Authenticating CAN Bus

CPS and IoT Security

Alessandro Brighente

Master Degree in Cybersecurity

Università degli Studi di Padova

SPRITZ Security & Privacy Research Group

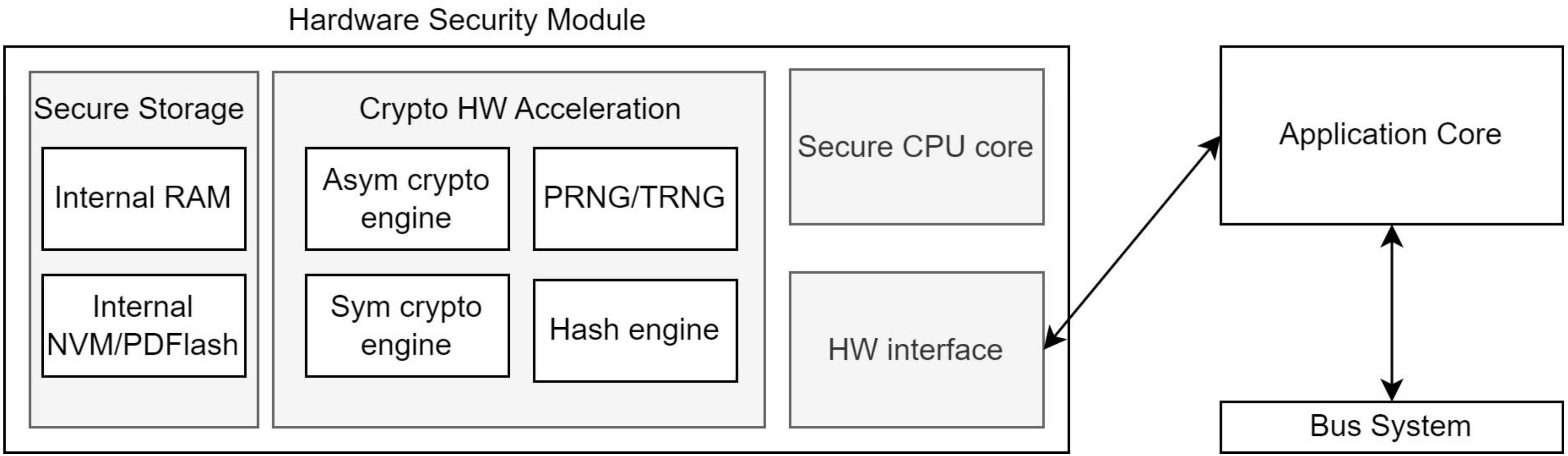
Missing Parts

- We would like to develop cryptographic protocols to deal with security problems in CAN bus
- We said that CAN IDs do not identify the transmitter, so we need something to identify ECUs
- We need to handle security parameters and cryptographic primitives, so we need to modify ECUs

Handling of Cryptographic Primitives

- To offer accountability and confidentiality, we need nodes to securely store cryptographic primitives
- We also need to guarantee real-time constraints and handle limited resources in terms of memory and computing capabilities
- European E-Safety Vehicle Intrusion Protected Applications (EVITA) is a project with the objective of designing such security-enabled ECUs
- They focused on the design of Hardware Security Modules (HSM) as a root of trust to be easily integrated as on-chip extensions to ECUs

Hardware Security Module



General architecture of an automotive HSM. The module resides in the same chip as the application CPU core

HSM Levels

- The components of the HSM can be divided into mandatory and optional depending on the security requirements that needs to be fulfilled
- EVITA specifies three variants to meet different security levels and cost effectiveness
  - Full HSM
  - Medium HSM
  - Light HSM

Full HSM

- Focuses on protection in-vehicle networks from vehicle-to-everything communication
- Maximum level of security and performance with main cryptographic building blocks being
  - ECC-256-GF(p): High-performance asymmetric cryptographic engine based on a high-speed 256-bit elliptic curve arithmetic
  - WHIRLPOOL AES-based hash as per NIST indications
  - AES-128 symmetric encr/decr
  - AES-PRNG with a true random seed from an internal physical source
  - 64-bit monotonic counter as clock alternative

Medium HSM

- Focuses on in-vehicle operations and their security
- It is compatible with the full version, but it lacks
  - Hardware error correction code engine
  - Hardware hash engine
- It can execute very fast symmetric cryptography operations in hardware and some non time-critical asymmetric cryptography operations in software
- Security credentials are stored out from the application CPU

Light HSM

- Focuses on protecting ECUs, sensors and actuators
- Security limited to a very specialized symmetric AES hardware accelerator
- All security credentials are handled by the application CPU
- Allow to meet cost and efficiency requirements of sensors and actuators

The second need: identification

- In a broadcast protocol as CAN bus we do not need identification
- However, when we think about authentication we need to talk about sender A and receiver B
- We need to somehow modify the standard to account for this need
- Currently, there is no specification on how to attach identity information to packets or how to create such identities
- In industrial context, SAE J1939 states that it is possible to use the extended ID field (29 bits instead of 11) to include ECU-specific identifiers

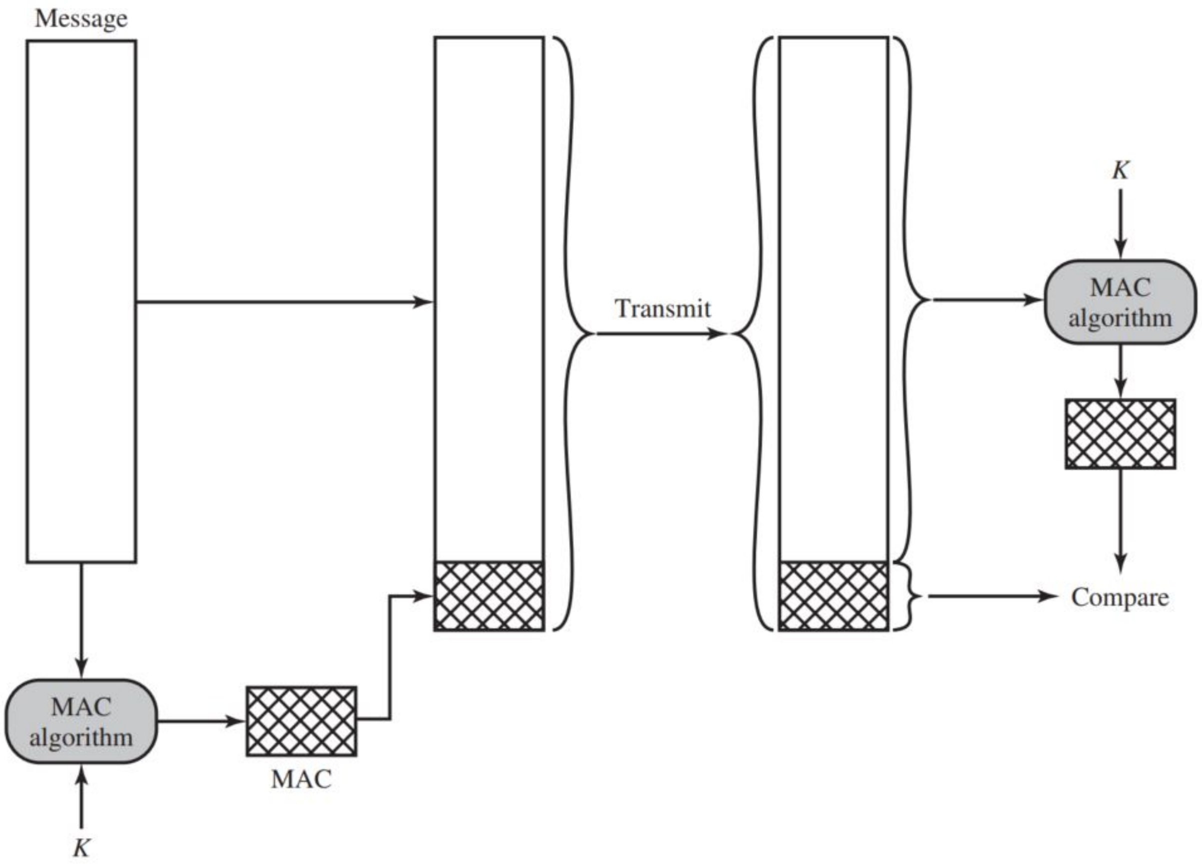
Some authentication protocols in CAN bus

- We now review some of the authentication protocols that have been proposed in the literature for authentication purposes
    - CANAuth (2012)
    - CaCAN (2014)
    - LeiA (2016)

CANAuth

- It is based on Hash-based Message Authentication Codes (HMACs) and session keys
- To avoid attaching the HAMC to the standard CAN frame or to send a single packet over multiple runs, authors decided to use CAN+
- CAN+ is a protocol that increases CAN speed (data rate) up to 16 times
- It exploits a *gray zone* between synchronization and sampling to transmit additional information by overclocking

Message Authentication Code



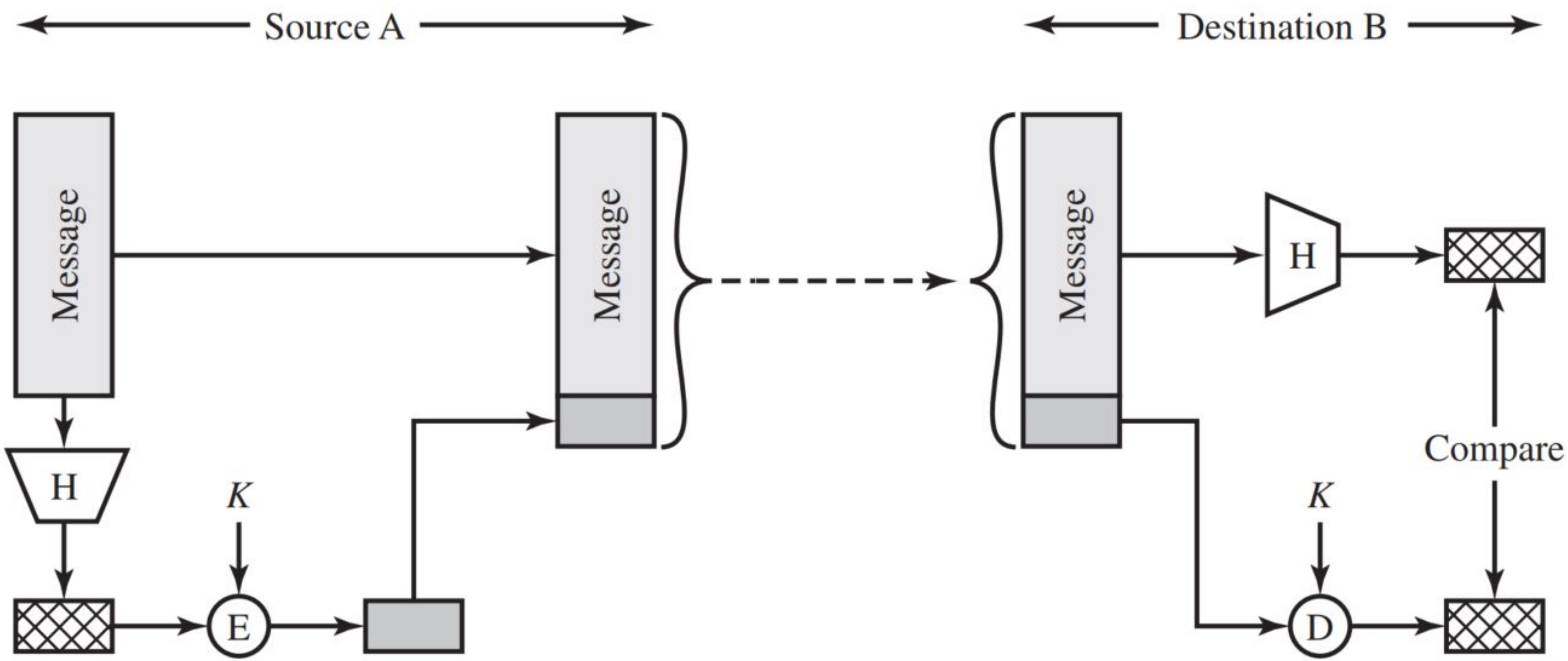
Message Authentication Code (MAC)

- If the received code matches the calculated code:
    - the message has not been altered
    - the message is from the alleged sender
    - the receiver can be assured of the proper sequence (if any)

- To compute MAC it is typical to use DES

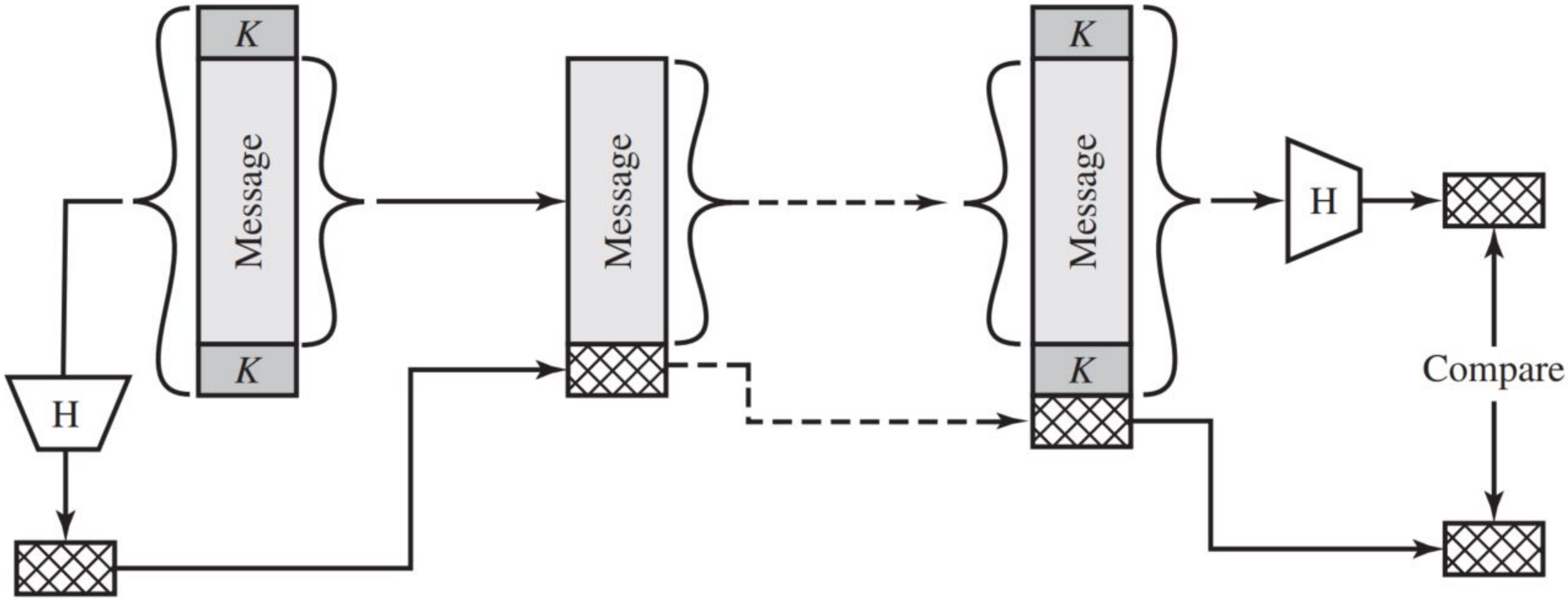
Hashed Authentication

- Hash functions can be used for authentication



Hashed Authentication

- Hash functions can be used for authentication



CANAuth

- The key establishment procedure assumes that each ECU has a pre-shared 128-bit master key K_i
- The protocol uses group messages, i.e., messages that can be authenticated using the same group key
- We denote the i-th group as $\mathcal{G}_i$
- Keys are stored in a tamper-proof memory

CANAuth

- The session key is generated by the first node that attempts to send a message from the i-th group

$$K_{s,i} = HMAC(K_i, ctrA_i \parallel r_i)\ mod\ 2^{128}$$

- The HMAC receives as input a counter for message M and a random number
- By possessing the master key, every node can generate the correct session key upon knowing the counter and random value
- The first node to transmit is the one that sets these parameters

CANAuth

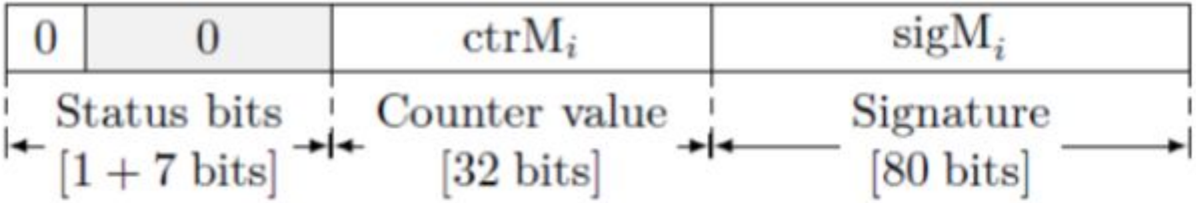
- The session key establishment has two phases
  - Transmission of parameters, where the transmitter broadcasts a CAN message with attached a CAN+ frame delivering the counter and random number
  - Authentication of key establishment, where the sender broadcasts a second message containing a signature that proves the effective knowledge of the session key

$$sigA_i = HMAC(K_{s,i}, ctrA_i \parallel r_i) \bmod 2^{112}$$

- If any of the nodes detects an error in at least one of these two messages, the procedure restarts with a new counter

CANAuth

- Once the session key establishment process has been successfully completed it is possible to authenticate messages
- The CANAuth data frame M is composed by status bits, a 32 bits counter value (different from the one used for session key agreement) for replay attack resistance and a signature



CANAuth

- The message counter shall be increased by at least one for every authenticated message
- All ECUs keep track of this counter, and they accept a packet if and only if the counter in the packet is greater than the stored one
- If the counter is ok, they check the signature

$$sigM_i = HMAC(K_{s,i}, ctrM_i \,||\, M_i) \; mod \, 2^{80}$$

CaCAN

- Centralized authentication in CAN (CaCAN) introduces a central Monitor Node (MN) with the purpose of authenticating all ECUs in the network
- Notice that MN is central only from a logical point of view, no topology modification
- CaCAN requires MN and ECUs to share a 512-bit key to compute MACs
- This key is computed from a pre-shared secret S assumed to be stored in a Read-Only-Memory together with the unique ECU identifier

CaCAN

- During the authentication and key distribution phase, CaCAN implements a two-way authentication with a challenge-response phase
- Suppose that the j-th ECU needs to authenticate itself to the MN
  - The MN sends a random nonce n to the ECU
  - Both the MN and ECU compute the authentication code

$$AC = \text{SHA-256}\,(S \parallel n)$$

  - After some time, the MN sends to the ECU a data frame whose payload is composed by few bits of AC
  - The ECU checks its correctness, and replies with a continuation of the AC
  - MN checks its correctness

CaCAN

- When nodes exchange messages, the MN checks that they carry the MAC

$$MAC_i = HMAC\,(ECU_{ID}, msg_i, FC_i, K_j)$$

- It contains the ECU’s identifier, the payload, a counter, and j-th node’s key
- The MAC is 1 byte long, i.e., we just keep the first 8 bits of the MAC
- If it detects that there is an error in a MAC, it overwrites the message with an error frame

CaCAN

- The counter implemented in CaCAN is 32-bits long FC = UC || LC, so it cannot be sent along with the payload
- The authors decided to attach only the lowest few bits LC
- The MN however stores the full counter for each node and applies the following policy
  - If $LC_i = OLC_i$ (OLC_i are the holding lowest bits of the monitor node), the message is discarded as it is classified as a reply attack.
  - If $LC_i > OLC_i$, the message is accepted and OC_i is updated.
  - If $LC_i < OLC_i$, the message is accepted, OLC_i is updated ($OLC_i = LC_i$), and the holding upper value OUC_i of the counter is increased by 1.

LeiA

- Lightweight Authentication Protocol for CAN (LeiA) is the first protocol compliant with AUTOSAR specifications
- It uses session keys and lightweight cryptographic primitives
- Each ECU needs to store a tuple $(ID_i, K_i, e_i, K_i^s, c_i)$
    - CAN ID for data type i
    - 128-bit long symmetric key to generate session keys
    - 56-bit epoch value increment at each vehicle start up
    - 128-bit session key to generate MACs
    - 16-bit counter included in MAC computations

LeiA

- ECUs are initialized by generating a tuple containing
  - a collection of master keys, one for each ID $s = < K_0, ..., K_{n-1} >$
  - A collection of epoch and counter values $n_s = < (c_0, e_0), ..., (c_{n-1}, e_{n-1}) >$

- For each ID, they Key Generation Algorithm (KGA) is used to derive the corresponding session keys as
  - $e_i = e_i + 1;$
  - $K_i^s = KGA(K_i, e_i);$
  - $c_i = 0.$

LeiA

SPRITZ Security & Privacy Research Group

Università degli Studi di Padova

- After session key generation, ECUs can authenticate their messages
- Update the counter c and generate MAC

$$MAC_i = AGA(K_i^s, c_i, msg)$$

- AGA is the Authentication Generation Algorithm and msg is the payload of the message
- If the counter overflows, everything restarts and e is incremented to compute a new session key

- For any message, the counter is included in the extended identifier field preceded by a 2-bit command code that specifies the content of the payload
  - 00 for normal data;
  - 01 for the MAC of the data;
  - 10 for the epoch value;
  - 11 for the MAC of the epoch.

LeiA

- LeiA comes with a resynchronization method
- It is used when a MAC cannot be verified and the receiver sends an error signal
- When this is the case, the sender broadcasts a message containing its current epoch and counter values and the MAC for the epoch value
- This allows receivers to resynchronize their epoch and counter
- Notice that receivers will update their counters only if the received ones are higher than their currently stored ones, otherwise it might be a replay attack