j^γ, RD_j^γ and the remaining bits REM_j^γ after the ID is extracted. RD_j^γ is used only when receivers need to send a remote frame.

However, we assume f_2 to truncate $OUTPUT_j^\gamma$ to fit the size of id_j^γ and rd_j^γ.

Receiver side The receiver updates the anonymous ID in their filter if e_2 accepts the frame or if the time period for ID_j ends.

As soon as the receiver accepts or drops (i.e. time-over) the previous frame with $AD_j^{\gamma-1}$, it executes the same process displayed in Fig. 6 without waiting for the next message. The filter **F** immediately replaces $AD_j^{\gamma-1}$ with AD_j^γ in the

frame acceptance list, and receivers now wait for the next frame with the new ID id_j^γ.

Anonymous ID for a remote frame A *Remote* frame is transmitted by a receiver ECU, when it needs to request a data frame. Since the Remote frame does not contain the data field, it can use 4 bits of the DLC field for an anonymous ID. We can also use the reserved bit r0.

Since the CAN standard specifies both data and remote frames to share the same identifier, we design different anonymous IDs for the two types of frame. While the ID of a Data frame is AD_j^γ, we assign $RD_j^\gamma \neq AD_j^\gamma$ as the ID of a Remote frame which $(16-a)$ bits (the standard format) or $(34-\alpha)$ bits (the extended format) extracted from REM_j^γ.

As soon as the sender accepts the remote frame with RD_j^γ, it send the data frame with AD_j^γ.

4.3.3 Payload data authentication code generation

The use of anonymous IDs prevents injection of unauthorized frames. However, there is still possibility that attacker's physically modified ECU can modify frame data by overriding CAN bus, although they represent extreme cases. Thus we also provide data authentication for payloads. Let a MAC is generated for a payload data.

When the data is generated, sender ECU generates a c-bit MAC. The sender then puts the MAC into the data field. The size of c is a design parameter, where 8–32 bits can be assigned.

For practical security strengths, 11(or 29) $-a+b+c \geq 32$ is recommended, where 11(or 29) $-a+b$ is the size of anonymous ID and c is the size of MAC.

Remark that MAC is be generated only after frame data is generated/received, while anonymous ID has no relation with the frame data, and thus can be generated immediately after the previous frame is transmitted/received.

For the Remote frame in Fig. 7(b), this phase is skipped since it has no data field.

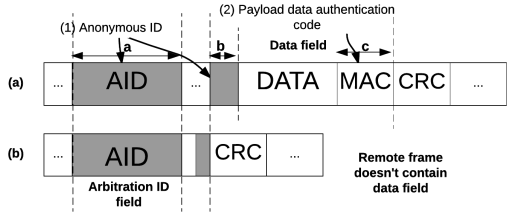

Figure 7: (a) A data frame that has data field in the payload requires both anonymous ID and MAC; (b) a Remote frame only requires anonymous ID.

4.3.4 Frame authentication by receiver

When the receiver reads a message, it checks whether the frame ID AD_j^γ matches one of IDs in the acceptance filter list $\mathbf{F}$. If a match is found, the receiver accepts the frame and then performs an integrity check using CRC.

For a non-zero-bit data frame, the receiver accepts the message after verifying MAC with $sk_j^{\delta,P}$.

5. Protocol Features

This section provides security and performance features of our IA-CAN.

5.1 Sender authentication

IA-CAN provides sender authentication in addition to message authentication. We define sender authentication in the sense that only the parties that have access to the shared key are able to generate the IA tag. This is protected by the use of standard cryptographic mechanisms, and a third party without knowledge of the shared secret key cannot predict the IA tag. Hence the receiver knows that the legitimate sender originated the CAN message.

Skipping message authentication enables truly real-time operation, however the receiver has no assurance that the message that was originally

broadcast by the legitimate sender was not manipulated in between.

For example, an attacker will be able to flip 1-bits to 0-bits in the CAN-ID, payload and CRC, making sure that the CRC stays valid. However, the original sender will detect the modification on the CAN bus, since voltage is put on the CAN bus when no voltage is supposed to be used, and send an error frame. This error message will again be protected with IA-CAN. Again, the attacker can modify the error message, however, the attacker cannot halt the error message. For the majority of periodic CAN messages, we believe this to be sufficient. We recommend to use an additional MAC in the payload for CAN messages that might have an immediate impact.

5.2 Resilience against DoS/starvation attacks

It is unclear whether it is possible to mitigate certain denial-of-service (DoS) attacks, such as modifying the sender's baud rate to affect the entire CAN bus. A subtle attack are so-called starvation attacks to a single ECU in which CAN messages are send to the attacked ECU in order to overwhelm and deactivate it.

Using traditional MAC protection, it is potentially possible to starve an ECU by sending CAN messages with a CAN-ID that this ECU will process, use the maximum rate that this ECU was programmed to accept, and include a faulty MAC. The ECU will then try to evaluate the MAC by performing a MAC calculation. Due to the large number of CAN messages and the fact that verifying MACs requires more computational resources than only processing a CAN message, the ECU will eventually be overwhelmed and there will be a significant latency between the time a CAN message is received and the time a CAN message is processed, hence violating the real-time requirements.

Instead IA-CAN allows to pre-calculate IA tags (i.e. anonymous ID). Also there is one IA tag per time period, making the number of messages that can be validated highly deterministic. Therefore a starvation attack as described above using IA-CAN will not affect the ability of the ECU to verify received CAN messages in real time, hence ensuring availability to process crucial legitimate CAN messages.

In different words, IA-CAN always requires the same computational effort and latency time, regardless whether an attack is mounted or not.

5.3 Key assignment

For IA-CAN, it seems reasonable to define a shared key between sender and receiver(s) for each CAN-ID. Sharing the same key for several CAN-IDs is not advisable since the same IA tag would then be used for several CAN-IDs, hence enabling an attacker to monitor an IA tag and then broadcasting a message with a different CAN-ID using the same IA tag.

Hence a sender and receiver needs to compute a new IA tag in each time period for each CAN-ID that this node broadcasts or monitors. When using a MAC, keys can be assigned far more flexible. For instance, all ECUs can be assigned the same key (which is not advisable though), overlapping groups of ECUs with a shared key can be determined, or a key can be determined to each CAN ID.

For IA-CAN, for each CAN-ID an IA tag must be calculated per time period. A standard MAC has clear advantages since the required computing resources do not depend on the number of keys but only on the number of received messages.

5.4 Periodic messages vs. event based CAN messages

We believe that IA-CAN is a proper fit for periodic messages, especially for those messages with a periodicity that matches the time-period synchronization used by IA-CAN. Then for each periodic message, a new IA tag is used. If a second CAN message with the same IA tag is received, the receiver discards that message since the only conclusion is that an attacker read the first message and then injected a CAN message with the

same IA tag. While IA-CAN can also be utilized for event based CAN messages, it is less efficient here. Say, an event based message is only sent once per vehicle trip, however, once it is sent, that message is send several times in a short time period. To use IA-CAN, we need to define a time period for synchronization that reflects the periodicity of the event based message such that one message is sent per time period. However, this requires to calculate the IA tag for each time period, even though no message is send during that time period. Hence, for event based CAN messages, it seems advantageous to use a standard MAC rather than IA-CAN.

5.5 Relaxed time synchronization

Timestamp requires the transmission of timestamp itself if the receivers cannot know when the timestamp itself is generated. Instead of using the timestamp, we use REM per each time interval. REM is never transmitted in the communication, still shared between authorized entities.

5.6 Resume after errors

If an error occurs, e.g. because an ECU is reset, it first needs to be synchronized to the current time period in order to be able to calculate the proper IA tag. We assumed above that a time synchronization mechanism is available, e.g. by using a CAN time server. If the time synchronization signal is broadcast regularly, no additional mechanism is required to resume after an error.

5.7 Future communication technology

Future technology, such as CAN-FD, will increase throughput significantly. One could easily argue that the additional offered payload (e.g. 64 bytes for CAN-FD vs. 8 bytes for CAN) enables the straight-forward use of a MAC in each message that requires at 8 bytes, hence still leaving enough payload bytes for the message content. However, certain aspects will remain similar to today. With higher throughput it will actually be easier to mount ECU starvation attacks since ECU performance does not automatically increase and since there is more bandwidth available to inject forged CAN packets. Hence we believe that IA-CAN is even more valuable for future technologies such as CAN-FD.

6. Performance evaluation

We evaluate the efficiency of IA-CAN by comparing general payload authentication models against injection and flooding attack on CAN.

6.1 Computation overhead of IA-CAN

We first show the computation overhead ove IA-CAN. Table 1 lists the computations required for IA-CAN, where **H** is the amount of computation for a cryptographic one-way compression function such as SHA-1, **XOR** the computation of XOR, and **F** the amount of computation for a non-cryptographic function that divides one string into multiple parts. The overheads of **XOR** and **F** are negligible compared to **H**.

Table 1: Computation overhead

Phase	Computation	Operation
Session refresh	3 **H**	Session initiation time
Anonymous ID generation	1 **H** + 1 **F**	Idle-time
Payload authentication code generation	1**XOR** + 1 **H**	Run-time (Sender)
Data frame authentication	1**XOR** + 1 **H**	Run-time (Receiver)
Remote frame authentication	1**XOR**	Run-time (Receiver)

The actual computation time of **H** varies with processors. We tested HMAC-SHA1[2] on 8-bit processor (16MHz Arduino UNO R3) and 32-bit processors (40MHz Chipkit 32MAX, NXP LPC1768). **H** of 8-bit processor takes 12 ms, while **H** of 32-bit processors take 23–230μs. For the test on ARM Cortex M3, **H** takes approximately less than 25 μs.

[2]the APPENDIX A of [15]. We also used the SHA-1 code downloaded from http://code.google.com/p/cryptosuite/

6.2 Attack simulation

We then evaluate IA-CAN with attack simulation. We compare IA-CAN with general payload authentication models [7, 8, 9, 6].

6.2.1 Simulation results

We first tested simulated flooding attacks using *Mathematica 9* on an Intel i7 2.7GHz dual core processor. We compute HMAC-SHA1 operation 25000 times for each frame assuming the attacker injects tons of frames to 100% utilized CAN bus.

The attacks proceed with frames at $1000ms$ and $5ms$-intervals on the 125/500/1000kbps CAN bus. For $1000ms$-interval frame, IA-CAN shows 90–$679ms$ with 6-bit AID, 30–156 ms with 16-bit AID, those are far below the $1000ms$ bound.

For the attack proceeded with $5ms$-interval frames, IA-CAN for the frame takes approximately 1.7/2.4/3.2ms, while the general PA using MAC takes 23/85/167ms, which exceeds the $5ms$ time-bound. Using IA-CAN for the data frame, assignment of 6-bit anonymous IDs takes 4.1/4.8/8.2ms, while assigning 16-bit anonymous IDs still takes 2.8/4.6/5ms.

6.3 Attack implementation

We then implemented CAN testboard as Fig. 8. We used mbed NXP LPC1768 (microcontroller) with MCP2551 (the CAN transceiver) as ECU. These devices are used in the automotive industries. We connected sender, receiver and attacker ECU to 500 kbps CAN bus. We also used HMAC-SHA1 as the cryptographic primitive to generate anonymous ID and MAC.

Fig. 9 shows the comparison of unsecure CAN, Secured with MAC, Secured with AID and Secured with AID and MAC cases. While only authenticating messages incurs significant performance degradation, IA-CAN shows real-time performance with immediate sender authentication.

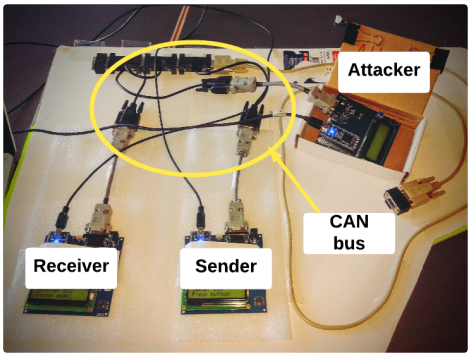

Figure 8: Test environment

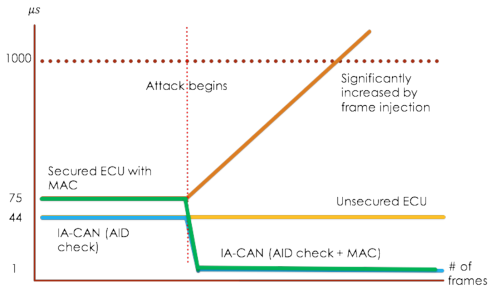

Figure 9: Comparision under attack

7. Conclusion and Outlook

In this paper, we presented the protocol and the prototype implementation results of IA-CAN. IA-CAN is a new secure CAN protocol that provides sender authentication in addition to message authentication. We argued why sender authentication is sufficient for most use-cases of automotive in-vehicle communication, and we listed security and performance features of IA-CAN. IA-CAN is superior to the standard MAC approach by preventing ECU starvation attacks, and it is highly deterministic in terms of required computational effort whether an attack is mounted or not. IA-CAN also works on CAN-FD. The performance results of our prototype

implementation demonstrated those features.

References

[1] C. Szilagyi and P. Koopman, "Flexible multicast authentication for time-triggered embedded control network applications," in *Dependable Systems Networks, 2009. DSN '09. IEEE/IFIP International Conference on*, 29 2009-july 2 2009, pp. 165 –174.

[2] ——, "Low cost multicast authentication via validity voting in time-triggered embedded control networks," in *Proceedings of the 5th Workshop on Embedded Systems Security*, 2010.

[3] C. W. Lin and *et al.*, "Cyber-security for the controller area network (CAN) communication protocol," *ASE Science Journal*, vol. 1, no. 2, pp. 80–92, December 2012.

[4] H. Schweppe, *et al.*, "Car2X communication: Securing the last meter," *WIVEC 2011, 4th IEEE International Symposium on Wireless Vehicular Communications, 5-6 September 2011, San Francisco, CA, United States*, pp. 1–5, Jun. 2011.

[5] O. Hartkopp, *et al.*, "MaCAN - message authenticated CAN," *escar 2012, Embedded Security in Cars Conference 2012, Berlin - Germany*, November 2012.

[6] D. Nilsson, *et al.*, "Efficient in-vehicle delayed data authentication based on compound message authentication codes," in *Vehicular Technology Conference, 2008. VTC 2008-Fall. IEEE 68th*, 2008, pp. 1–5.

[7] B. Groza and P.-S. Murvay, "Broadcast authentication in a low speed controller area network," *E-Business and Telecommunications, International Joint Conference, ICETE 2011, Seville, Spain, July 18-21, 2011, Revised Selected Papers*, vol. 314, pp. 330–344, Feb. 2012.

[8] B. Groza, *et al.*, "LiBrA-CAN: a lightweight broadcast authentication protocol for controller area network," *Proceedings of 11th International Conference, CANS 2012, Darmstadt, Germany.*, pp. 185–200, December 2012.

[9] A. V. Herrewege, *et al.*, "CANAuth - a simple, backward compatible broadcast authentication protocol for CAN bus," *10th escar Embedded Security in Cars Conference*, November 2011.

[10] T. Ziermann, *et al.*, "CAN+: A new backward-compatible controller area network (CAN) protocol with up to 16x higher data rates," *in DATE. IEEE*, pp. 1088–1093, 2009.

[11] S. Checkoway, *et al.*, "Comprehensive experimental analyses of automotive attack surfaces," in *SEC'11: Proceedings of the 20th USENIX conference on Security*. USENIX Association, Aug. 2011, pp. 1–16.

[12] T. Hoppe, S. Kiltz, and J. Dittmann, "Security threats to automotive CAN networks - practical examples and selected short-term countermeasures," *SAFECOMP 2008, LNCS 5219*, pp. 235–248, 2008.

[13] K. Koscher, *et al.*, "Experimental security analysis of a modern automobile," *Security and Privacy (SP), 2010 IEEE Symposium on*, pp. 447–462, 2010.

[14] L. Apvrille, *et al.*, "Secure automotive onboard electronics network architecture," in *FISITA 2010 World Automotive Congress, Budapest, Hungary*, vol. 8, 2010.

[15] Federal Information Processing Standards Publication, "The keyed-hash message authentication code (HMAC)," *FIPS PUB 198*, 2002.