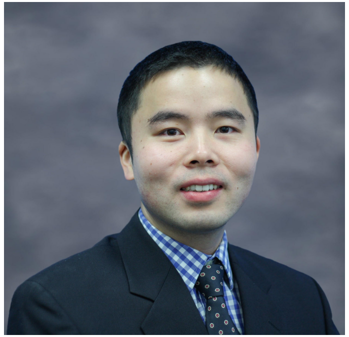
SHE Functional Specification and Memory Update Protocol

Timothy Huang

July 2020

About Me - Timothy Huang



Cybersecurity Software Engineer
Magna International · Full-time
Nov 2019 – Present · 9 mos
Auburn Hills, Michigan, United States

- AUTOSAR based backup camera ECU project.
- Implemented and tested ECU in-vehicle key management software.
- Implemented intrusion detection applied to ECU in-vehicle network.

Embedded Software Engineer
Delphi Technologies · Full-time
Dec 2017 – Nov 2019 · 2 yrs
Troy, Michigan, United States

- Lead the cyber-security software work for a hybrid plugin vehicle inverter project.
- Implemented the PWM API on the NXP eMIOS module through object-oriented design.

Embedded Software Engineer
Delphi · Full-time
Jul 2016 – Dec 2017 · 1 yr 6 mos
Troy, Michigan, United States

- Supported all phases of the software development include requirement analysis, design, development, review and testing for Delphi's Engine Management System.

Education

Rochester Institute of Technology
Master of Science - MS, Computer Science

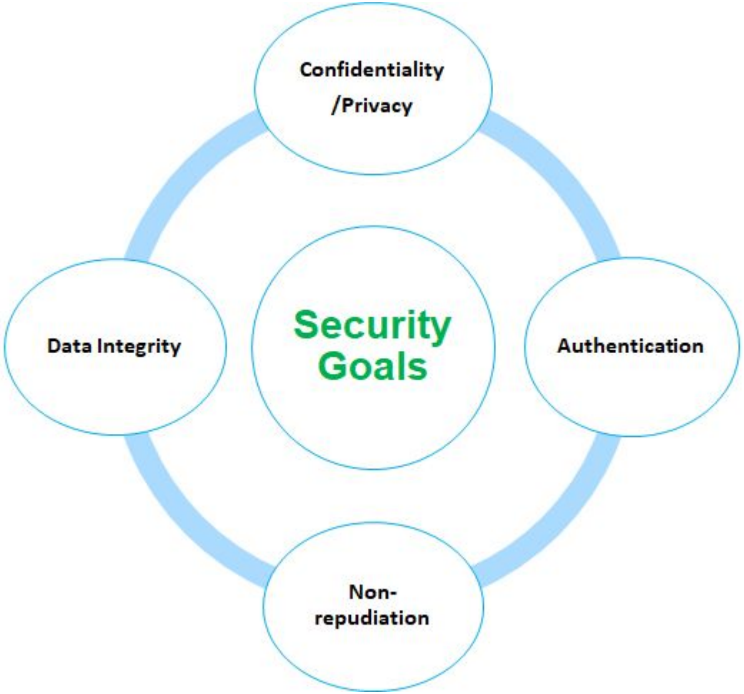
Beihang University
Master of Engineering - MEng, Electronic and Communication Engineering

Beihang University
Bachelor of Engineering - BE, Integrated Circuit Design

LinkedIn: https://www.linkedin.com/in/yhtim/

Contents

The Purpose of Cryptography



- ***Confidentiality/Privacy***

  > Ensuring no one can read the message except the intended receiver. **Prevent eavesdropping**

- ***Data Integrity***

  > Assuring the received message was not altered in any way. **Prevent tampering**

- ***Authentication***

  > Proving one's identity. **Prevent spoofing**

- ***Non-repudiation***

  > Preventing the sender from later denying they sent the message.

Cryptography Techniques

- **Symmetric Key Cryptography (E.g., AES-128)**

  Stream Ciphers – Employs only “**Confusion**”. E.g., CR4, A5/1

  Block Ciphers – Employs both “**Confusion**” and “**Diffusion**”. E.g., DES, AES

- **Asymmetric Key Cryptography (E.g., RSA)**

  Based on **Number Theory**, not “Confusion” or “Diffusion”

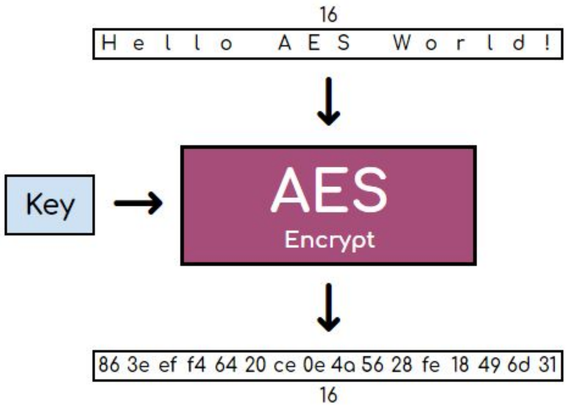
- **Secure Hash Algorithms (E.g., SHA-256)**

  Map data of **arbitrary size** onto data of a **fixed size**.

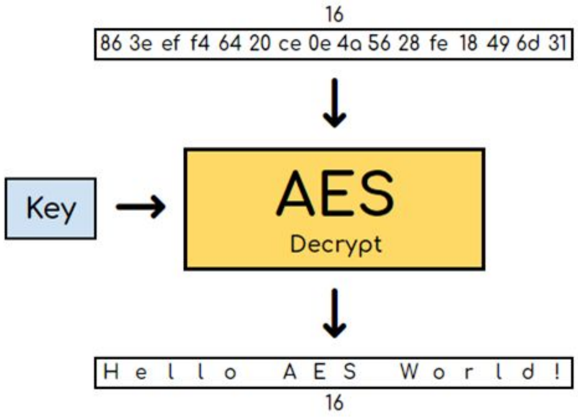
  One-way Function: Easy to get the hash with the given input, but computationally hard to reverse.

Symmetric Key Cryptography - AES

- **AES**: Advanced Encryption Standard.
- **Block cipher**: This algorithm takes a fixed size input (in this case, 16 bytes – 128 bits) called the plaintext, and generate an output of the same size called the ciphertext.
- **Symmetric**: It means that the same key is used for both encryption and decryption.



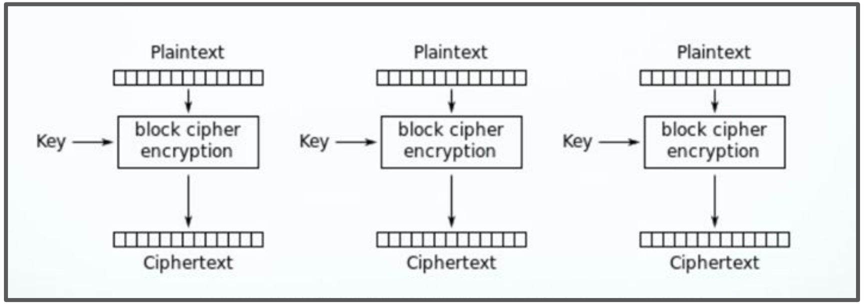
Encrypt data



Decrypt data

Symmetric Key Cryptography - Modes of Operation

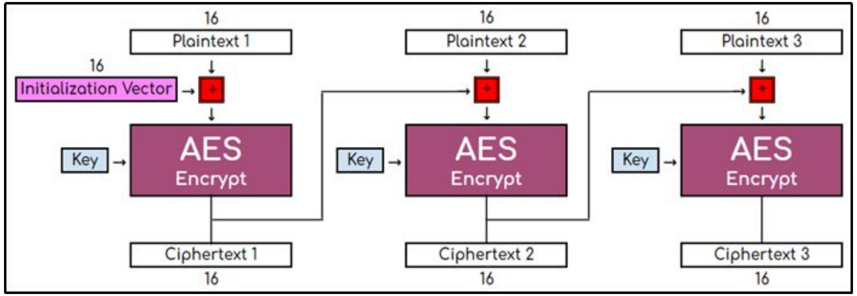
ECB mode



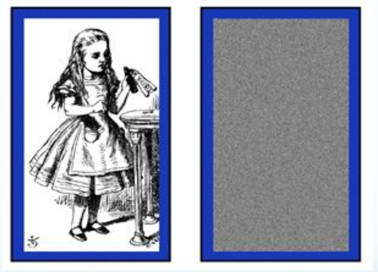
Electronic Codebook mode encryption

ECB Mode Encryption

CBC mode



Cipher Block Chaining mode encryption



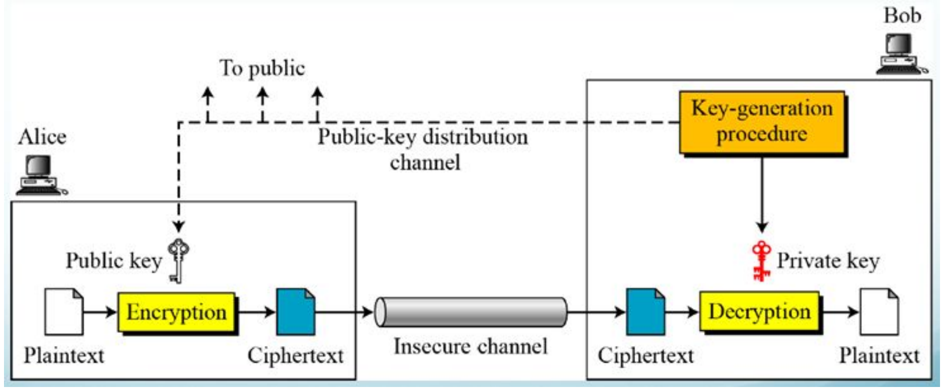
CBC Mode Encryption

Asymmetric Key Cryptography

Two keys:

- Sender uses recipient's Public Key to Encrypt.
- Recipient uses his/her Private Key to Decrypt.

Asymmetric weakness: ***Slow*** ☹



Secure Hash Algorithm Property

- A basic **compression function** on blocks of data: arbitrary size to a fixed size.
- It is a **one-way** function: Message → Hash (easy and quick); Hash → Message (hard).
- It is **deterministic** so the same message always results in the same hash.
- It is **quick to compute** the hash value for any given message.
- **avalanche effect**: A small change to a message should change the hash value so extensively that the new hash value appears uncorrelated with the old hash value.
- It is **infeasible to find two different messages** with the same hash value (ideally should have no collision).

Cryptography Techniques

- Symmetric Key Cryptography
- Asymmetric Key Cryptography
- Secure Hash Algorithm

Security Services

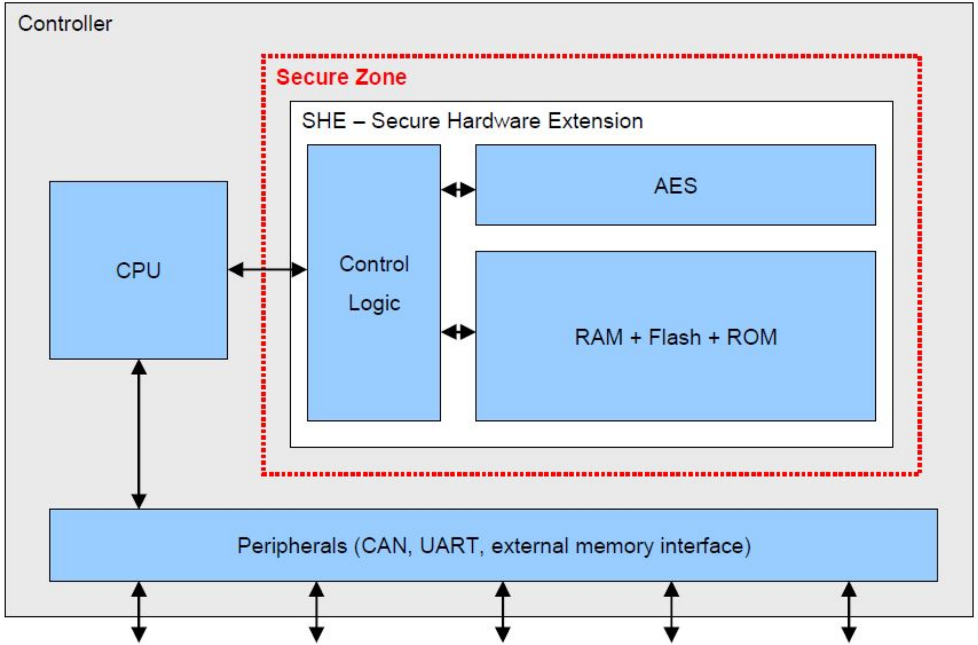
- ***Encryption/Decryption***
- ***Message Authentication Code***
- ***Key Derivation***
- Hash
- Digital Signature

Tools and techniques in Cryptography that can be selectively used to provide a set of desired security services.

Contents

Simplified Logical Structure of SHE

- The Secure Hardware Extension (SHE) is an **on-chip extension** to any given microcontroller.
- It is intended to move the control over cryptographic keys from the software domain into the hardware domain and therefore **protect** those **keys from software attacks**.



Simplified logical structure of SHE

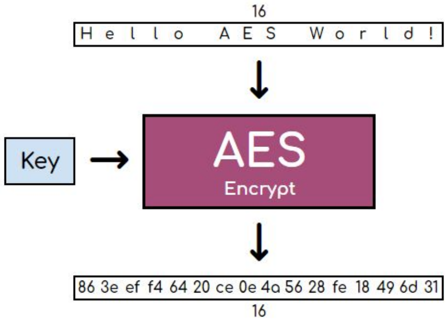
Algorithms in SHE

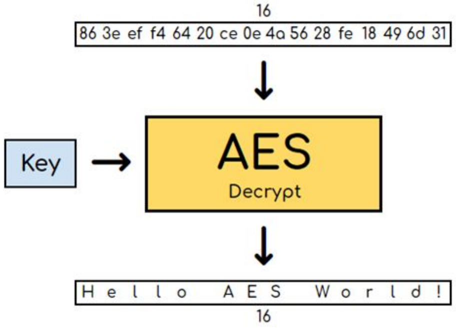
All cryptographic operations of SHE are processed by an ***AES-128***:

- **Encryption/Decryption**: ECB mode for single blocks of data; CBC mode for larger amounts of data.
- **MAC Generation/Verification**: implemented as a CMAC using the AES-128.
- **Key Derivation**: using the Miyaguchi-Preneel (MP) compression algorithm with the AES as block cipher.

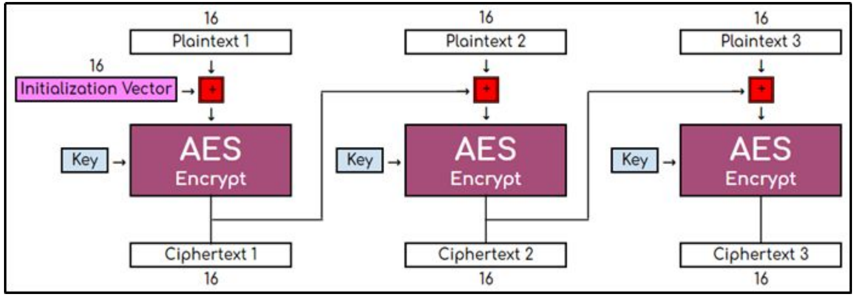
Algorithms in SHE - Enc/Dec

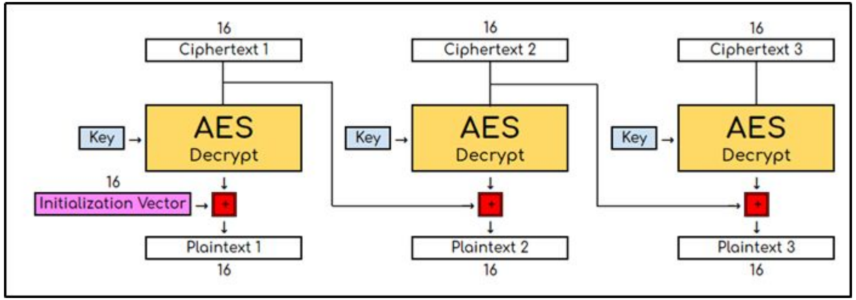
- ECB mode for single blocks of data





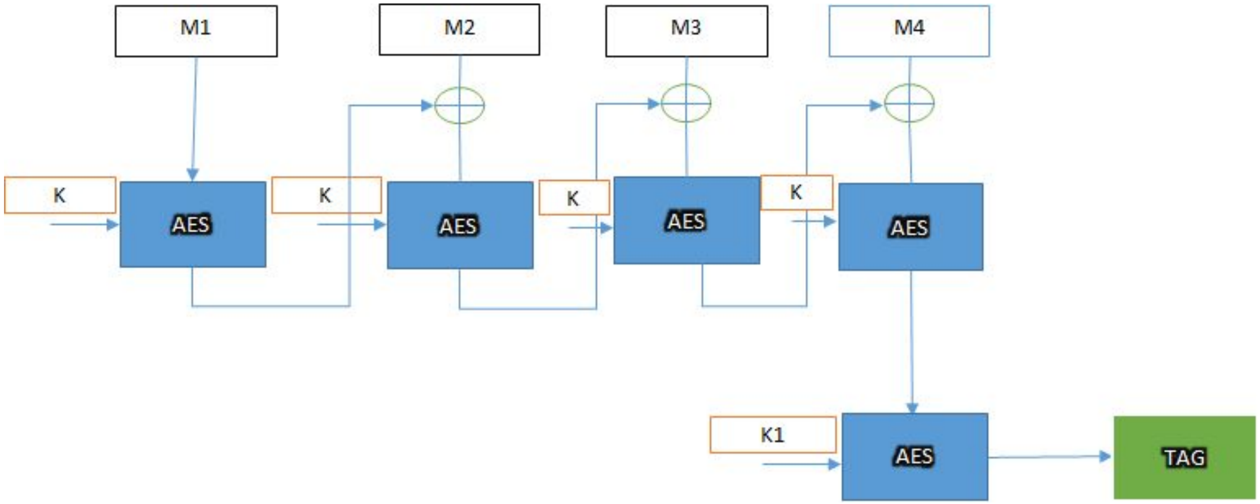
- CBC mode for larger amounts of data.





Algorithms in SHE - MAC

- Implemented as a CMAC using the AES-128.
- CMAC is usually refers the AES based CBC-MAC.



Algorithms in SHE - Key Derivation

- Using the Miyaguchi-Preneel (MP) compression algorithm with the AES as block cipher.

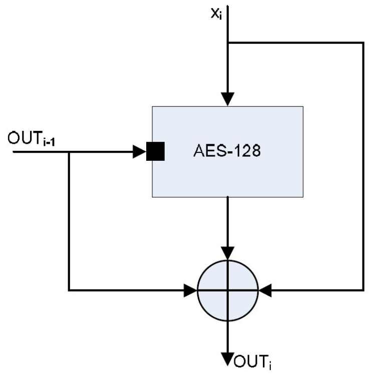
$\mathrm{KDF(K,\ C):\ AES\text{-}MP(K}\ |\ \boxed{\mathrm{C}})$

$\mathrm{AES\text{-}MP(x_i):\ OUT_i = ENC_{ECB,OUT_{i-1}}(x_i)\ \oplus x_i \oplus OUT_{i-1};\ i > 0;\ OUT_0 = 0;}$

C is the constant value concatenated with the key value

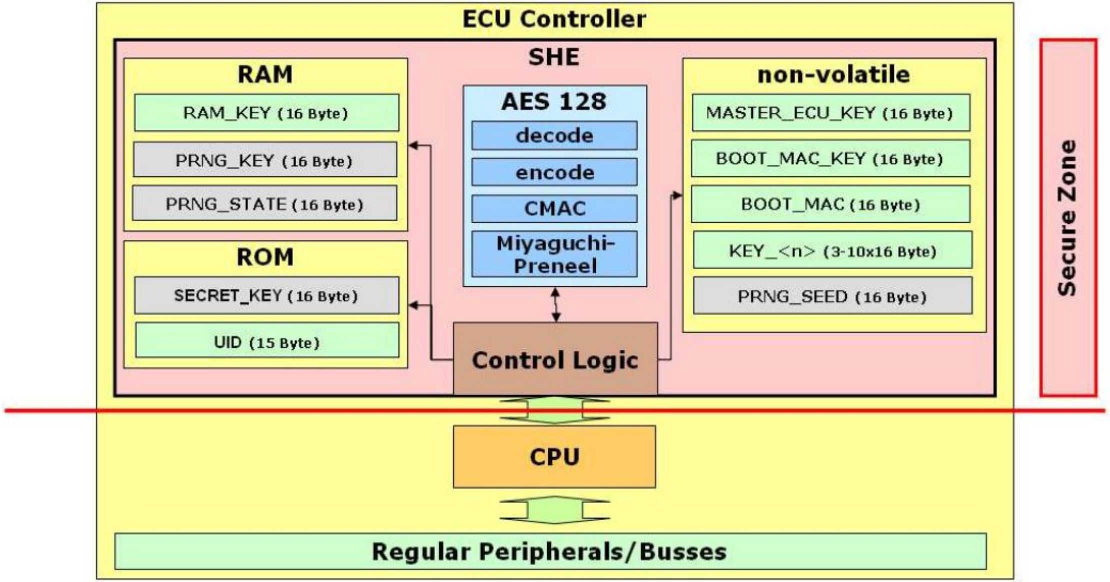
Constant	Value
KEY_UPDATE_ENC_C	0x01015348 45008000 00000000 000000B0
KEY_UPDATE_MAC_C	0x01025348 45008000 00000000 000000B0
DEBUG_KEY_C	0x01035348 45008000 00000000 000000B0
PRNG_KEY_C	0x01045348 45008000 00000000 000000B0
PRNG_SEED_KEY_C	0x01055348 45008000 00000000 000000B0
PRNG_EXTENSION_C	0x80000000 00000000 00000000 00000100

Constant values used within SHE



Miyaguchi-Preneel one-way compression function

Data Storage of SHE



Detailed logical structure of SHE

Key name	Address (hexadecimal)	Memory area
SECRET_KEY	0x0	ROM
MASTER_ECU_KEY	0x1	non-volatile
BOOT_MAC_KEY	0x2	
BOOT_MAC	0x3	
KEY_1	0x4	
KEY_2	0x5	
KEY_3	0x6	
KEY_4	0x7	
KEY_5	0x8	
KEY_6	0x9	
KEY_7	0xa	
KEY_8	0xb	
KEY_9	0xc	
KEY_10	0xd	
RAM_KEY	0xe	volatile

Key addresses

Pseudo Random Number Generation

1. PRNG key load

`PRNG_KEY = KDF(SECRET_KEY, PRNG_KEY_C)`

2. Seed generation

`PRNG_SEED_KEY = KDF(SECRET_KEY, PRNG_SEED_KEY_C)`

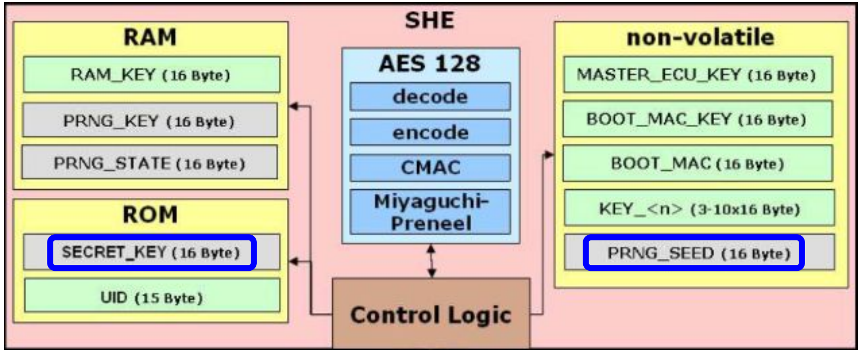
$\mathrm{PRNG\_SEED}_i = \mathrm{ENC}_{\mathrm{ECB},\mathrm{PRNG\_SEED\_KEY}}(\mathrm{PRNG\_SEED}_{i-1})$

The updated PRNG_SEED will copy to PRNG_STATE as the initial state.

3. Random generation

$\mathrm{PRNG\_STATE}_i = \mathrm{ENC}_{\mathrm{ECB},\mathrm{PRNG\_KEY}}(\mathrm{PRNG\_STATE}_{i-1})$

$\mathrm{RND} = \mathrm{PRNG\_STATE}_i$



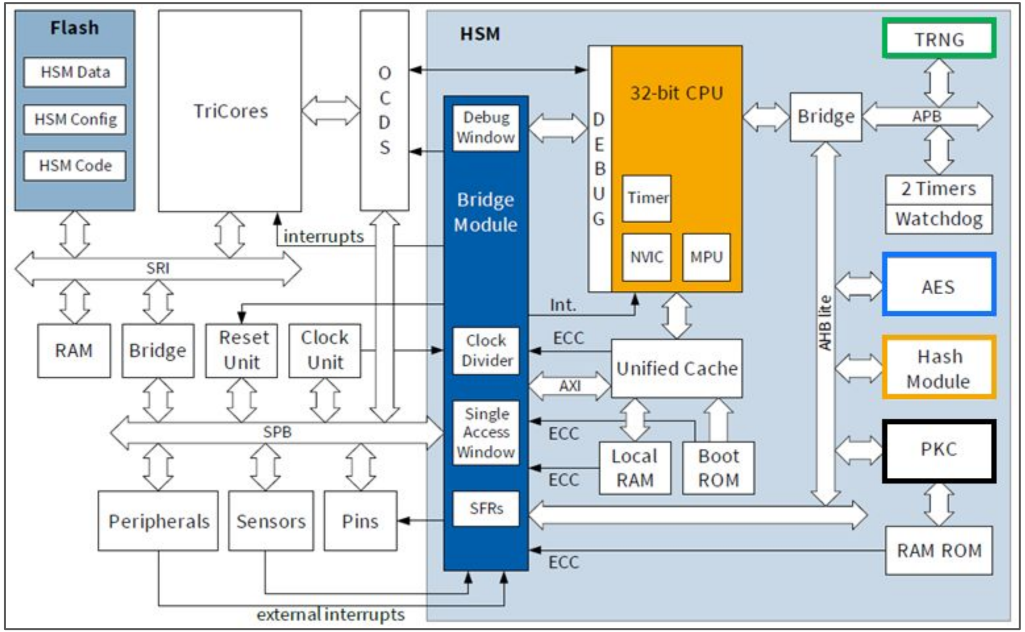
SHE Data Storage

Constant	Value
KEY_UPDATE_ENC_C	0x01015348 45008000 00000000 000000B0
KEY_UPDATE_MAC_C	0x01025348 45008000 00000000 000000B0
DEBUG_KEY_C	0x01035348 45008000 00000000 000000B0
PRNG_KEY_C	0x01045348 45008000 00000000 000000B0
PRNG_SEED_KEY_C	0x01055348 45008000 00000000 000000B0
PRNG_EXTENSION_C	0x80000000 00000000 00000000 00000100

Constant values used within SHE

Infineon AURIX™ HSM

- Infineon HSM is an extensive implementation of the SHE specification.
- SHE can be implemented in several ways, the SHE specification is intended to provide a detailed description of a technical realization.
- SHE behavior can be emulated on an HSM.



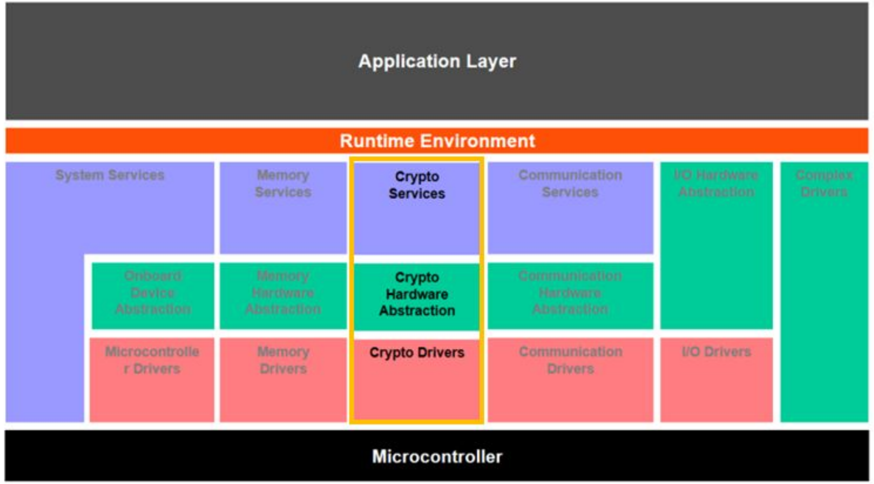
1 – True Random Number Generator

2 – AES 128 bit Enc/Dec Device

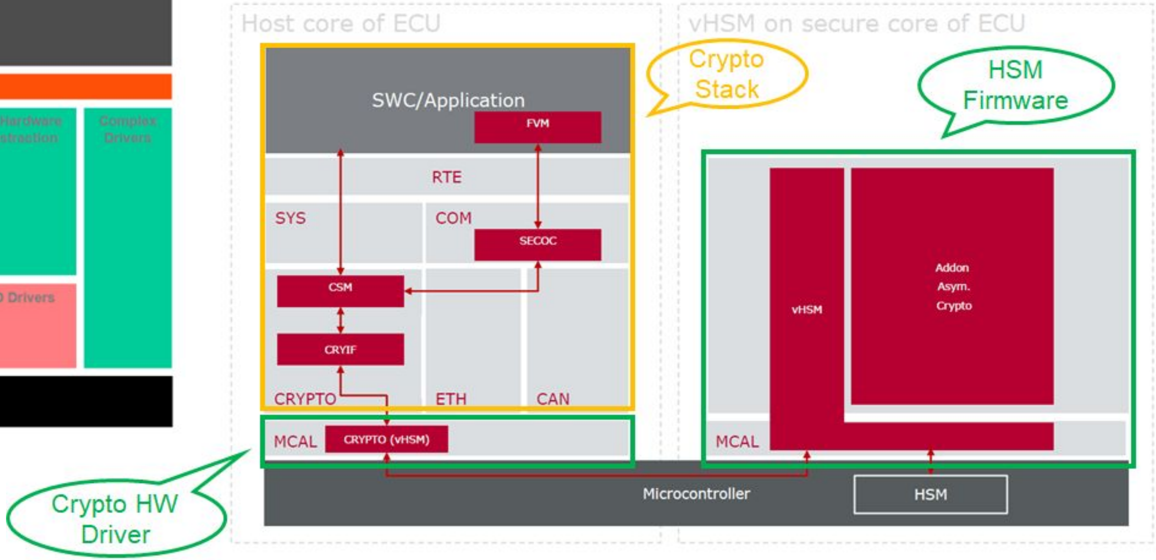
3 – Hash Module

4 – Public Key Cryptography Module

AUTOSAR based Software for HSM



AUTOSAR Layered View with CSM



Interaction of HSM (Secure Core) with AUTOSAR (Host Core)

User-accessible Functions

Encryption: CMD_ENC_ECB
Encryption: CMD_ENC_CBC
Decryption: CMD_DEC_ECB
Decryption: CMD_DEC_CBC
MAC generation: CMD_GENERATE_MAC
MAC verification: CMD_VERIFY_MAC
Secure key update: CMD_LOAD_KEY
Plain key update: CMD_LOAD_PLAIN_KEY
Export key: CMD_EXPORT_RAM_KEY
Initialize random number generator: CMD_INIT_RNG
Extend the PRNG seed: CMD_EXTEND_SEED
Generate random number: CMD_RND
Bootloader verification (secure booting): CMD_SECURE_BOOT
Impose sanctions during invalid boot: CMD_BOOT_FAILURE
Finish boot verification: CMD_BOOT_OK
Read status of SHE: CMD_GET_STATUS
Get identity: CMD_GET_ID
Cancel function: CMD_CANCEL
Debugger activation: CMD_DEBUG

- SHE provides several functions to the main CPU.
- In general, only a single function can be executed at a given time.
- Only the commands CMD_GET_STATUS and CMD_CANCEL may be called while another function is processed.

Contents

Direct Memory Update - Simple but Bad Idea

1. Target **Key Address**

   Eg. 0x4 for KEY_1

2. New **Key Value**

   Eg. 128 bit key value

Backend

Update Memory

Key name	Address (hexadecimal)	Memory area
SECRET_KEY	0x0	ROM
MASTER_ECU_KEY	0x1	
BOOT_MAC_KEY	0x2	
BOOT_MAC	0x3	
KEY_1	0x4	
KEY_2	0x5	
KEY_3	0x6	
KEY_4	0x7	non-volatile
KEY_5	0x8	
KEY_6	0x9	
KEY_7	0xa	
KEY_8	0xb	
KEY_9	0xc	
KEY_10	0xd	
RAM_KEY	0xe	volatile

Memory Slot

Secure Memory Update

- To update a memory slot the backend must have knowledge of a valid **authentication secret**.

What authentication secret the backend should have???

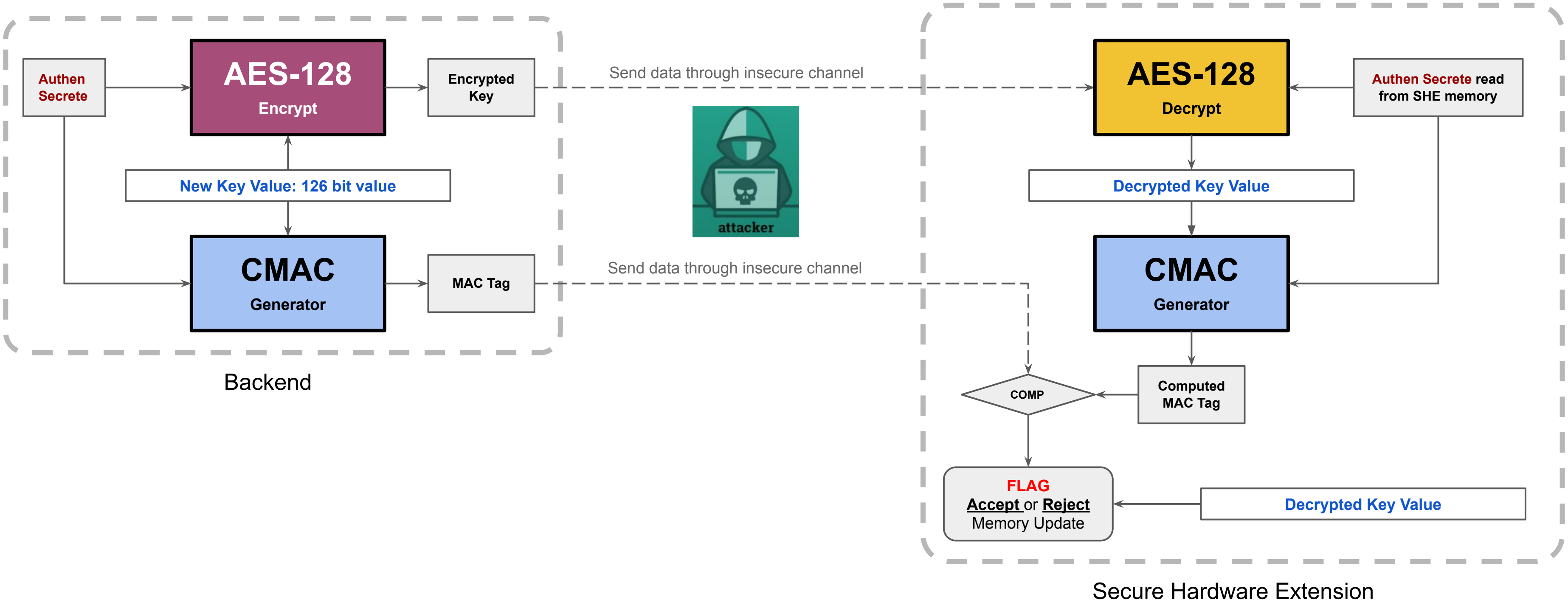
1. The value of **MASTER_ECU_KEY**;

OR

2. **The current key value** in the memory slot which the backend wants to update.

> Either secret the backend has is identified by **AuthID**.

Secure Memory Update - Basic Workflow



Secure Memory Update Messages

Key name	Address (hexadecimal)	Memory area
SECRET_KEY	0x0	ROM
MASTER_ECU_KEY	0x1	
BOOT_MAC_KEY	0x2	
BOOT_MAC	0x3	
KEY_1	0x4	
KEY_2	0x5	
KEY_3	0x6	
KEY_4	0x7	non-volatile
KEY_5	0x8	
KEY_6	0x9	
KEY_7	0xa	
KEY_8	0xb	
KEY_9	0xc	
KEY_10	0xd	
RAM_KEY	0xe	volatile

M1	120 bits UID	4 bits UpdateKeyID	4 bits AuthID

- UID: is the unique identification of SHE from factory.
- UpdateKeyID: is the identification number of **key we want to update** - the logical address of the SHE memory slot.
- AuthID: is the identification number of **Authen Secret** - the logical address of the SHE memory slot.

M2`	28 bits Counter	5 bits Flags	'0...0' 95 zeros	128 bits New Key Value

M2	AES128-ENC-CBC(**M2`**) using **Authen Secret K_1**

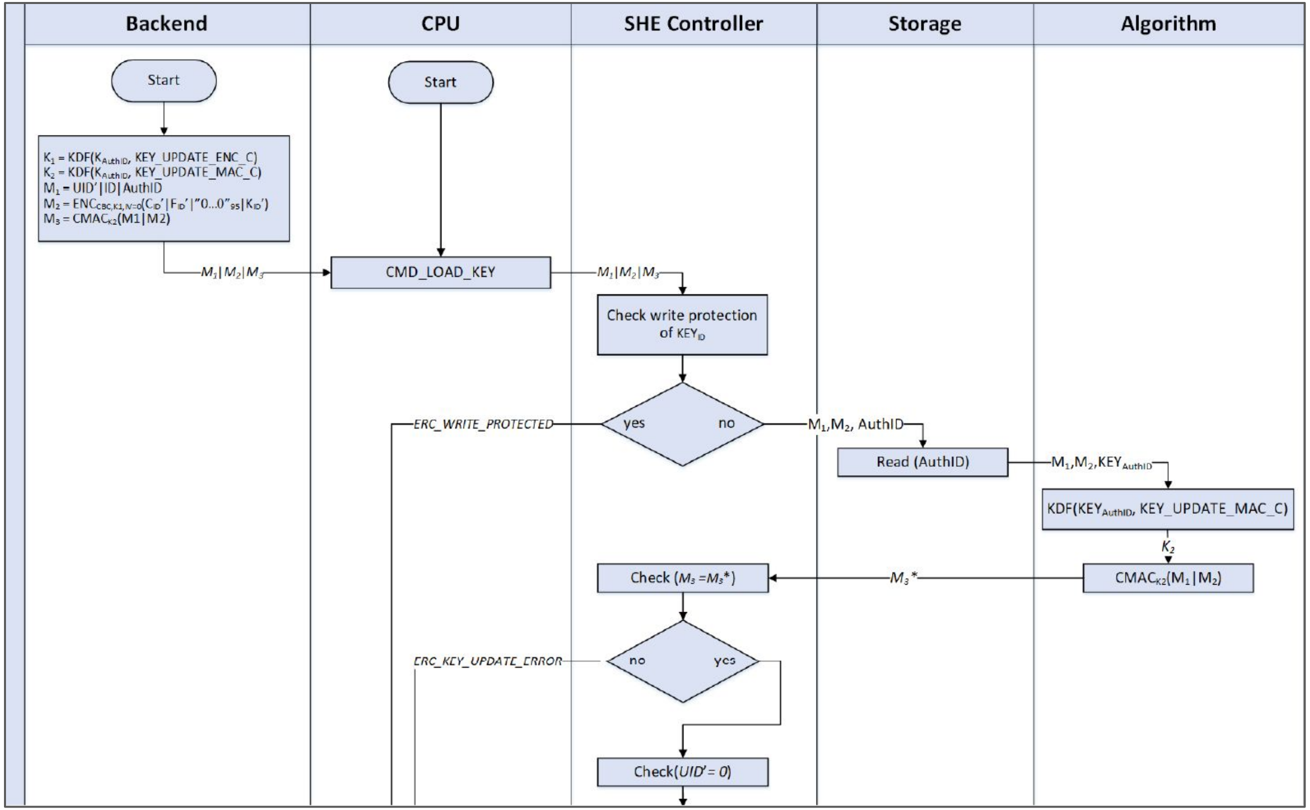
- Counter: each update the counter will increment 1.
- Flags: protection flags, each bit indicates a protected feature of the key.
  > WRITE | BOOT | DEBUGGER | KEY_USAGE | WILDCARD (5 bits)
- Zeros: just used for padding the message to 128 bits.

M3	AES128-CMAC(**M1** | **M2**) using **Authen Secret K_2**

K_{AuthID} is the key value at AuthID, currently stored in the SHE memory slot.
KDF is key derivation function to get the authen secret keys.

> **K_1** = KDF(K_{AuthID}, KEY_UPDATE_ENC_C)
> **K_2** = KDF(K_{AuthID}, KEY_UPDATE_MAC_C)

SHE Memory Update Protocol

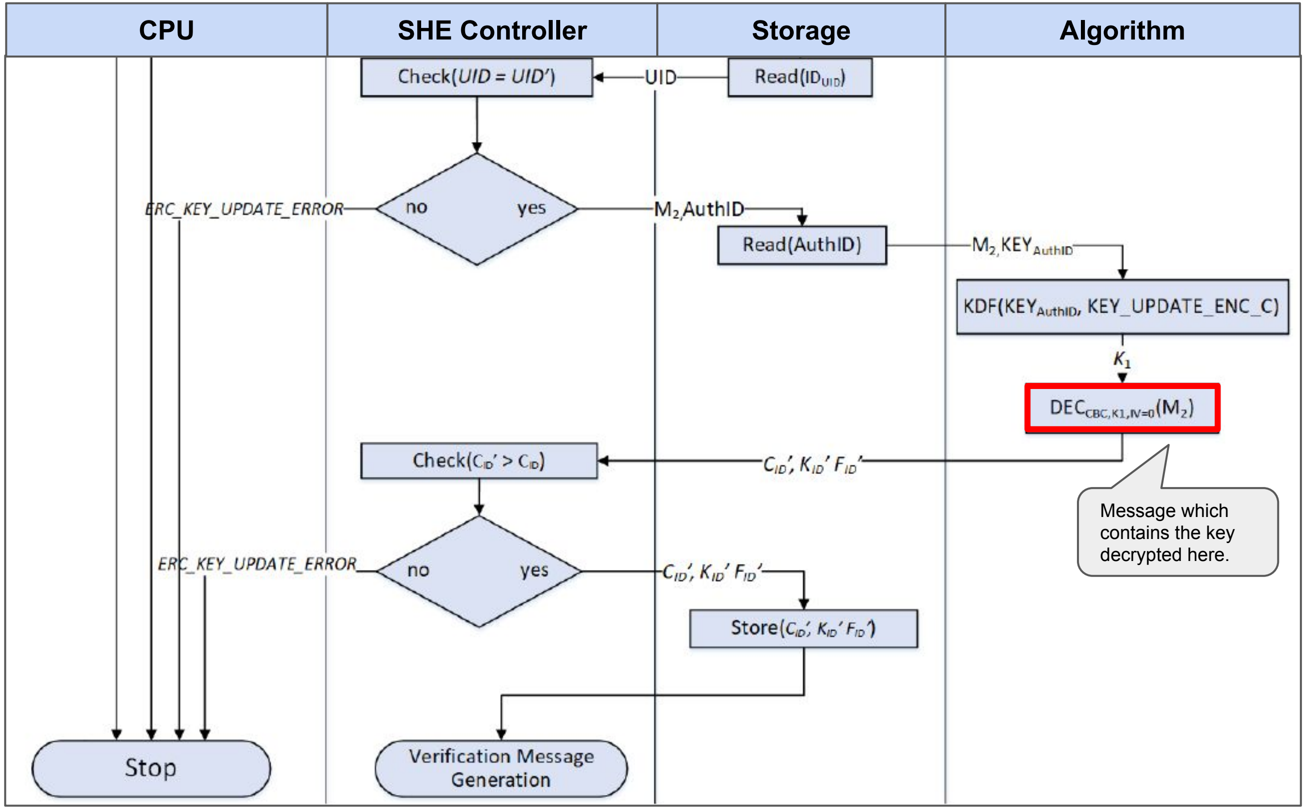


Key name	Address (hexadecimal)	Memory area
SECRET_KEY	0x0	ROM
MASTER_ECU_KEY	0x1	AuthID
BOOT_MAC_KEY	0x2	
BOOT_MAC	0x3	
KEY_1	0x4	ID
KEY_2	0x5	
KEY_3	0x6	
KEY_4	0x7	non-volatile
KEY_5	0x8	
KEY_6	0x9	
KEY_7	0xa	
KEY_8	0xb	
KEY_9	0xc	
KEY_10	0xd	
RAM_KEY	0xe	volatile

Example:
The authen secret is **MASTER_ECU_KEY**;
The target update key slot is **KEY_1**.

(Cont. next page)

SHE Memory Update Protocol (Cont.)



Key name	Address (hexadecimal)	Memory area
SECRET_KEY	0x0	ROM
MASTER_ECU_KEY	0x1	AuthID
BOOT_MAC_KEY	0x2	
BOOT_MAC	0x3	
KEY_1	0x4	ID
KEY_2	0x5	
KEY_3	0x6	
KEY_4	0x7	non-volatile
KEY_5	0x8	
KEY_6	0x9	
KEY_7	0xa	
KEY_8	0xb	
KEY_9	0xc	
KEY_10	0xd	
RAM_KEY	0xe	volatile

Example:
The authen secret is **MASTER_ECU_KEY**;
The target update key slot is **KEY_1**.

Memory Update Verification Messages

M4` | 28 bits Counter | Padding: ‘1’ 1 one and ‘0...0’ 99 zeros

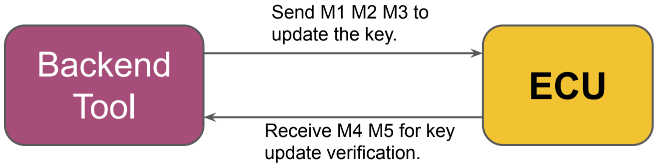
M4* | AES128-ENC-EBC(**M4`**) using **Verification Secret K_3**

M4 | 120 bits UID | 4 bits UpdateKeyID | 4 bits AuthID | 128 bits **M4***

- Padding in M4` = 0x8000000000000000000000000 (1 and 99 0’s), which is a fixed value.
- Basically, the first 128 bits of the M4 is the same as M1.
- The second 128 bits of the M4 is called M4*, which is an EBC encryption using **K_3** over M4`.

M5 | AES128-CMAC(**M4**) using **Verification Secret K_4**

- Finally M5 is generated by calculating a CMAC over the message M4 with **K_4**.
- The messages of **M4** and **M5** are then transferred to the backend.



K_{ID} is the new key value just updated to the SHE memory slot.
KDF is key derivation function to get the verification secret keys.

> **K_3** = KDF(K_{ID}, KEY_UPDATE_ENC_C)
> **K_4** = KDF(K_{ID}, KEY_UPDATE_MAC_C)

References

[1] Handbook of Applied Cryptography

http://cacr.uwaterloo.ca/hac/

[2] AUTOSAR specification of secure hardware extensions

https://www.autosar.org/fileadmin/user_upload/standards/foundation/19-11/AUTOSAR_TR_SecureHardwareExtensions.pdf

[3] Infineon Technologies AURIX™ Security Solutions

https://www.infineon.com/cms/en/product/microcontroller/32-bit-tricore-microcontroller/aurix-security-solutions/

[4] Vector solutions for automotive cybersecurity

https://www.vector.com/int/en/know-how/technologies/safety-security/automotive-cybersecurity/

Thank You

Any questions?