AUTOSAR CRYPTO STACK

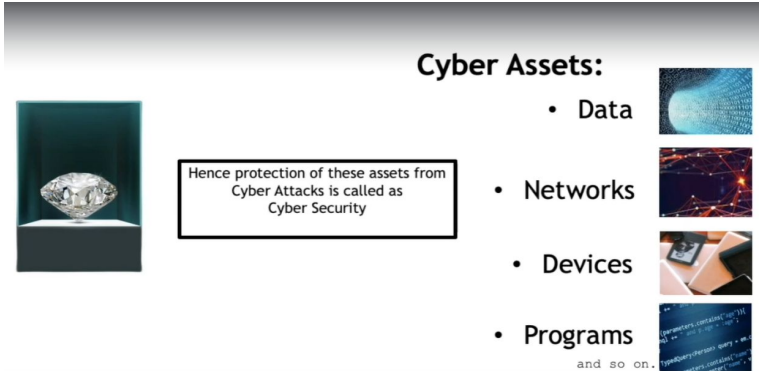
- Nikhil Ravikumar

Content

- What is Cyber Asset?
- What is Cyber Security?
- Technologies that pose cyber security threat
- Probable Vulnerabilities
- The industry’s most common security activities
- Three aspects of secured communication ?
- Cryptography Foundation
- Autosar Crypto Stack Overview
- Terms and descriptions used in this section
- AUTOSAR Crypto Stack Modules
- Software Vs Hardware Crypto Drivers
- SHE Vs HSM
- Cryptographic Job Management and Key Handling
- Streaming Approach vs Single Call Approach
- AUTOSAR Crypto Stack Configuration
- AUTOSAR Crypto Stack Operation Flow
- OpenSSL Useful Commands
- Security Concepts Using AUTOSAR Crypto Stack

What is Cyber Asset?

- In the automotive industry, cyber assets are the digital and electronic components, including hardware, software, and data, crucial for vehicle operation and safety that need protection against cyber threats.
  - Networks (CAN, LIN, Ethernet)
  - Devices (ECU,HSM)
  - Programs(Firmware)
  - Data (vehicle configuration, user settings)



What is Cyber Security?

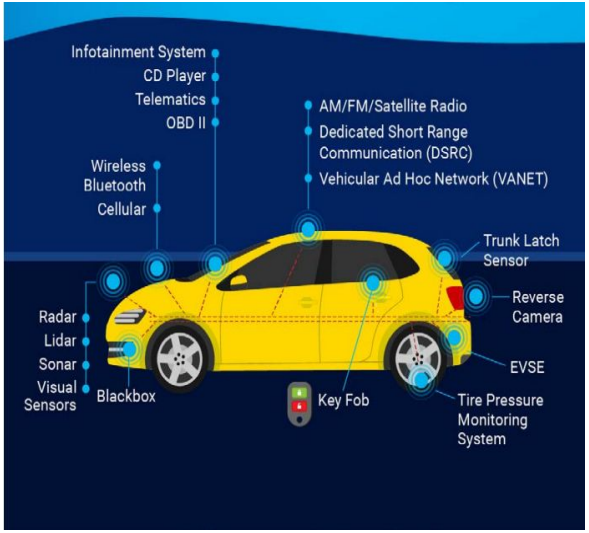
- Cybersecurity for cyber assets involves implementing measures and practices to protect digital and electronic resources from unauthorized access, attacks, or damage, ensuring their confidentiality, integrity, and availability.
- **Threats**
  - Threats refer to potential events or actions that could cause harm to systems, networks, or data. They represent the possibility of a malicious attack or an accidental event that could compromise security.
    - e.g.: Malware

What is Cyber Security?

- **Vulnerabilities**
  - Vulnerabilities are weaknesses or flaws in a system that can be exploited by threats. They are points of exposure that can be targeted.
    - Eg:-Weak Passwords
- **Risks**
  - Risks refer to the potential for loss, damage, or destruction of assets as a result of a threat exploiting a vulnerability.

Technologies that pose cyber security threat

- Connected Vehicles
- Telematics
- Keyless car theft
- EV charging station exploitation
- Infotainment systems
- Self-driving (autonomous) vehicles
- Software-focused service provider
  - (e.g. cloud, insurance provider, streaming service)
- Steering systems
- Electrification components
- Cameras
- RF technologies
  - (e.g. Wi-Fi, Bluetooth, Hot spots)
- FOTA Updates

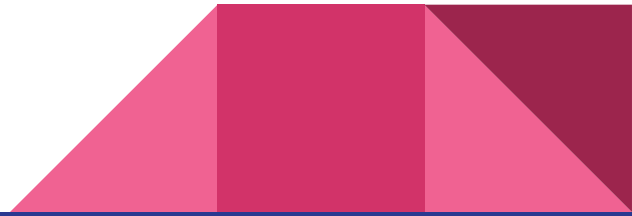


Probable Vulnerabilities

- Accidental coding errors
- The use of insecure/outdated open source software components
- Malicious code injection
- Lack of internal policies or rules that clarify security requirements
- Lack of understanding/training on secure coding practices
- Pressure to meet project deadlines
- Lack of quality assurance and testing procedures
- Product development tools have inherent bugs
- Incorrect permissions
- Back end systems

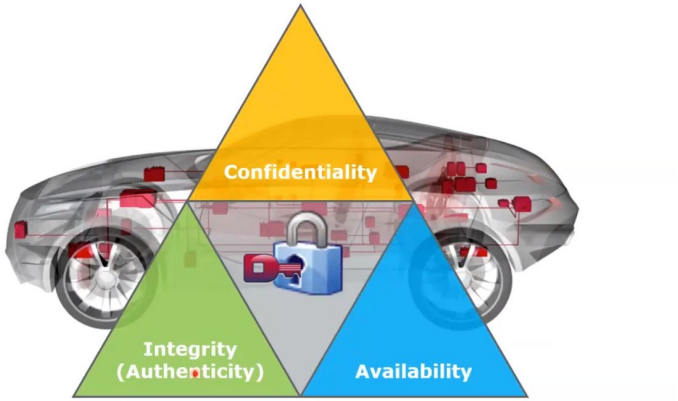
The industry’s most common security activities

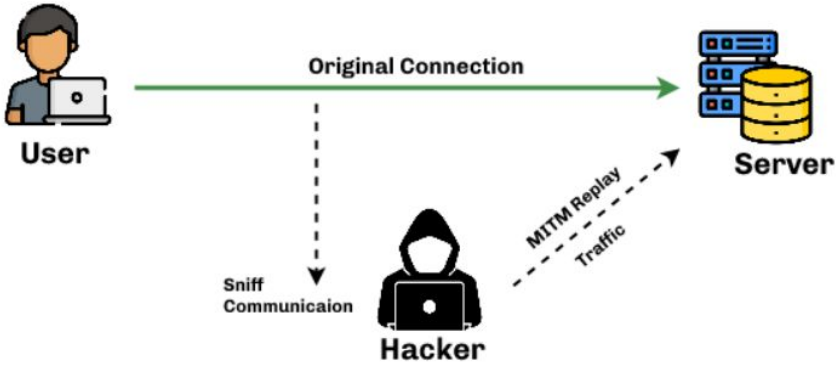
- Security patch management
- Penetration testing
- Dynamic security testing/DAST
- System debugging
- Static analysis/SAST (automated)
- Security requirements definitions
- data (during testing)
- Educate developers on secure coding methods
- Code review (manual)
- Run-time application self-protection
- Threat modeling
- Fuzz testing
- Software composition analysis
- Secure architecture design
- Identification method



Three aspects of secured communication ?

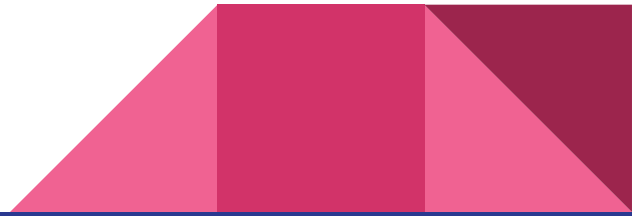
- **Confidentiality** (Data is protected from unauthorized viewing and access/ asset is kept secret)
- **Integrity** (Consistency, Accuracy & Trustworthiness of data over its lifecycle period)
- **Availability** (Information should be readily and consistently available over its lifecycle)





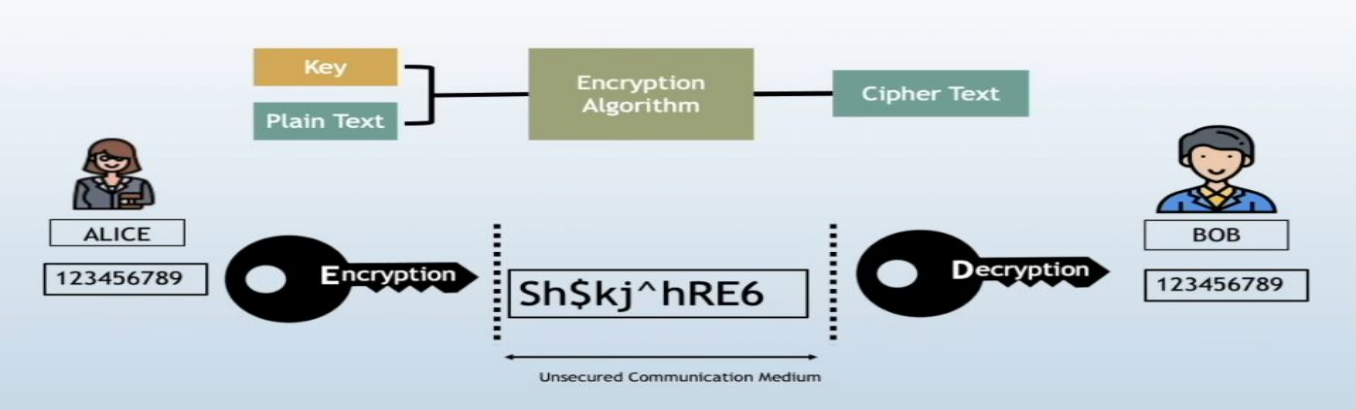
Cryptography Foundation

- Encryption
- Hashing
- Hashing Vs Encryption
- Digital Signature
- Signature Vs Encryption

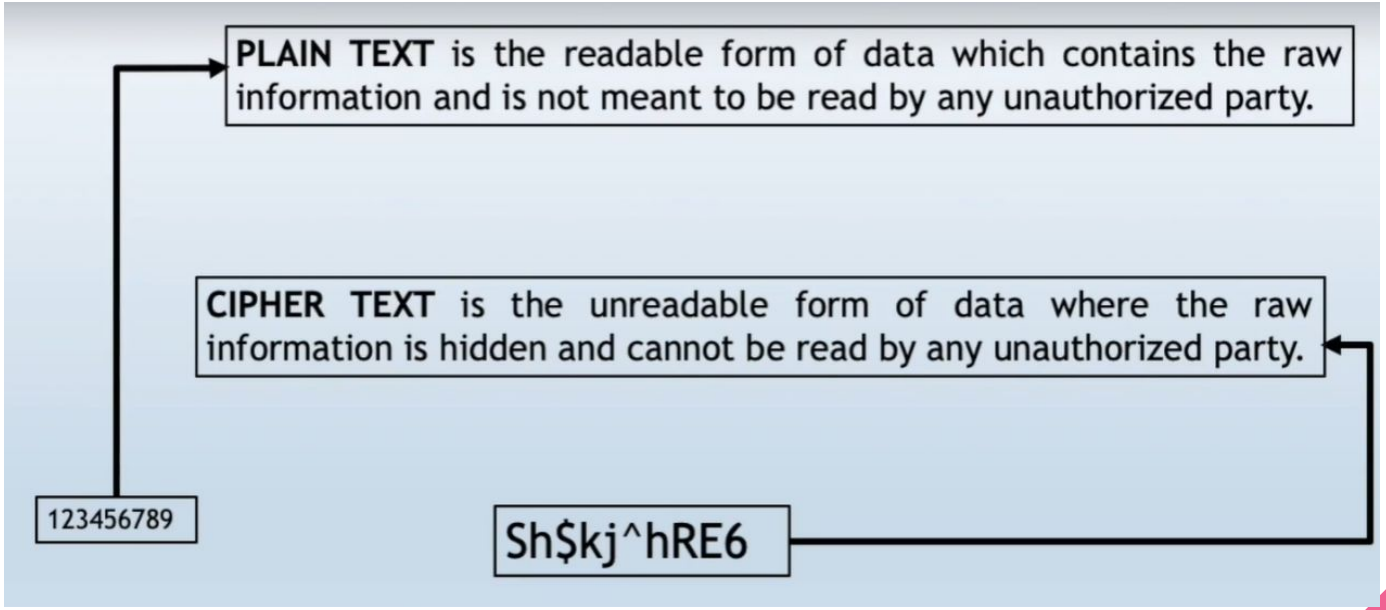


Encryption

- Encryption is the technique of securing digital informations with the help of some mathematical operations.
- Encryption converts the data into some unreadable form.

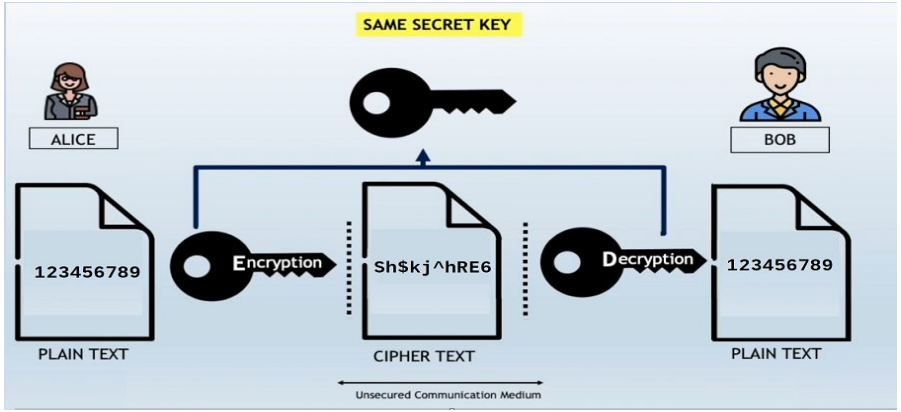


Difference between PLAIN TEXT and CIPHER TEXT

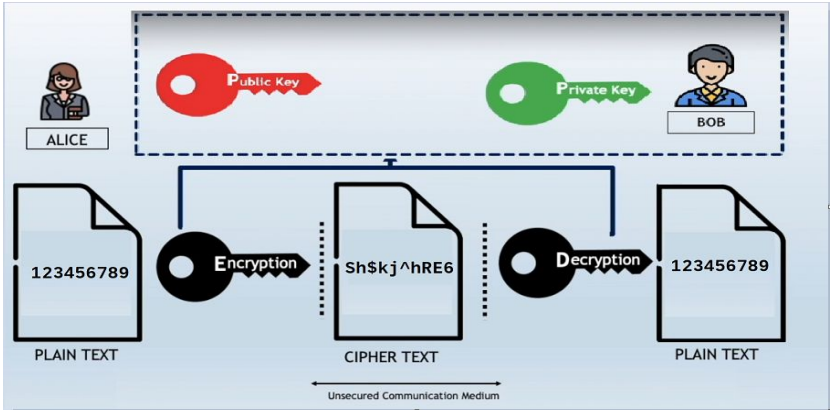


Different types of Encryption

- Symmetric Encryption



- Asymmetric Encryption



Symmetric Encryption

- Symmetric Encryption algorithm relies on a **single key** for encryption and decryption of information.
- Both the sender and receiver of the message need to have a pre-shared secret key that they will use to convert the plaintext into ciphertext and vice versa.
- The key which is being used for encrypting the plain text is decrypting the ciphertext.
- The key must be kept private and be known only to the sender and the receiver.

Challenges of using single Private Key

- If the same key is used across multiple sessions or for different types of data, an attacker who manages to compromise the key could potentially decrypt all data encrypted with it, leading to a widespread breach.
- When a key is reused, it becomes challenging to determine if and when a breach occurred. If a key is compromised, all data encrypted with that key is at risk, making it hard to isolate and address the issue effectively.

Asymmetric Encryption / Public Key Encryption

- Asymmetric Encryption algorithm relies on a **pair of keys** for encryption and decryption of information.
- The pair of keys includes a **public key**, which can be shared with anyone, and a **private key**, which is kept secret by the owner.
- Public Key is the key which is being used for encrypting the plain text into cipher text.
- Private Key is the key which is being used for decrypting the cipher text message into plain text.
- In asymmetric encryption, the sender uses the recipient's public key to encrypt the data.
- The recipient then uses its private key to decrypt the data.

Difference between Symmetric and Asymmetric Encryption

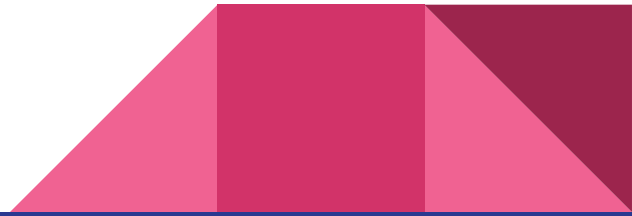
Symmetric Encryption	Asymmetric Encryption
It involves a single key for encryption and decryption	It involves two keys ,public key for encryption and private key for decryption
It consumes less CPU resources due to simpler algorithms	It consumes more CPU resources due to more complex algorithms
Less secure due to shared key	More secure due confidential private key

Symmetric Encryption Algorithms

- **AES (Advanced Encryption Standard)**
  - Key Sizes: 128, 192, or 256 bits.
  - Use Case: Secure Data Storage,Secure Boot
  - Type : AES-128 CMAC
- **DES (Data Encryption Standard)**
  - Key Size: 56 bits.
  - Use Case: Although outdated due to security vulnerabilities, DES may still be found in legacy systems.

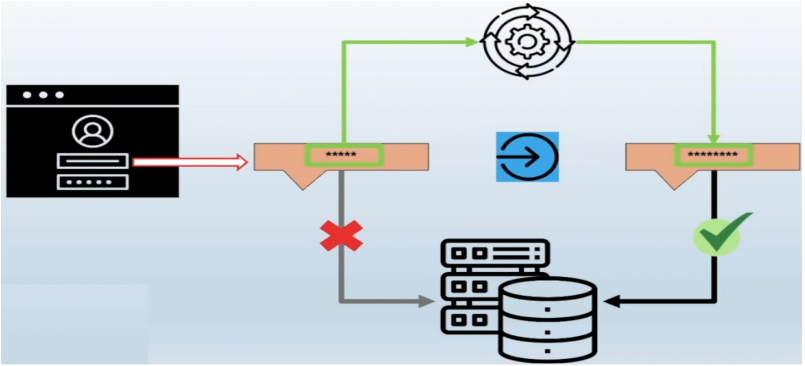
Asymmetric Encryption Algorithms

- **RSA (Rivest-Shamir-Adleman)**
  - Key Sizes: Commonly 2048 or 4096 bits.
  - Use Case: Secure Diagnostics, Secure Update.
- **ECC (Elliptic Curve Cryptography)**
  - Key Sizes: Generally smaller than RSA (e.g., 256 bits).
  - Use Case: Offers strong security with smaller keys, making it suitable for resource-constrained environments like ECUs.



Hashing

- Hashing is a cryptographic operation that converts plain text in to unique deterministic string of fixed length.
- Hashing is a one way process and for the same input to the hashing algorithm, the output would always be the same.
- It should be difficult to find two different inputs that produce the same hash output.
- Given a hash value, it should be infeasible to reverse-engineer the original input.



Hashing vs Encryption

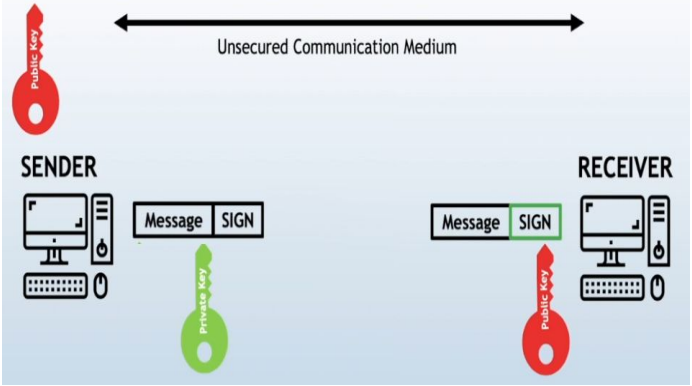
- Encryption is bidirectional whereas Hashing is Unidirectional.
- Encrypted data length generally depends on the payload length whereas Hashed data length has a fixed value.
- Encryptions is used for data confidentiality whereas Hashing is used primarily for data integrity and verification.
- Encryption typically uses key whereas Hashing typically does not use a key.
- Encryption is not designed for collision resistance whereas Hashing is designed to minimize the chances of different inputs producing the same hash value (collision).

Hashing Algorithms

- **SHA-256 (Secure Hash Algorithm 256-bit):**
  - A cryptographic hash function that produces a 256-bit hash value. It is part of the SHA-2 family and is widely used for its security and collision resistance.
  - Use Case: Secure Diagnostics, Secure Update.
- **SHA-1 (Secure Hash Algorithm 1):**
  - An older hash function that produces a 160-bit hash value. It is less secure compared to SHA-256 and is generally being phased out in favor of SHA-256.

Digital Signature

- Digital Signature is a verification method.
- Digital Signature is used to verify the authenticity of the sender and the integrity of the message sent.
- They work on the public key cryptography architecture typically, an asymmetric key system which encrypts with a public key and decrypts with a private key.
- Digital signatures do not provide confidentiality.
- The sender provides the Message and the Private key to a signing algorithm to create the signature.
- This signature along with the message is sent to the receiver.



How digital signing works?

- The **Receiver** uses the **Public key** of the **Sender** to **Verify** that the sender itself has sent this message and hence the authenticity of the message.
- This also verifies the integrity of the message since we have fed the message itself as an input to the signing algorithm.
- Calculation of signature on big data can be time consuming and hence the original message is first passed to a hash function to create a message digest/Hash.
- This message digest/Hash along with the Private key of the sender is fed to a signing algorithm to create the Signed Digest.
- This Signed Digest along with the message is sent to the receiver.
- Signed Digest is fed to the signature verification algorithm along with the Public key of the sender to get the message digest/Hash.
- The receiver uses the message to generate a message digest/Hash.
- It is then compared to the received message digest/Hash.

Importance of Digital Signature

- **Authenticity:**
  - Digital signatures verify that a message or transaction was created using the signer's private key.
- **Non-repudiation:**
  - They provide a record that prevents the signer from denying their signature on a document or transaction.
- **Security:**
  - Encryption algorithms safeguard the data from unauthorized access and tampering.

Signing Algorithms

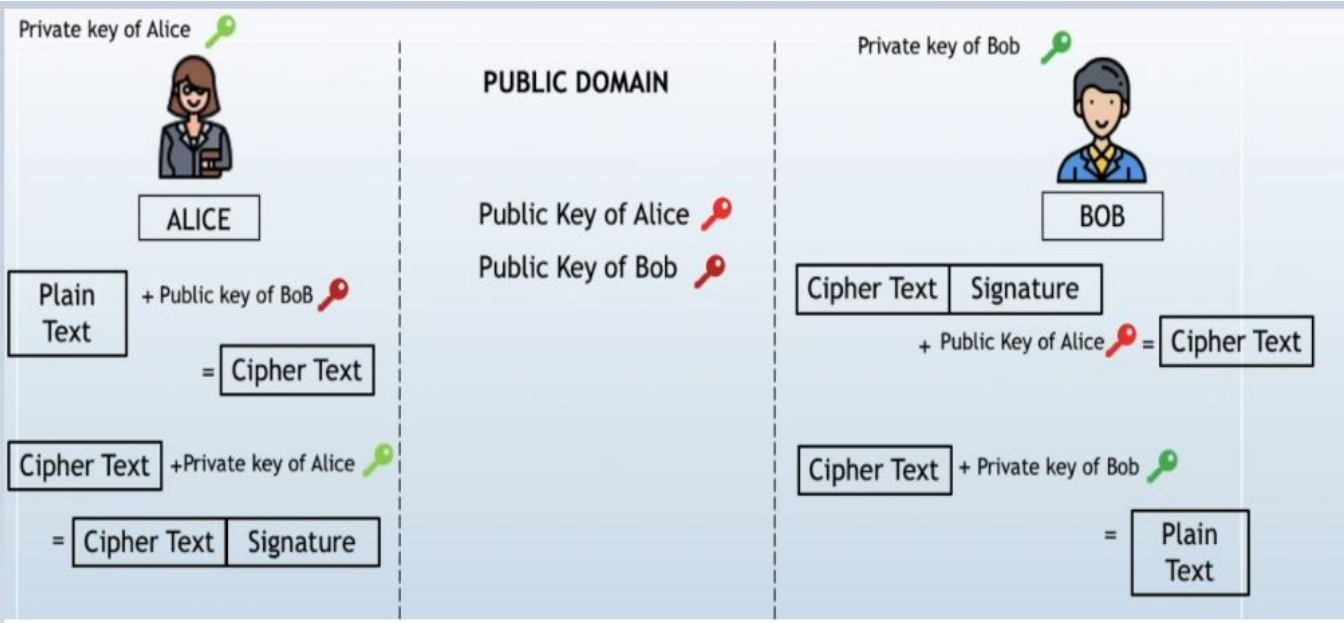
- **RSA (Rivest-Shamir-Adleman):**
  - An asymmetric encryption algorithm used for digital signatures and key exchange. It relies on the difficulty of factoring large prime numbers.
- **ECDSA (Elliptic Curve Digital Signature Algorithm):**
  - A variant of the Digital Signature Algorithm (DSA) that uses elliptic curve cryptography to provide secure signatures with shorter key lengths compared to RSA.
- **EdDSA (Edwards-Curve Digital Signature Algorithm)**:
  - A modern and efficient signature scheme based on elliptic curves.

Signature Vs Encryption

Signature	Encryption
Signature ensures message integrity & authentication.	Encryption ensure Confidentiality
Sender signs the message using the sender's Private key.	Sender encrypts the message using the receiver's Public Key.
Receiver verifies the message using the sender's Public Key.	Receiver decrypts the message using the receiver's Private key.

	Sender Uses	Receiver Uses
Signing	Sender Private Key	Sender Public Key
Encryption	Receiver Public Key	Receiver Private Key

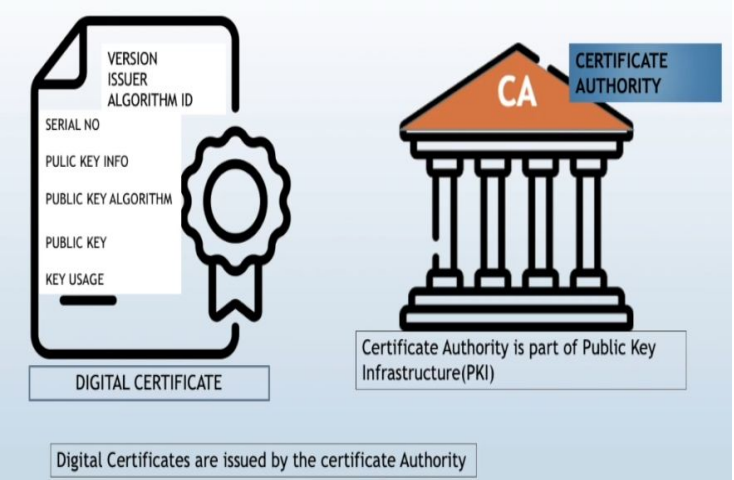
Combination Of Signature And Encryption



Back to TRIAD

What are Digital Certificates?

- A digital certificate is a file or electronic password that proves the authenticity of a device, server, or user through the use of cryptography and the public key infrastructure (PKI).
- Digital certificate authentication helps organizations ensure that only trusted devices and users can connect to their networks.
- These certificates include a public key and the identity information of the certificate holder, along with the CA's digital signature.
- Digital certificates are issued by a certificate authority(CA).



What is a Certificate Authority (CA)?

- CA is a trusted organization responsible for issuing, managing, and validating digital certificates.
- CAs issue digital certificates that authenticate the identity of entities.
- CAs validate the identity of the certificate requesters through **Registration authority(RA)** before issuing certificates to ensure the authenticity and integrity of the digital identity being established.
- CAs manage the lifecycle of certificates, including issuance, renewal, and revocation.
- CAs maintain Certificate Revocation Lists (CRLs) and Online Certificate Status Protocol (OCSP) services to keep track of certificates that are no longer valid.

What is PKI?

- **Public Key Infrastructure**
  - A technology for authenticating users and devices in the digital world.
  - The basic idea is to have one or more trusted parties digitally sign documents certifying that a particular cryptographic key belongs to a particular user or device.

What is an Entity?

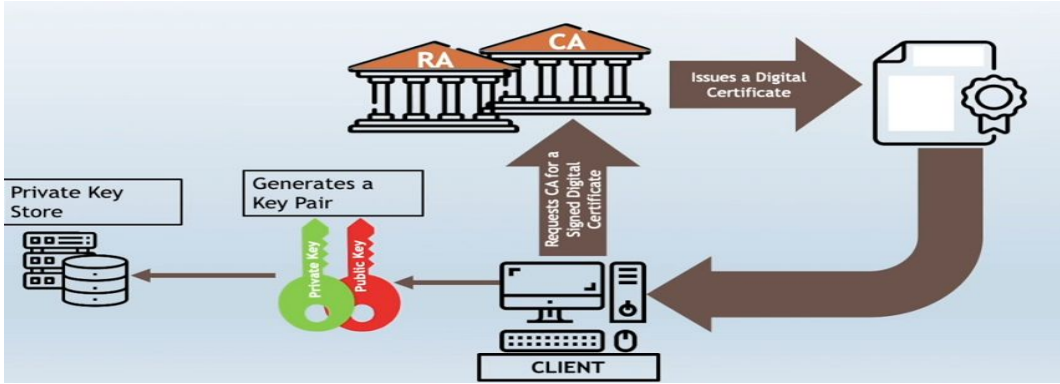
- The users and devices that have keys are often just called entities.
- Besides a user or device, it could be a program, process, manufacturer, component, or something else.

Generation And Structure of Digital Certificates

- Generate a key pair, which consists of a public key and a private key.
- Keep the private key secure and private.
- Share the public key with the CA as part of the certificate request.
- Create a Certificate Signing Request (CSR).
- The CSR is a block of encoded text that includes information about the entity requesting the certificate, such as its organization name, common name (e.g., domain name), locality, and the public key.
- Submit the CSR to the CA via web forms or emails.
- The CA will verify the information provided in the CSR through a Registration authority(RA).

Generation And Structure of Digital Certificates…

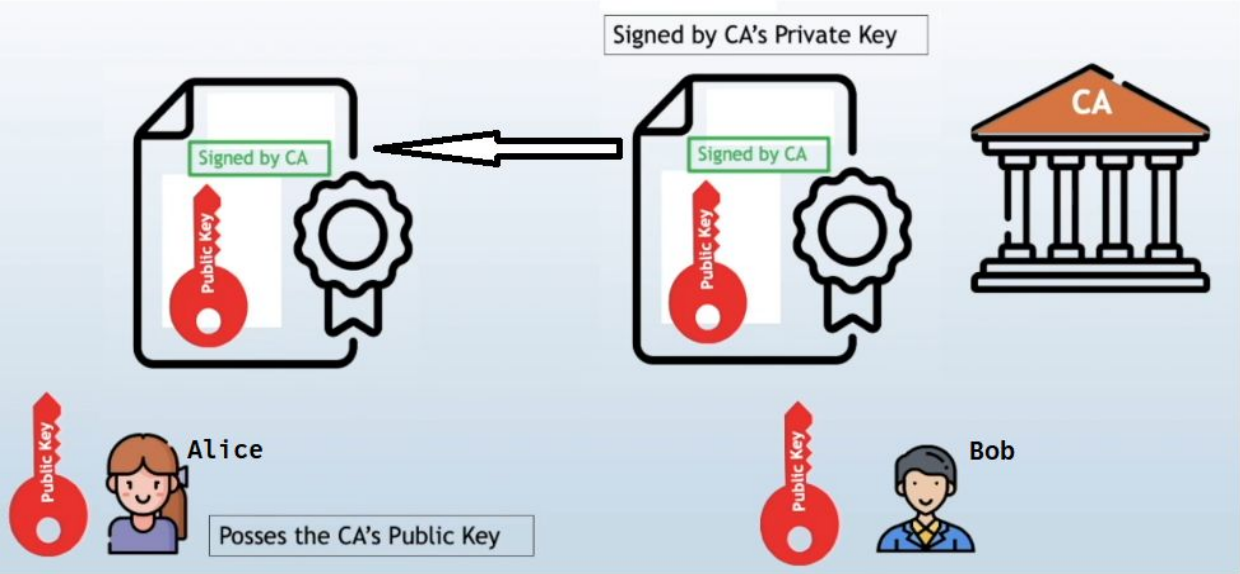
- Once the RA verifies the request and the details, CA will issue a digital certificate.
- This certificate includes your public key and the CA's digital signature, confirming its authenticity.
- The format of these certificates is specified by the X.509.
- X.509 is an International Telecommunication Union (ITU) standard defining the format of public key certificates.
-



Verification Of Digital Certificates

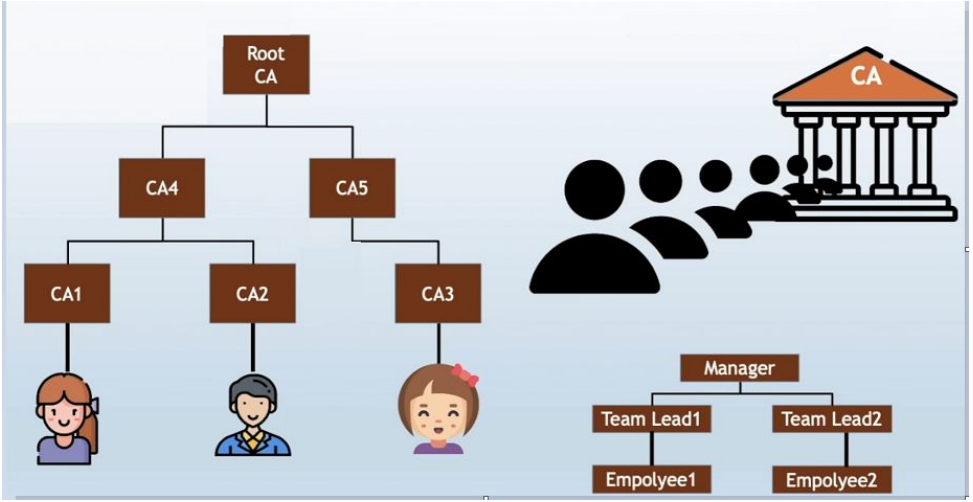
- Both Sender and Receiver should register with a Certificate Authority (CA) that they mutually trust.
- Sender sends the Digital certificate which has the Public key of the sender which is signed by the Certificate Authority.
- This Digital Certificate has to be verified by the receiver.
- The receiver who has already registered with the Certificate Authority already possess the Public Key of the Certificate Authority.
- The receiver then verifies the authenticity of the received Digital Certificate using the Public Key of the Certificate Authority.

- The receiver can then fetch the authenticated Public key from the Digital Certificate which is the sender's Public Key.
- The receiver can then use this Public Key to send encrypted messages to the sender.
-



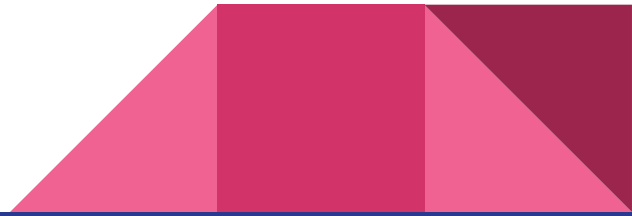
Hierarchy of CA

- CAs maintain a hierarchical structure.
  - Root certificates
  - Intermediate Certificates
  - End-User Certificates



Root certificates

- The root certificate is the top-most certificate in the chain of trust.
- Root Certificates are issued by Root Certificate Authority (Root CA).
- The root certificate is self-signed, meaning it is signed by its own private key.
- Root certificates are highly secure and their private keys are kept in very secure environments because they are critical to the entire trust framework.

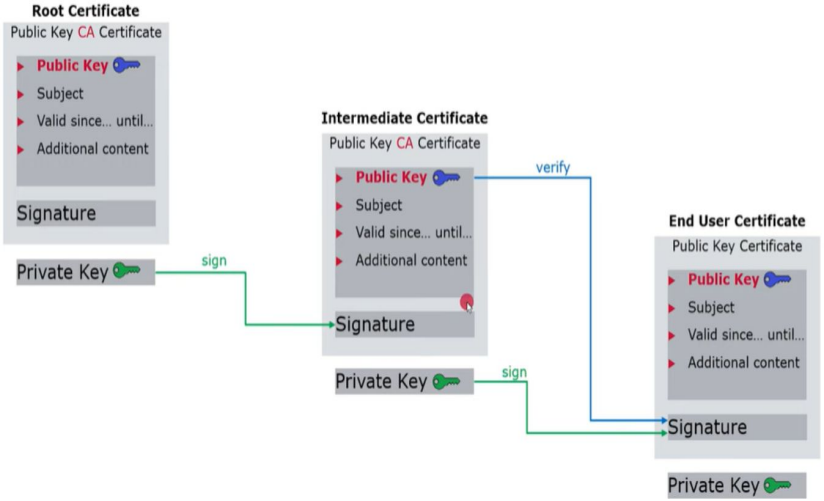


Intermediate Certificates

- Intermediate certificates are issued and signed by the root certificate or another intermediate certificate.
- Intermediate certificates helps to distribute the administrative load and enhances security by limiting the exposure of the root certificate.

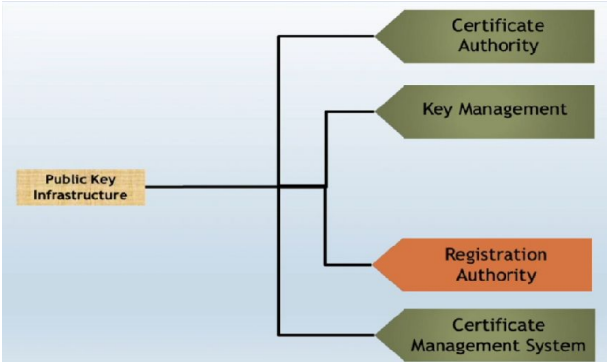
End-User Certificates

- End-user certificates are the certificates issued to individuals,organizations, or devices that need to prove their identity or secure communications.
- End-user certificates are issued by intermediate CAs in the chain, not directly by the root CA.
- Each end-user certificate includes the public key of the certificate holder along with identifying information and is signed by the issuing intermediate CA.



Anatomy of PKI

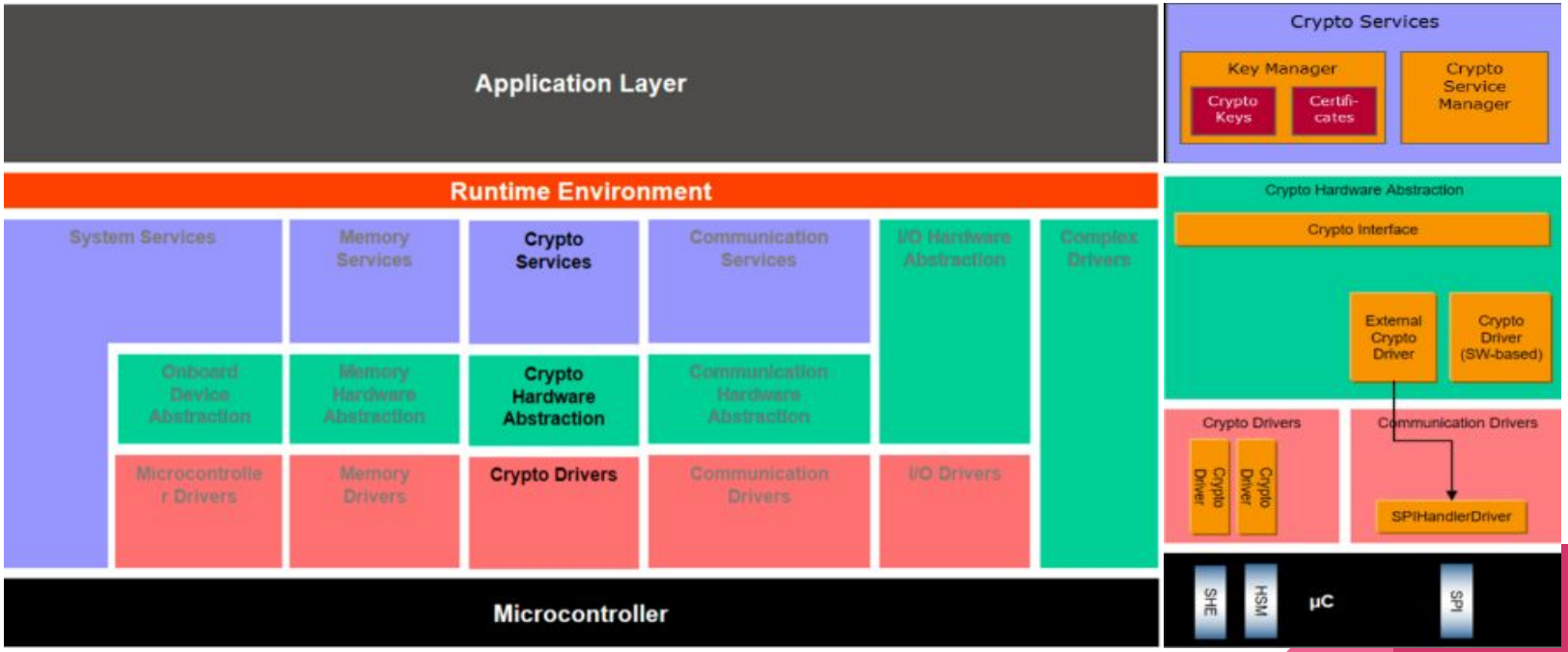
- **Certificate Authority**
  - Issuing certificates to the clients.
  - Provide services to clients to verify the Digital Certificates.
- **Key Management**
  - Key Generation.
  - Key Storage.
  - Key Destruction.
  - Key Archival.
- **Registration Authority**
  - Registration Authority checks the identity of the client requesting for the digital certificates.
- **Certificate Management System**
  - Maintains the infrastructure of the Digital Certificate.



Cyber Security Solution in Autosar Classic Platform

- **Increasing Connectivity:**
  - Modern vehicles use Cellular, WiFi, and Bluetooth, expanding their connectivity and potential attack surfaces.
- **Cybersecurity Risks:**
  - Hackers can exploit these connections to manipulate critical vehicle functions, risking fatal accidents and injuries.
- **Autosar Crypto Stack:**
  - Autosar Crypto stack provides essential cybersecurity protections, including hashing, asymmetric signature verification, and symmetric encryption, to safeguard against malicious attacks.

Autosar Crypto Stack Overview



Terms and descriptions used in this section

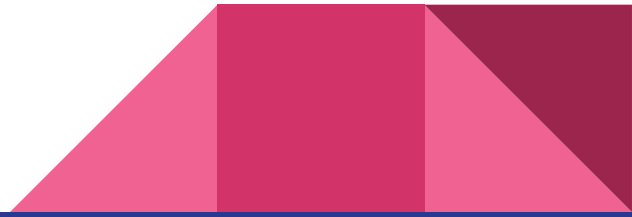
- **Job**:
  - A job is a configured 'CsmJob' which is an instance of configured cryptographic primitive, Keys and a reference channel.
- **Primitive**:
  - A primitive is an instance of a configured cryptographic algorithm realized in a Crypto Driver Object.
  - It refers to a functionality provided by the CSM to the application, the underlining 'algorithmfamily' (e.g. AES, MD5, RSA, etc.), and the 'algorithmmode' (e.g. ECB, CBC, etc).
- **Crypto Driver Object**:
  - A Crypto Driver Object is an instance of a crypto module (hardware or software), which is able to perform one or more different crypto operations.

- **Operation:**
  - An operation of a crypto primitive declares what part of the crypto primitive shall be performed.
  - There are three different operations:
    - **START:**
      - Operation indicates a new request of a crypto primitive, and it shall cancel all previous requests.
    - **UPDATE:**
      - Operation indicates, that the crypto primitive expect input data.
    - **FINISH:**
      - Operation indicates, that after this part all data are fed completely and the crypto primitive can finalize the calculations.
- **Channel:**
  - A channel is the path from a Crypto Service Manager (CSM) queue via the Crypto Interface (CryIf) to a specific Crypto Driver Object.
- **Key:**
  - A Key can be referenced by a job in the Csm.
  - In the Crypto Driver, the key references a specific key type.

- **Key Type:**
  - A key type consists of references to key elements.
  - The key types are typically pre-configured by the vendor of the Crypto Driver.
  - Further key types may be added by the user.
  - Preconfigured key types could be used as template by duplicating them.
- **Key Element:**
  - Key elements are used to store data.
  - This data can be e.g. key material or the IV needed for AES encryption.
  - It can also be used to configure the behavior of the key management functions like Persisting or Access rights.

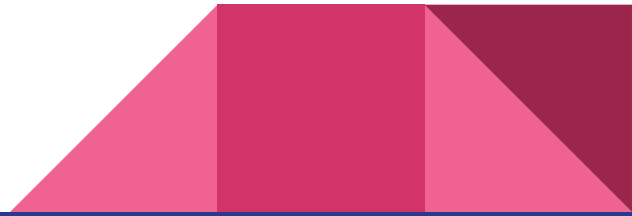
AUTOSAR Crypto Stack Modules

- The Crypto Service Manager (CSM)
- Key Manager (KeyM)
- The Crypto Interface (CRYIF)
- The Crypto Driver (CRYPTO)



The Crypto Service Manager (CSM)

- The CSM, as a service layer, provides the interface for Application Software Components (SWC) or BSW for cryptographic services.
- The CSM provides cryptographic services, based on a crypto driver which relies on a software library or on a hardware module.
- The CSM schedules and prioritizes services and calls the crypto interface (CryIf) for further operation.
- The CSM services are generic allowing different applications to use the same service with different cryptographic algorithms.

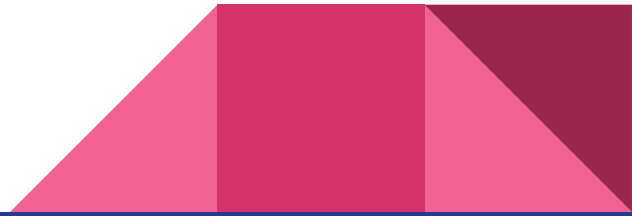


Key Manager (KeyM)

- keys and certificates management.
- Set/Update root and/or intermediate certificate
- Get certificate elements
- Get status of a certificate
- Verify certificate against certificate chain

The Crypto Interface (CRYIF)

- The CryIf is located between the CSM and the underlying crypto drivers.
- The CryIf provides a generic interface to Software Crypto and Hardware Crypto for all upper layers (BSW).
- The CryIf receives requests from the CSM and forwards them to the appropriate cryptographic operation in the Crypto Driver.
- The CryIf forwards the results given by Crypto Driver to the CSM.

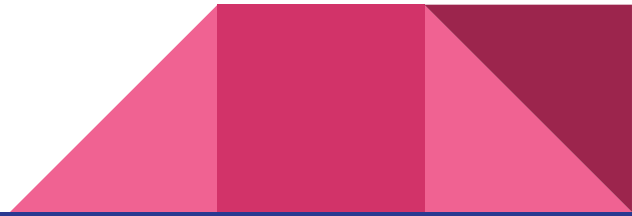


The Crypto Driver (CRYPTO)

- The Crypto Driver module is located in the microcontroller abstraction layer.
- The Crypto Driver implements a generic interface for synchronous and asynchronous cryptographic primitives.
- The Crypto Driver supports key storage, key configuration, and key management for Cryptographic services.
- The Crypto Driver can have one or more Crypto Driver Objects, with separate workspaces.
- Each Crypto Driver Object can provide arbitrary many Crypto Primitives.
- One Crypto Driver Object can only perform one Crypto Primitive at the same time.
- The Crypto Driver Object is the endpoint of a crypto channel.
- The Crypto Driver Object can be a hardware device or software.

Hardware accelerator

- These accelerators are specialized circuits or chips within the HSM specifically optimized for performing cryptographic functions more efficiently than general-purpose CPUs.
- They are not general-purpose processors but are tailored for high-speed, secure cryptographic operations.
- can perform multiple cryptographic operations simultaneously.
- By performing operations directly in hardware, these accelerators achieve lower latency and faster execution times for cryptographic tasks.



Software Vs Hardware Crypto Drivers

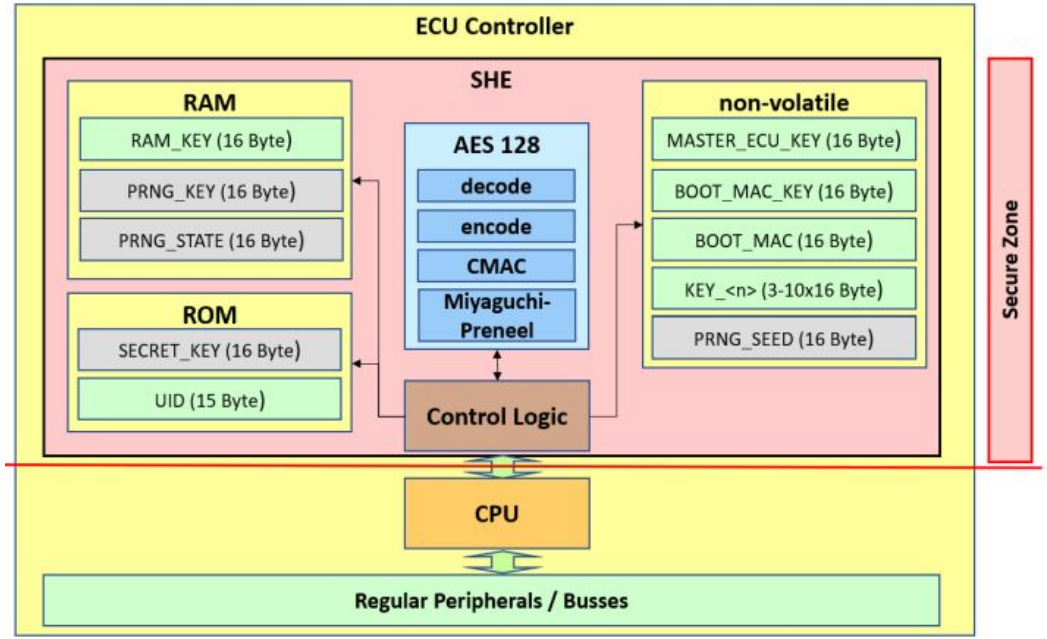
Characteristic	Software Crypto Drivers	Hardware Crypto Drivers
Execution Environment	Runs on the same application core	Runs on a dedicated hardware module (e.g., HSM, TPM)
Cryptographic Computations	Executed on the CPU (software implementation)	Executed in the HSM or hardware core (hardware-accelerated)
Security	Limited, as it shares the application core with other tasks	Enhanced security due to isolation and tamper resistance
Trust Anchor	Relies on external trust mechanisms (e.g., PKI)	Acts as a trust anchor (secure key management, signing, etc.)
Random Number Generation	Typically software-based (pseudo-random number generators)	True Random Number Generators (TRNG) available in hardware
Performance	Slower, especially for large datasets or complex algorithms	Faster, hardware-accelerated, optimized for cryptographic tasks
Use Cases	Low-volume or general-purpose cryptography	High-security environments, large-scale data encryption, HSMs

Secure Hardware Extension (SHE)?

- The Secure Hardware Extension (SHE) is a specification originally defined by the EVITA project to provide a standardized lightweight security module within microcontrollers.
- SHE is specifically designed to perform core cryptographic operations and manage secure storage for automotive applications.
- **Key Features:**
  - Lightweight Security: Offers basic cryptographic primitives such as symmetric encryption (e.g., AES-128).
  - Key Management: Enables secure storage, loading, and deletion of cryptographic keys.
  - Cost-Effective: Designed for cost-sensitive environments, SHE is integrated into existing microcontrollers without requiring additional hardware.
  - Limited Scope: Primarily focused on symmetric cryptography.it is not suitable for advanced use cases like secure boot or public key cryptography.

❖ Use Cases:

- ➢ Immobilizers for theft protection.
- ➢ Secure in-vehicle communication for CAN networks.
- ➢ Protecting firmware updates via encrypted payloads.
- ➢

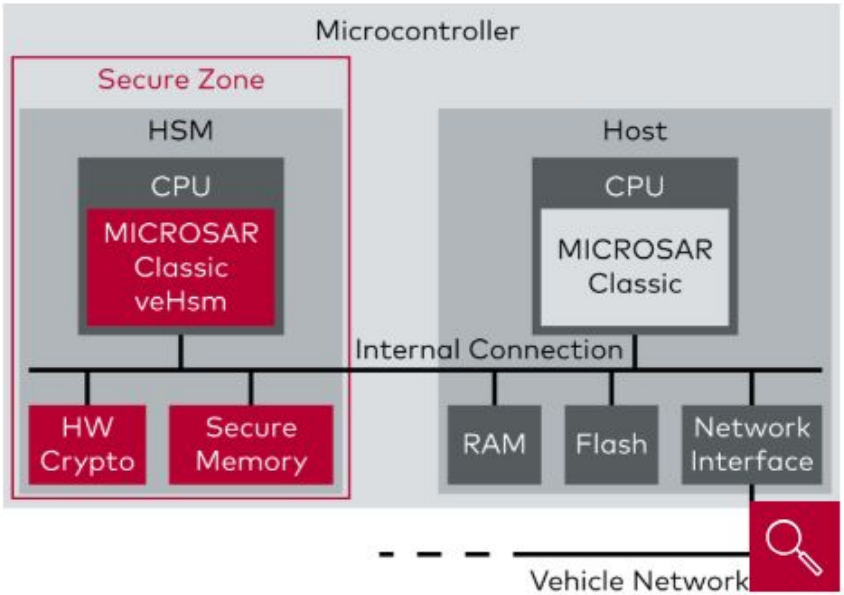


Hardware Security Module (HSM)

- HSM is a more advanced and versatile security solution.
- HSM is a dedicated co-processor within the microcontroller or as a standalone hardware unit designed to perform robust cryptographic operations.
- **Key Features:**
  - Advanced Security: Supports both symmetric and asymmetric encryption (e.g., RSA, ECC).
  - Dedicated Processing Power: Operates independently of the main processor, ensuring secure execution of cryptographic tasks.
  - Comprehensive Key Management: Offers a full lifecycle of cryptographic key generation, storage, and rotation.
  - Extended Applications: Used for secure boot, secure firmware updates, and secure communication (e.g., TLS).

❖ Use Cases:

- ➢ Secure boot and secure flash for ensuring firmware integrity.
- ➢ Establishing and maintaining secure communication channels using MACSec or TLS.
- ➢ Implementing secure vehicle-to-everything (V2X) communications.
- ➢



SHE Vs HSM

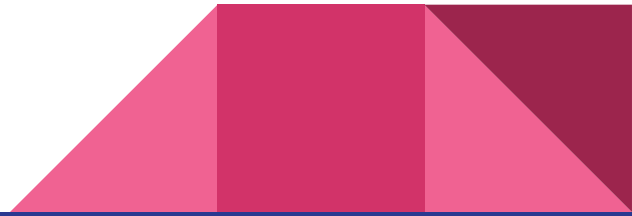
Characteristic	Secure Hardware Extension (SHE)	Hardware Security Module (HSM)
Architecture	SHE is an on chip extension within the ECU itself.	HSM is a dedicated hardware device or standalone modules not just within a single ECU.
Scalability	SHE can only be used by a single ECU.	HSM can be used by multiple systems or applications.
Primary Deployment Context	SHE is deployed specifically for automotive ECUs.	HSMs are used for other high-security applications in various industries.
Standardization	SHE is part of the AUTOSAR (Automotive Open System Architecture) standards.	HSMs follow various industry standards and certifications.
Cryptographic Support	Symmetric Encryption Only	Symmetric and Asymmetric(AES,RSA)
Key Management	Basic (AES key storage)	Advanced(Key generation,rotation)
Cost	Cost Effective	Expensive due to dedicated hardware

Cryptographic Job Management and Key Handling

- Job Concepts
- Synchronous and Asynchronous Jobs
- Queuing and Priorities
- Streaming Approach vs Single Call Approach
- Key Handling

Job Concepts

- Requests to the CSM for cryptographic routines are represented as jobs.
- A job contains the information which cryptographic routine and which cryptographic key shall be processed.
- A job does not contain the actual key data itself but references the appropriate key.
- Since the computation of cryptographic services might be very computationally intensive, job processing shall be considered as synchronous or asynchronous.



Synchronous and Asynchronous Jobs

- **Synchronous Jobs**
  - When synchronous job processing is used, the CSM service will be executed immediately in the context of the caller.
  - The result of the cryptographic routine will be available directly when the function returns.
- **Asynchronous Jobs**
  - Asynchronous jobs are processed later by the dedicated CRYPTO, in the context of a scheduled main function or in hardware.
  - The CRYPTO notifies CRYIF about the completion of an asynchronous job using a callback function of the CRYIF.
  - And CRYIF forwards the results by a callback function of the CSM.

Queuing

- The CSM may have several queues where cryptographic jobs are temporarily stored based on their priority before being passed to the Crypto Driver for execution.
- The path from a CSM queue via the CryIf to a Crypto Driver Object is called a channel.
- Each queue of the CSM is mapped to one channel to access the crypto primitives of the Crypto Driver Object.

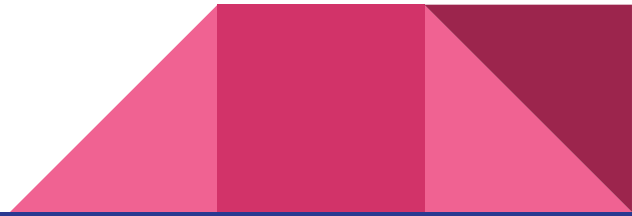
Priority

- The priority of each CSM job will be defined by its configuration.
- The higher the priority value, the higher the job’s priority.
- Before processing jobs, the CSM sorts the job queues based on the assigned priority.
- Jobs will be executed corresponding to their priority.

Job Processing Based on Queuing and Priority

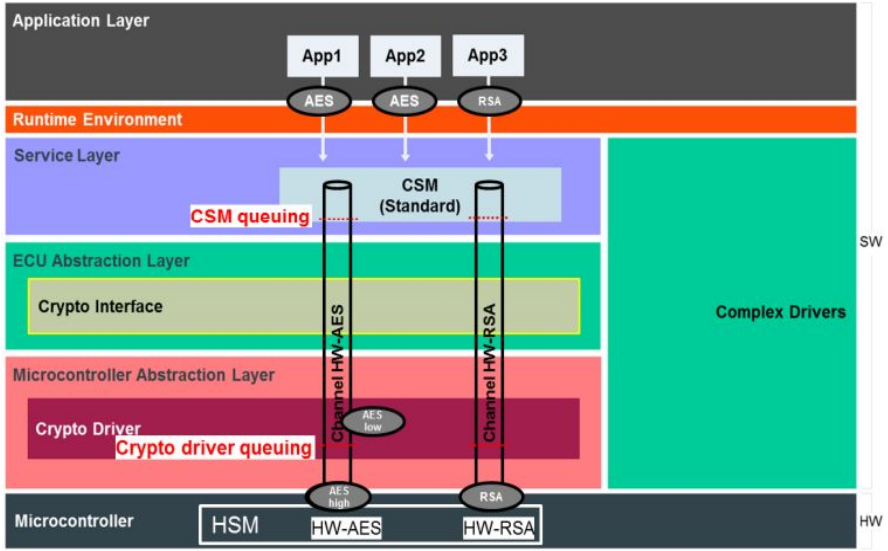
- If a synchronous job is issued and the priority is greater than the highest priority available in the queue
  - Synchronous jobs is executed immediately and it blocks other jobs (both synchronous and asynchronous) until its completion.
  - Asynchronous jobs wait in the queue until the synchronous job execution is completed.
- If a synchronous job is issued and the priority is less than the highest priority available in the queue,
  - The CSM shall return **CRYPTO_E_BUSY**.

- If an asynchronous job is issued and If the Crypto Driver Object is busy, but the queue of the CSM is not full,
  - The job will be listed into that queue in order of its priority.
  - The queued jobs are passed to the CRYIF during the cyclic Csm_MainFunction().
  - The CRYIF will forward the jobs to the particular CRYPTO.
  - As soon as the Crypto Driver Object is free, the job with the highest priority from the CSM queue will be executed.
  - Once a job is executed, it is removed from the queue, and the CSM checks for the next job in line.
- If an asynchronous job is issued and If the Crypto Driver Object is busy and the CSM queue is full,
  - The job request shall be rejected with the return value CRYPTO_E_BUSY.



Example to understand Queuing in detail

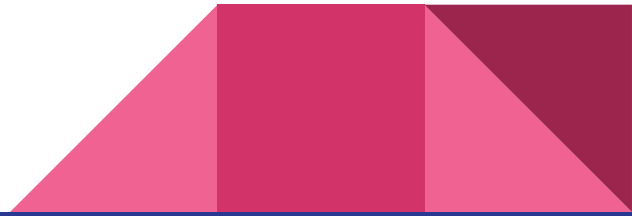
- There is a HSM with two Crypto Driver Objects (HW-AES and HW-RSA), each of them has an own channel.
- Each channel is connected to a CSM queue and a Crypto Driver Object queue.
- In this case, both Crypto Driver Objects are processing a crypto job (AES-high and RSA) each, while the queue of the Crypto Driver Object contains one more job (AES low).



Possible Scenarios and Job Process Behavior

- **Scenario 1:** AES-high Job Completion
  - **Condition:** If the HW-AES of the HSM finished the AES-high job.
  - **Behavior:** The CSM will process the AES-low job as the next job in the queue.
- **Scenario 2:** Immediate RSA Job Processing
  - **Condition:** If the Crypto Driver Object of the RSA is not busy.
  - **Behavior:** The RSA job will be processed immediately without delay.
- **Scenario 3:** RSA Busy, Queue Not Full
  - **Condition**: If the Crypto Driver Object of the RSA is busy but the queue of the Crypto Driver Object is not full.
  - **Behavior**: The RSA job will be added to the queue of the Crypto Driver Object in order of its priority. Once the Crypto Driver Object is free, the job with the highest priority from the queue will be executed.

- **Scenario 4**: RSA Busy, Queue Full
  - **Condition**: If the Crypto Driver Object of the RSA is busy and the queue of the Crypto Driver Object is full.
  - **Behavior**: The RSA job will be stored in the CSM queue, prioritized accordingly.
- **Scenario 5:** Both Queues Full
  - **Condition**: If the Crypto Driver Object of the RSA is busy, the queue of the Crypto Driver Object is full, and the CSM queue is also full.
  - **Behavior**: The CSM will reject the RSA job request.



Streaming Approach vs Single Call Approach

- The Streaming Approach and the Single Call Approach are two distinct operation modes used by Crypto Drivers for handling cryptographic operations, particularly when dealing with large amounts of data.
- **Streaming Approach:**
  - The Streaming Approach processes data incrementally or in chunks.
  - The data is sent in multiple small parts or "streams" for encryption/decryption rather than all at once.
  - This allows the operation to be carried out over several calls.
  - The data is broken into smaller segments or blocks, and each segment is processed one by one.

- Typically used for large datasets or continuous data streams, like video or large files.
- It involves multiple function calls, such as Start, Update, and Finish.
  - Start: Begins the operation with the initial data chunk.
  - Update: Handles subsequent chunks of data.
  - Finish: Completes the operation, handling any final data.

Advantages:

- Efficient for large data: Useful when data exceeds memory limitations or when you need to process large files or streaming data.
- No need to load the entire data into memory: Allows for processing data as it is received without requiring the entire dataset to be loaded at once.
- Real-time data processing: Suitable for continuous streams of data, such as network communication or video encryption.

- **Single Call Approach:**
  - The Single Call Approach processes all data in one single function call.
  - The entire dataset is provided in one go, and the cryptographic operation is completed in that single call.
- **How Single Call Approach Works:**
  - The entire block of data is passed at once to the crypto driver, which handles the entire operation in one execution.
  - It involves just one function call that typically performs the entire cryptographic operation.
- **Advantages:**
  - Simpler and faster for small data:
    - It is more efficient for smaller datasets or when the data size is manageable.
  - Less overhead:
    - Since there is no need for multiple calls or chunks, it reduces the overhead of managing the operation.
  - Single-step processing:
    - Ideal for one-time operations on small blocks of data.
    - Suitable forEncryption of small files or password hashing

Streaming Approach vs Single Call Approach

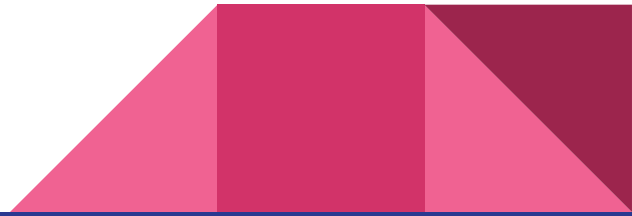
Characteristic	Streaming Approach	Single Call Approach
Data Handling	Processes data in chunks or streams.	Processes entire data in one single call.
Processing Calls	Requires multiple calls (Start, Update, Finish).	Requires only one function call to process all data.
Suitable for	Large datasets or continuous data streams.	Small data or single-time operations.
Memory Usage	Can handle large data without loading it all at once.	Requires the entire data to be loaded into memory.
Latency	Slightly higher due to multiple steps involved.	Lower, as it processes the entire data in one go.
Complexity	More complex (multiple function calls).	Simpler (single function call).
Real-time Processing	Suitable for real-time data streaming (e.g., media).	Not ideal for real-time data streaming.

Key Handling

- The key handling comprises the generation, update, im-/export, exchange, and derivation of cryptographic keys.
- Cryptographic keys may be stored in cryptographic hardware or using NvM.

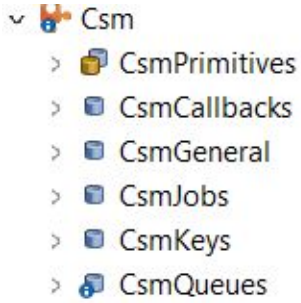
AUTOSAR Crypto Stack Configuration

- Configurations in Crypto Service Manager(CSM)
- Configurations in Crypto Interface(CRYIF)
- Configurations in Crypto Driver(CRYPTO)



Configurations In Crypto Service Manager(CSM)

- The configuration of the crypto stack is typically carried out using the top-down approach, starting from the CSM over CRYIF to CRYPTO.
- The user selects and configures the cryptographic Primitives.
- The user selects and configures the keys to be used.
- The user defines cryptographic jobs.
- The user defines cryptographic job queues.

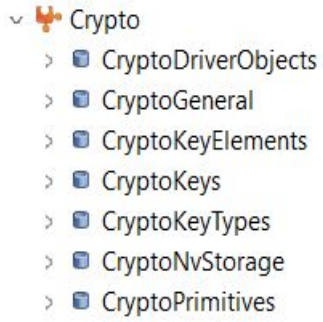


Configurations in Crypto Interface(CRYIF)

- The user selects and configures the CryIf Channels.
- The user selects and configures the CryIf Keys.

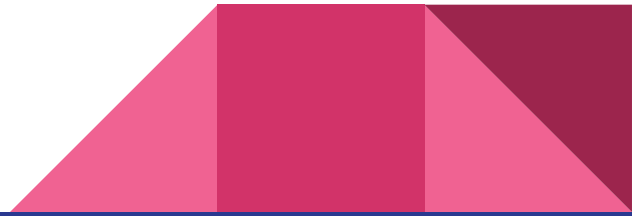
Configurations in Crypto Driver(CRYPTO)

- The user selects and configures the Crypto Driver Objects.
- The user selects and configures the Crypto General.
- The user selects and configures the Crypto Key Elements.
- The user selects and configures the CryptoKeys.
- The user selects and configures the Crypto Key Types.
- The user selects and configures the Crypto NvStorage.
- The user selects and configures the Crypto Primitives.



Configurations in detail

- CSM Configurations
- CRYIF Configurations
- CRYPTO Configurations



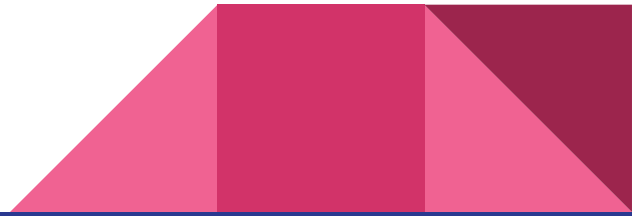
CSM Configurations

- **Configure CsmPrimitives**
  - Configure CsmPrimitives using any allowed schemes from the underlying Crypto Driver.
  - When creating a Primitive configuration, it is possible to configure all available and allowed schemes from the underlying Crypto Driver.
  - Each crypto Primitive configuration is realized as a constant structure of type Crypto_PrimitiveInfoType.
  - Each job Primitive configuration is realized as a constant structure of type Crypto_JobPrimitiveInfoType.
  - Several configurations for each cryptographic Primitive can be configured.
  - One configuration per job per Primitive is possible.

Configure CsmJobs

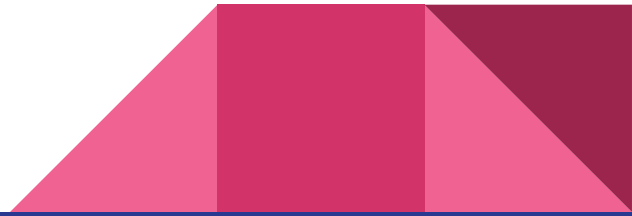
- Configure the **CsmJobPrimitiveRef** to refer to the used CsmPrimitive.
- Configure the **CsmProcessingMode** to CRYPTO_PROCESSING_ASYNC or CRYPTO_PROCESSING_SYNC.
- If CsmProcessingMode of the CSM job is configured as CRYPTO_PROCESSING_ASYNC , then configure CsmJobPrimitiveCallbackRef to contain a callback function.
- Configure the **CsmJobKeyRef** in the CsmJob with reference to the the key which shall be used for the CsmPrimitive.
- Configure the **CsmJobPriority**.The higher the job priority value, the higher the job's priority.
- Configure the **CsmJobQueueRef**.The CsmQueues shall sort the jobs according to the configured job's priority.

- **Configure CsmKeys**
  - Configure CsmKeyRef to the used CryIfKey in the underlying CryptoKey in the Crypto Driver.
- **Configure CsmQueues**
  - Configure CsmChannelRef to the underlying CryIfChannel channel.
  - Configure the CsmQueueSize to the max number of configured asynchronous jobs.
  - Configure the CsmTriggerAsynchJobsInCallback to enable Immediate job triggering.
    - If enabled, queued jobs will be passed to the CRYIF for processing directly in the Csm_CallbackNotification instead of delaying this to the next Csm_MainFunction call.



CryIf Configuration

- Configure CryIfChannels and refer in CryIfDriverObjectRef which Crypto Driver Object the CryIf channel is connected to.
- Configure CryIfCryptoModule and refer the Crypto Driver
- Configure the CryIfKeys and Specify to which crypto driver key the CryIf key is mapped to.



Crypto Driver Configuration

- Configure the CryptoDriverObjects
- Configure the CryptoGeneral
- Configure the CryptoKeyElements
- Configure the CryptoKeys
- Configure the CryptoKeyTypes
- Configure the CryptoNvStorage
- Configure the CryptoPrimitives

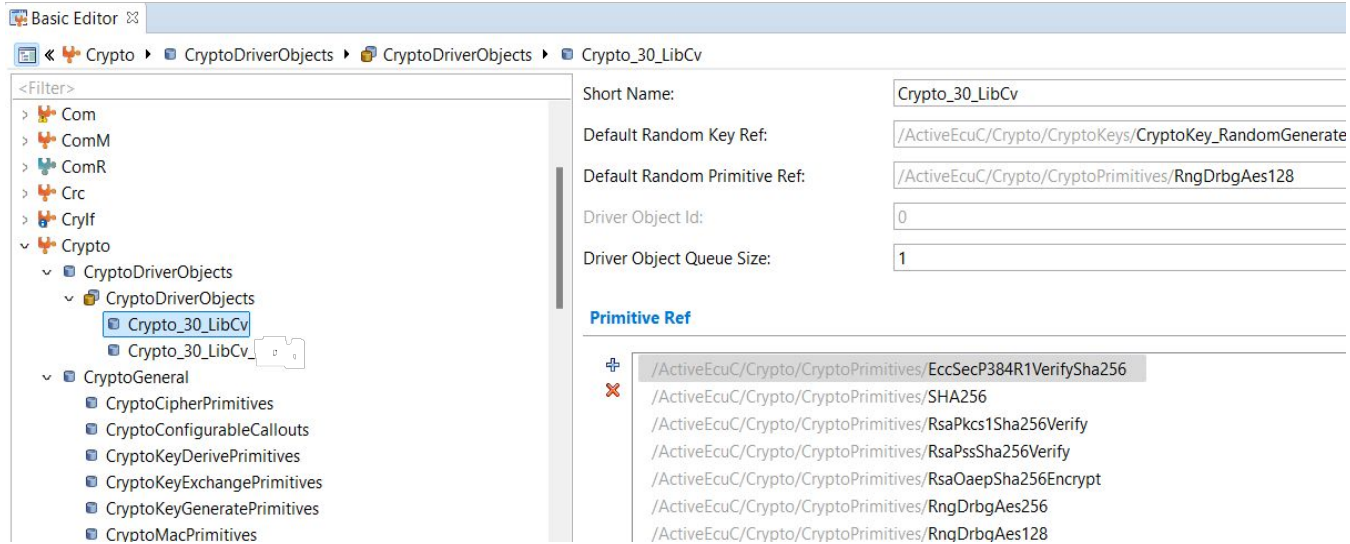
Configure the CryptoDriverObjects

- Multiple CryptoDriverObjects can be configured .
  - As soon as a job is started on a driver object for one algorithm, driver object is locked and therefore busy for another algorithm.
  - If it is necessary to execute another job in between, a further driver object can be created o execute the job in parallel.
  - Ensure that a job is owned and called by only one application since the driver cannot differ between one job from different callers.
- **Configure the CryptoDefaultRandomPrimitiveRef**
  - If a crypto driver object needs to perform a crypto primitive that requires a random number generator, but the configuration of this primitive does not provide parameter for a random number generator, then this default random number generator shall be used.
  - For the Default Random Source only the DRBG can be used.
  - An algorithm for random number generation needs to be seeded before generation.
  - The service RandomSeed is used to set the new entropy.
  - RNG is seeded with new entropy when the generation job is processed.

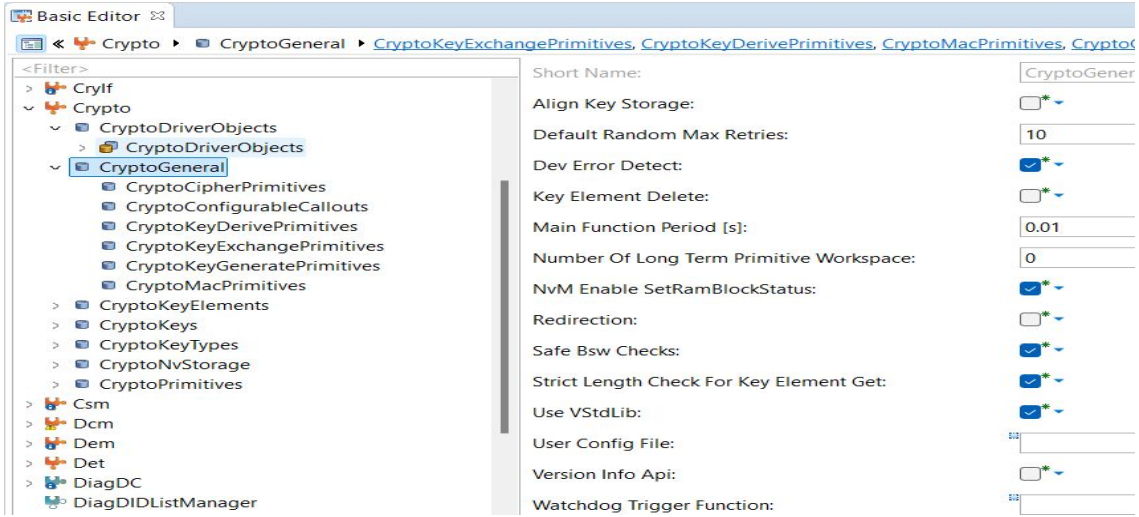
❖ Configure the CryptoDefaultRandomKeyRef

- ➢ The key contains key elements that are necessary to seed the random number generator.
- ➢ This element shall only be set if the primitive referenced by CryptoDefaultRandomPrimitiveRef requires a seed value.
- ➢ During the random number generation, the key is locked for writing.
- ➢ If one key is used for several RNG jobs, only the first executed random instance gets seeded with this entropy.
- ➢ Therefore, it is highly recommended that each random instance has its own key, to avoid side effects.
- ➢ The RNG needs to be seeded after each startup.
- ➢ The DRBG gets instantiated with the first Crypto_30_LibCv_RandomSeed call.
- ➢ The key element should be configured to be persisted, or else the DRBG will provide the same random numbers after restart.

❖ Sample Configuration of CryptoDriverObject

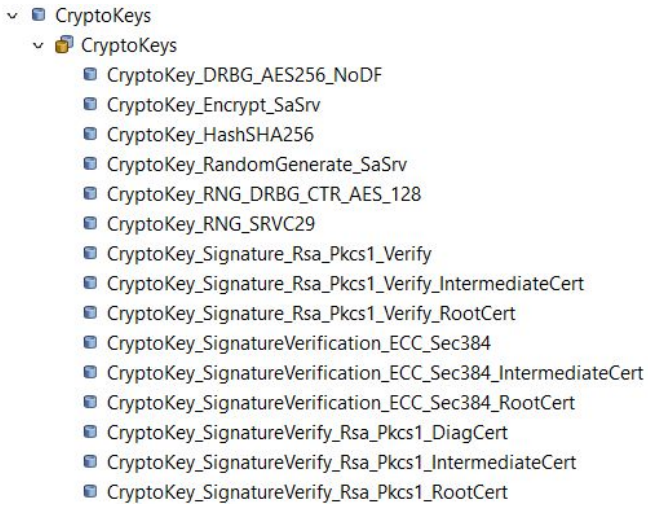


CryptoGeneral

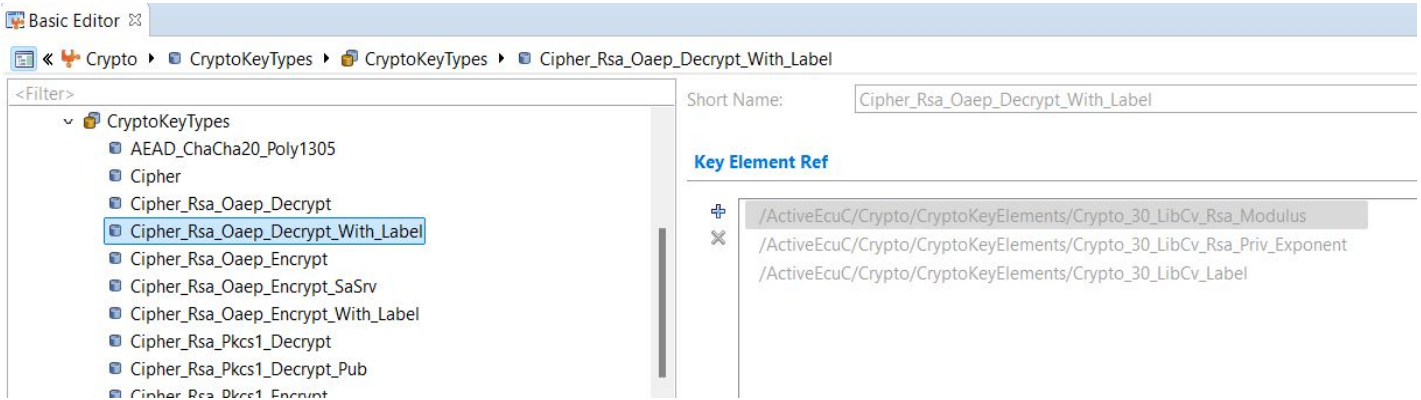
- Configure the common configuration options.
- Configure the algorithm used for the crypto cipher jobs,crypto mac jobs.
- Configure the callouts used for the callouts to notify the application.
- 

Configure the CryptoKeys

- In the Crypto Driver, the CryptoKeys references a specific key type.
- The CryptoKeyType provides the information about which key elements are contained in a CryptoKey referencing this CryptoKeyType.
- A CryptoKey consists of one or more key Elements.



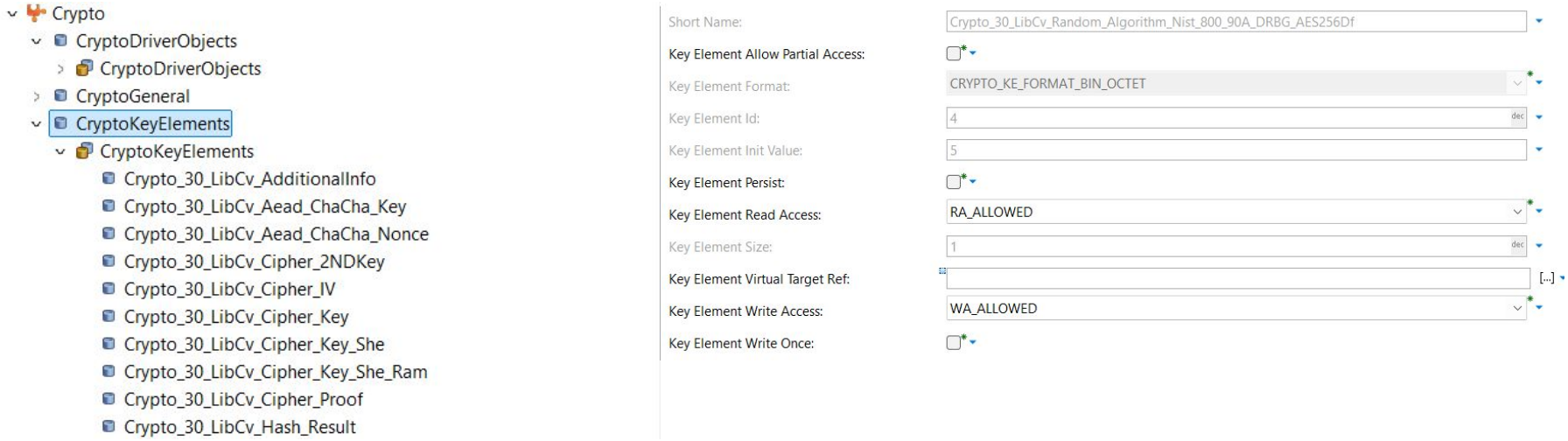
Configure the CryptoKeyTypes

- The CryptoKeyType consists of references to key elements.
- The CryptoKeyTypes are typically pre-configured by the vendor of the Crypto Driver.
- Further CryptoKeyTypes may be added by the user by using Pre configured key types as template by duplicating them.
- 

❖ Configure the CryptoKeyElements

- ➢ Key elements are used to store data, such as key material or the Initialization Vector (IV) needed for AES encryption.
- ➢ Key elements can also be used to configure the behavior of key management functions.
- ➢ Key elements from different keys have different memory areas, which can be in both non-volatile (NV) and RAM areas.
- ➢ The vendor pre-configures the key elements.
- ➢ Configure the read/write access (permissions for accessing the element).
- ➢ Configure the the maximum size of the key in bytes.
- ➢ Configure the need for persisting of the key element in non-volatile storage.
- ➢ Configure the initial value of the key element after startup if the element is not already initialized.

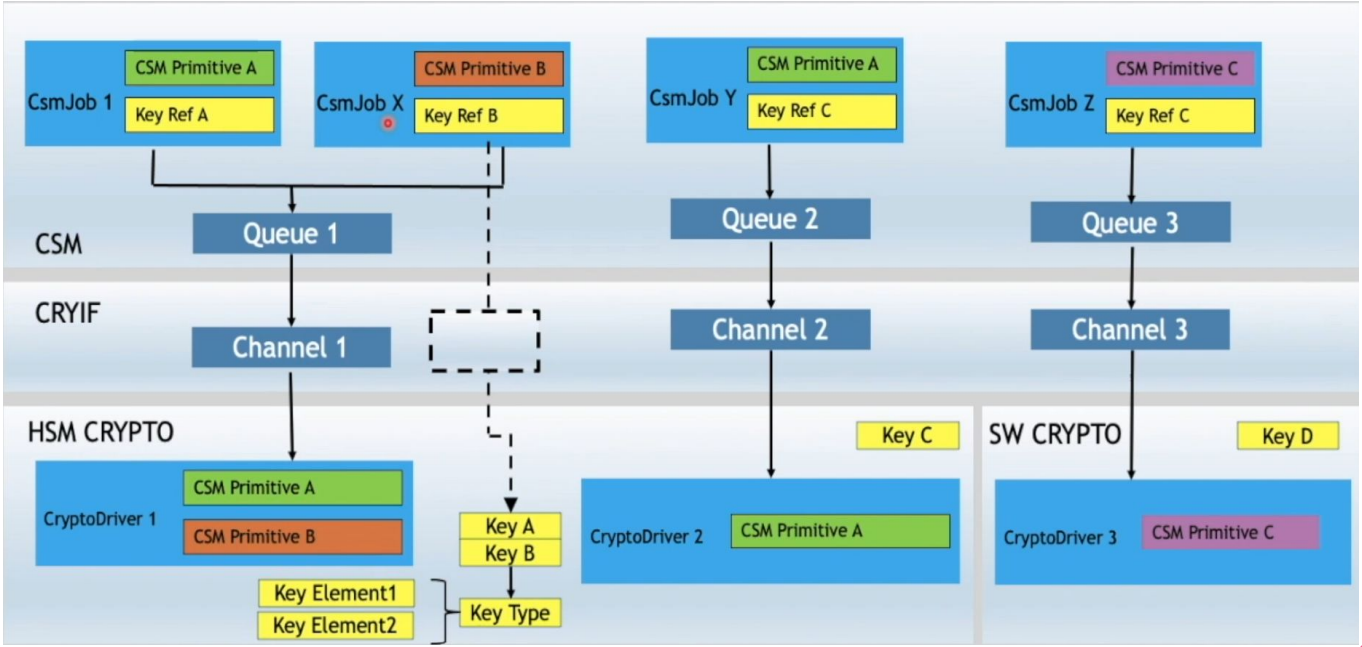
❖ Sample Configuration of CryptoKeyElements



- **Configure the CryptoNvStorageReferences with NvM blocks**.
  - The CRYPTO supports persisting of key elements to the NvM.
  - Keys with key elements that shall be persisted are to be assigned to these blocks.
  - The elements that are to be persisted will then be written by the driver on validation and values are retrieved during initialization of the driver.
  - Storing keys permanently is done by calling Crypto_KeySetValid with the configuration parameter CryptoKeyElementPersist set.
  - The key element storage in NVM shall not be initiated on a call to Crypto_KeyElementSet().
  - The storage operation shall only take place if the validation was successfully finished and the operation returns E_OK.
  - On initialization of the crypto driver, the callback function Crypto_<vi>_<ai>_NvBlock_ReadFrom_<NvBlock>() shall be called to retrieve the previously persisted key elements for the associated Keys.

- **Configure the CryptoPrimitives.**
  - Example 'algorithmfamily' (e.g. AES, MD5, RSA, ...), and a 'algorithmmode' (e.g. ECB, CBC, ...).
  - Primitive determines the algorithm family used for the crypto service.
    - Example CRYPTO_ALGOFAM_DRBG .
  - Primitive determines the algorithm mode used for the crypto service.
    - Example CRYPTO_ALGOMODE_CTRDRBG.
  - Primitive Determines the algorithm secondary family used for the crypto service.
    - Example CRYPTO_ALGOFAM_AES.
  - To support primitive variants which are not defined by AUTOSAR, the custom Primitive defines can be used within this Crypto driver.
    - The value of each define can be configured by changing the CryptoPrimitiveAlgorithmFamilyCustomId for the respective CryptoPrimitiveAlgorithmFamilyCustom or CryptoPrimitiveAlgorithmModeCustom.

Structural Overview Of Autosar Crypto Stack



AUTOSAR Crypto Stack Operation Flow

- Application Requesting A Cryptographic Operation
- Validating in CSM
- Queuing in CSM
- Forwarding in CSM
- Mapping in CryIf
- Queuing in CryIf
- Processing in Crypto Driver
- Forwarding Results In CryIf
- Forwarding Results In CSM

Application Requesting A Cryptographic Operation

- The application uses a CSM API to request a cryptographic operation.
- Parameters provided include:
  - Operation Type: Encryption, decryption, hashing, etc.
  - Mode:
    - For Streaming: START, UPDATE, or FINISH.
    - For Single Call: A single API invocation.
  - Algorithm: The cryptographic algorithm (e.g., AES, SHA).
  - Key Reference: Identifier of the cryptographic key.
  - Input Data: The data to be processed.
  - Callback Function (only for asynchronous): To receive the result once processed.

Validating in CSM

The Crypto Service Manager (CSM) performs the following:

- The CSM Validates the job parameters:
  - Algorithm, key reference, Data size, and mode for its algorithm compatibility, key validity and input data correctness respectively.
- The CSM ensures that enough resources are available (e.g., memory, CPU).
- The CSM Creates a Job Object, which includes:
  - Job priority.
  - Reference to the key.
  - Cryptographic operation type.
  - Callback function for result notification if the Mode of operation is Asynchronous.

Queuing in CSM

- The CSM Performs Queuing:
  - For Asynchronous mode:
    - Queues the job.
    - Jobs are sorted by priority in the CSM queue.
    - If the Crypto Driver is busy and cannot handle the job, the job waits in the queue.
  - For Synchronous mode:
    - The request is forwarded immediately.
    - If the Crypto Driver is busy, The job is rejected with CRYPTO_E_BUSY.

Forwarding in CSM

- CSM periodically calls its main function (Csm_MainFunction()), which processes jobs in the queue.
  - For an Asynchronous job:
    - The job is dequeued from the CSM queue.
    - The CSM forwards the job to the Crypto Interface (CryIf) which is responsible for mapping the job to the correct Crypto Driver.
  - For a Synchronous job:
    - The CSM forwards the job immediately to the Crypto Interface (CryIf) which is responsible for mapping the job to the correct Crypto Driver.

Mapping in CryIf

- CryIf identifies the appropriate Crypto Driver (CRYPTO) based on the configuration.
- CryIf maps the job to a specific Crypto Driver Object based on:
  - The Algorithm Type (AES, RSA, SHA).
  - The CryIf Channel configurations, which are logical channels that links CryIf to the corresponding Crypto Driver.
  - The key Reference provided in the job request to ensure that the correct cryptographic key is used.

Queuing in CryIf

- CryIf checks:
  - Availability of the Crypto Driver Object.
  - Job compatibility with the driver.
- CryIf forwards the job to the target Crypto Driver Object.
- If the target Crypto Driver Object is busy:
  - Asynchronous jobs are queued in CryIf.
  - Synchronous jobs are rejected (CRYPTO_E_BUSY).

Processing in Crypto Driver

- If hardware acceleration is available (e.g., HSM), the driver offloads the operation to the hardware.
- Key steps in the driver:
  - **For Streaming Approach**:
    - **START**:
      - CSM forwards the START phase to CryIf, which sends it to the Crypto Driver.
      - Initializes the cryptographic context, e.g., AES key setup, buffer allocation.
      - Loads the cryptographic key (e.g., AES key) into the Crypto Driver.
      - If hardware acceleration is used, the driver prepares the hardware resources.
      - Once initialization is complete
        - For asynchronous jobs the driver sends a callback to CryIf to inform the completion.
        - For synchronous jobs the river returns control to CryIf.

- **UPDATE:**
  - CSM calls Csm_Service_Update(keyID, inputDataChunk, chunkSize, outputData).
  - CryIf forwards the chunk of data to the Crypto Driver.
  - Processes the input data chunk incrementally.
  - Encrypts, hashes, or signs the chunk and updates its internal state (e.g., ciphertext or hash).
  - Once the chunk is processed,
    - For asynchronous jobs the driver sends a callback to CryIf
    - For synchronous jobs the driver immediately returns the processed chunk to CryIf.
- **FINISH**:
  - CSM calls Csm_Service_Finish(keyID, outputData, &outputLength).
  - CryIf forwards the FINISH request to the Crypto Driver.
  - Crypto Driver Completes the operation, e.g., finishes AES encryption and generates the final ciphertext.
  - Frees up any resources that were allocated during the START phase.
  - For asynchronous jobs the final result is sent to CryIf via a callback.
  - For synchronous jobs the final result(e.g., the ciphertext) is directly returned to CryIf.

- Key steps in the driver:
  - **For Single Call Approach:**
    - Performs the entire operation (START → UPDATE → FINISH) in one step.
    - Initializes the cryptographic context, e.g., AES key setup, buffer allocation.
    - Loads the cryptographic key (e.g., AES key) into the Crypto Driver.
    - Prepares internal state, including memory buffers for intermediate results.
    - Processes the input data chunk.
    - Crypto Driver Completes the operation.
    - Frees up any resources
    - For Asynchronous jobs:
      - The Crypto Driver notifies CryIf of the operation result via a callback in a separate thread/context.
    - For Synchronous jobs:
      - The Crypto Driver returns the result immediately to CryIf in the same thread/context.
    - The callback includes
      - Job result (e.g., success, error)
      - Output data (e.g., encrypted data, hash).

Forwarding Results In CryIf

- After the Crypto Driver processes the job, CryIf handles the result:
  - For Asynchronous jobs:
    - Forwards the result to the CSM via callback.
  - For Synchronous jobs:
    - Returns the result directly to the CSM.
  - CryIf may clean up resources associated with the job.

Forwarding Results In CSM

- The CSM delivers the final result to the application:
  - For Asynchronous jobs:
    - The application is notified via a callback function.
    - The CSM invokes the application’s callback function with:
      - Status of the cryptographic operation.
      - Processed data (e.g., ciphertext, signature).
  - For Synchronous jobs:
    - The result is returned as the API's output.
    - The application retrieves the result and continues its execution.

SEQUENCE DIAGRAM

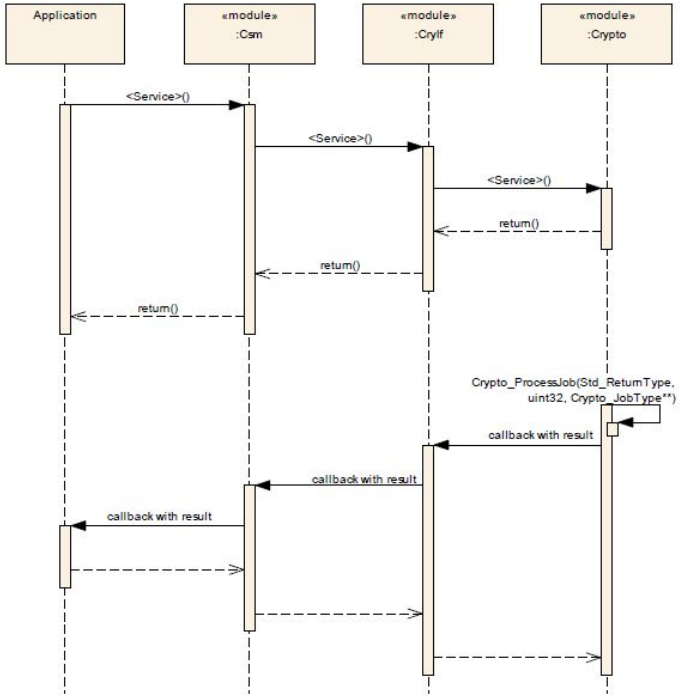


Figure 9-1 Sequence Diagram for Asynchronous Call with Callback

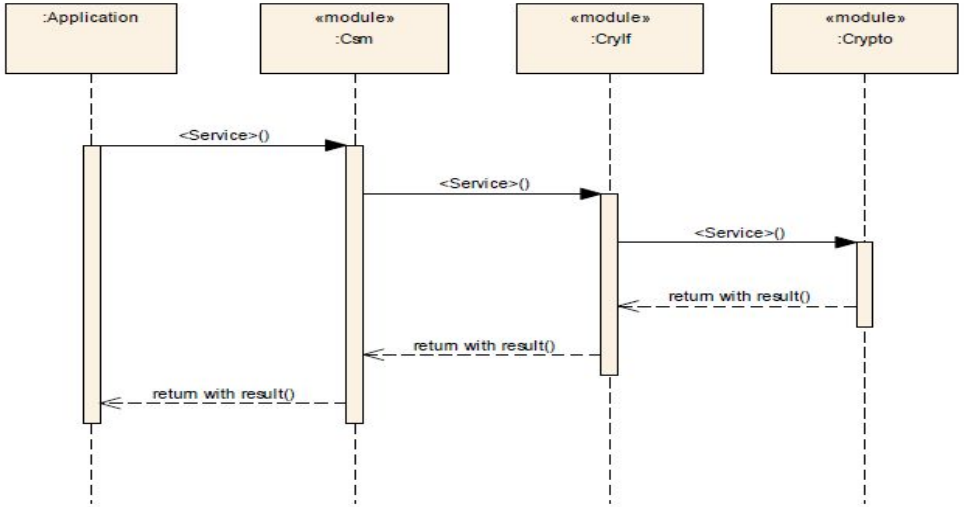


Figure 9-2 Sequence Diagram for Synchronous Call

OpenSSL Useful Commands

- **Generate RSA private key:**
  - *openssl genrsa -out PrivKey2048.pem 2048*
  - *openssl genrsa -out PrivKey3072.pem 3072*
- **Extract the public key from private key:**
  - *openssl rsa -in PrivKey2048.pem -outform PEM -pubout -out pubKey2048.pem*
  - *openssl rsa -in PrivKey3072.pem -outform PEM -pubout -out pubKey3072.pem*
  - input: PrivKey2048.pem (The Generated Private key)
  - output: pubKey2048.pem (The Extracted Public key)
- **Extract modulus and exponent from public key:**
  - *openssl rsa -pubin -in pubKey2048.pem -text -out pubKeuModExp.txt*
  - *openssl rsa -pubin -in pubKey3072.pem -text -out pubKeuModExp.txt*
  - input: pubKey2048.pem (Public key)
  - output:pubKeuModExp.txtpem (modulus and exponent)

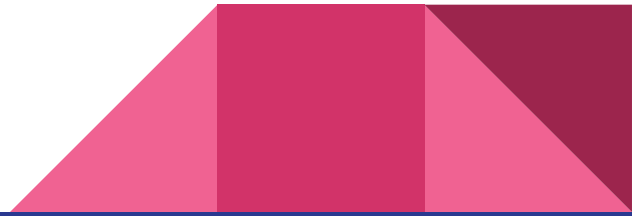
- **Extract modulus and exponent from private key:**
  - *openssl rsa -in PrivKey2048.pem -text -out privKeuModExp.txt*
  - *output:privKeuModExp.txt (modulus and exponent)*
- **RSA Encryption:**
  - *openssl pkeyutl -in plaintext.txt -encrypt -inkey publickey.txt -pubin -pkeyopt rsa_padding_mode:oaep -pkeyopt rsa_oaep_md:sha256 -pkeyopt rsa_mgf1_md:sha256 -out ciphertext.txt*
  - inputs: plaintext.txt (Text to be encrypted)
  - publickey.txt(Public key to encrypt with)
  - output: ciphertext.txt(Text after encryption)
- **RSA Decryption:**
  - *openssl pkeyutl -in ciphertext.txt -decrypt -inkey privatekey.txt -pkeyopt rsa_padding_mode:oaep -pkeyopt rsa_oaep_md:sha256 -pkeyopt rsa_mgf1_md:sha256 -out plaintextafterdecrypt.txt*
  - inputs: ciphertext.txt (Text to be decrypted)
  - privatekey.txt(private key to encrypt with)
  - output: plaintextafterdecrypt.txt (decrypted text)

- **RSA PSS sign**
  - *openssl dgst -sha256 -sigopt rsa_padding_mode:pss -sigopt rsa_pss_saltlen:32 -sign rsa_key.pem -out OutputFile.sig random.bin*
  - inputs: rsa_key.pem (Private key to sign with)
  - random.bin(Data to be signed)
  - output :OutputFile.sig(Signed Data)
- **RSA PKCS sign**
  - *openssl dgst -sha256 -sigopt rsa_padding_mode:pkcs1 -sign Privatekey.pem -out OutputFile.sig appl.bin*
- **RSA PSS verify**
  - *openssl dgst -sha256 -sigopt rsa_padding_mode:pss -sigopt rsa_pss_saltlen:32 -verify pubKey2048.pem -signature sig.bin random.bin*
  - *openssl dgst -sha256 -sigopt rsa_padding_mode:pss -sigopt rsa_pss_saltlen:32 -verify pubKey3072.pem -signature sig.bin random.bin*
  - Input: pubKey2048.pem(Public key to verify with)
  - sig.bin(signature to be verified)
  - random.bin(data before signing ro verify against)

- **RSA PKCS verify**
  - *openssl dgst -sha256 -sigopt rsa_padding_mode:pkcs1 -verify pubKey2048.pem -signature sig.bin appl.bin*
- **To view any text in DER format**
  - *openssl rsa -inform DER -pubin -text -in publicKey.der -out publicKey.txt*
  - *openssl rsa -inform DER -pubin -in publicKey.der -out publicKey.pem*
  - *openssl x509 -in FDSProject_2306_E.cer -inform DER -noout -pubkey -text> key_E.txt*
- **CMAC AES128**
  - *openssl dgst -mac cmac -macopt cipher:aes-128-cbc -macopt hexkey:11223344556677889900112233445566 -sha1 <file>*

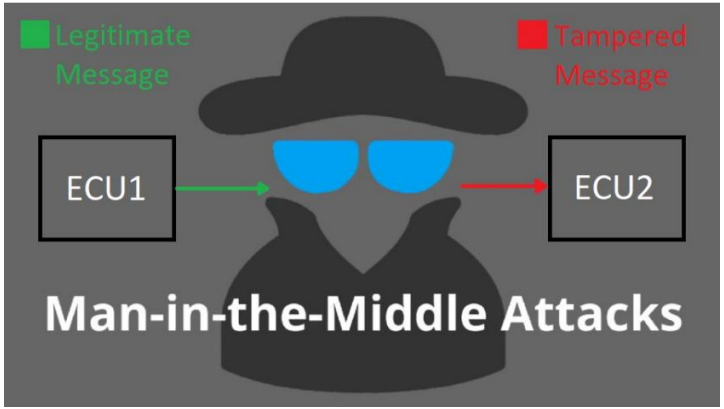
Security Concepts Using AUTOSAR Crypto Stack

- Secure on board communication(SecOC)
- Secure Diagnostics
  - Secure Access UDS Service 0x27
  - Secure Authentication UDS Service 0x29
- Secure Update
- Secure Boot
- Secure Storage
- Secure Debug
- Secure Hardware Protect
- Secure Software Development



Why is secure on-board communication necessary?

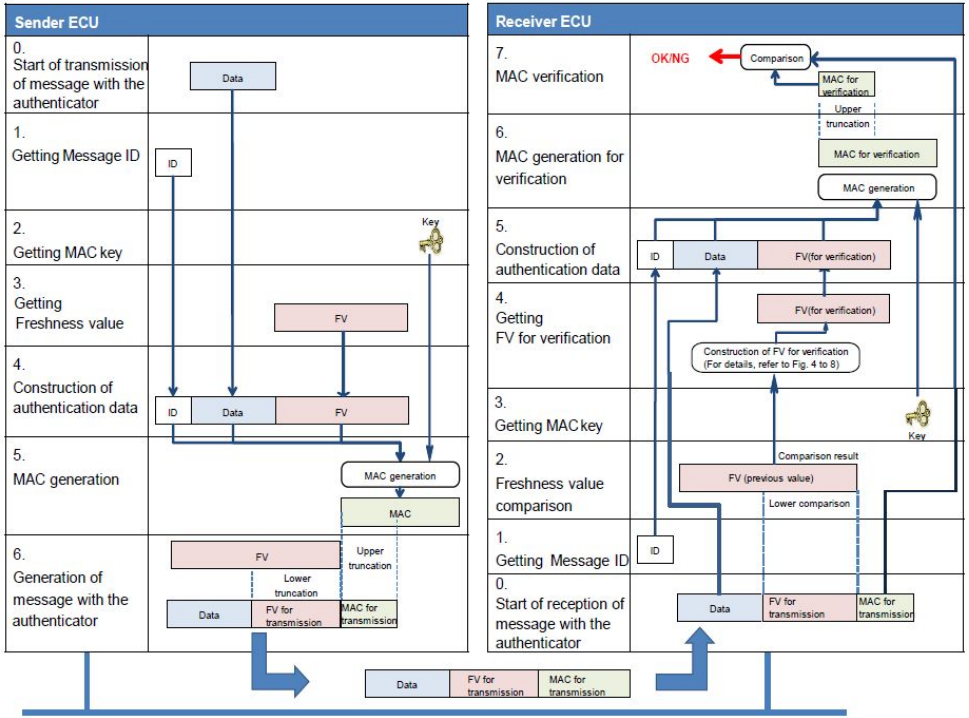
- **Possible threats**
  - Replay Attacks
  - Tampering and man-in-the-middle attacks
  - Random Errors
- **Possible repercussions**
  - Access to physical controls
    - Compromised safety of passengers
  - Tampering with mileage, emissions, etc
    - not meeting regulations
  - Intellectual property theft



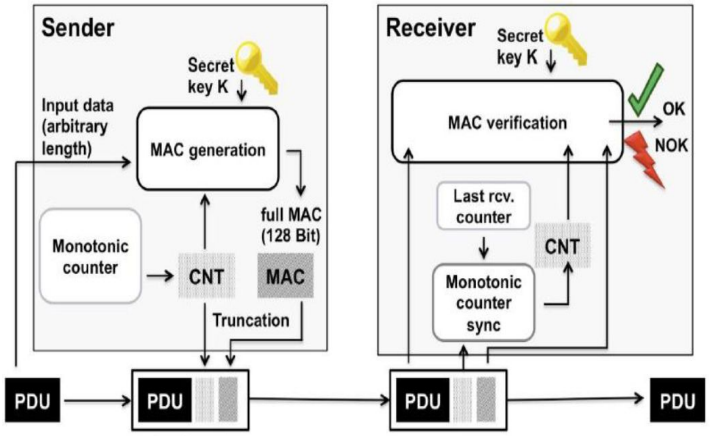
What is SecOC?

- SecOC provides mechanisms to secure communication channels between ECUs, ensuring that data sent over the network is protected against modification and impersonation.
- **Core Functions:**
  - Message Authentication: Verifies the integrity and origin of messages to ensure they haven't been altered in transit using MAC.
  - Replay Protection: Prevents attackers from capturing and retransmitting valid messages to manipulate system behavior using Freshness value.
  - Key Management: Manages cryptographic keys used for authentication and integrity checks.

How Authentication takes place ?

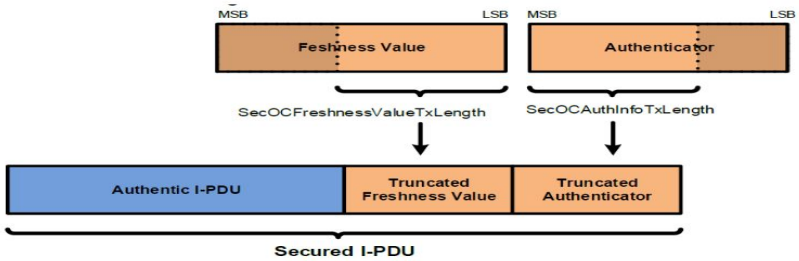


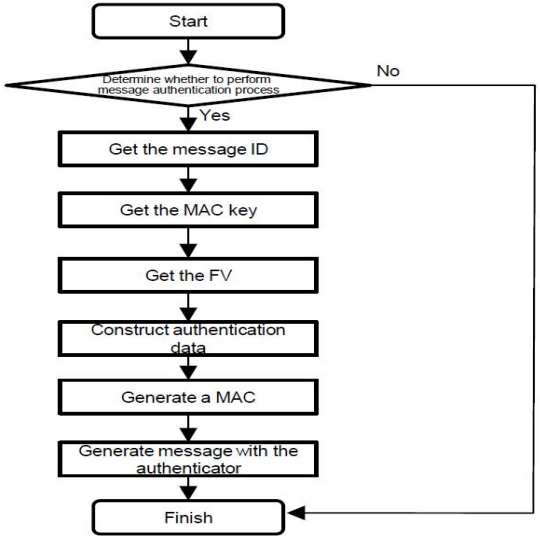
[ID: Message ID, Data: Protected data, FV: Freshness value, MAC: Message authentication code]



Transmission of Secured PDUs

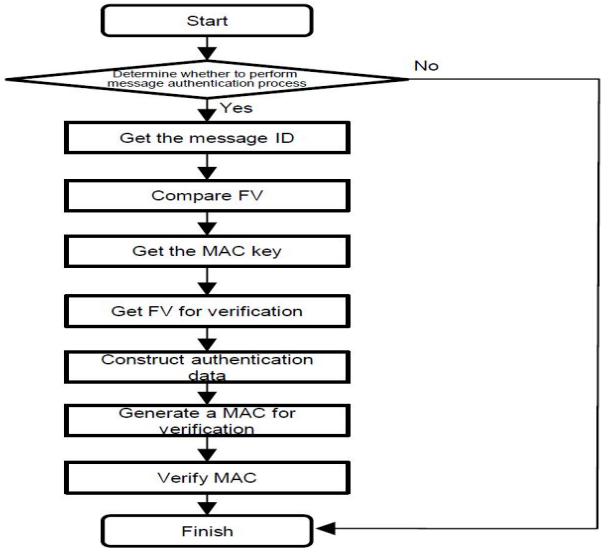
- On the sender side,
  - The SecOC module creates a Secured I-PDU by adding authentication information to the outgoing Authentic I-PDU.
    - The authentication information comprises of an Authenticator (e.g. Message Authentication Code) and optionally a Freshness Value.
  - Increment the Freshness Counter.
  - Transmit Secured I-PDU.





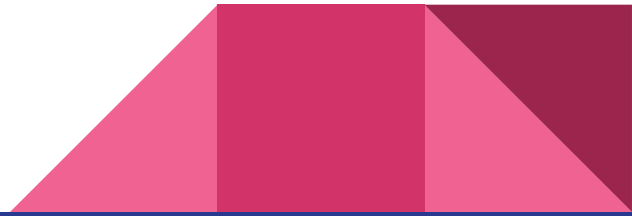
Reception of Secured PDUs

- On the receiver side,
  - The SecOC module checks the freshness and authenticity of the Authentic I-PDU by verifying the authentication information that has been appended by the sending side SecOC module.
  - Parse Authentic I-PDU, Freshness Value and Authenticator
  - Get Freshness Value from Freshness Manager
  - Construct Data to Authentication
  - Verify Authentication Information
  - Send Confirmation to Freshness Manager
  - Pass Authentic I-PDU to upper layer

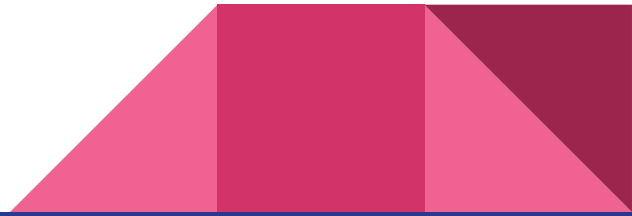


How FVM helps in Preventing Replay Attacks?

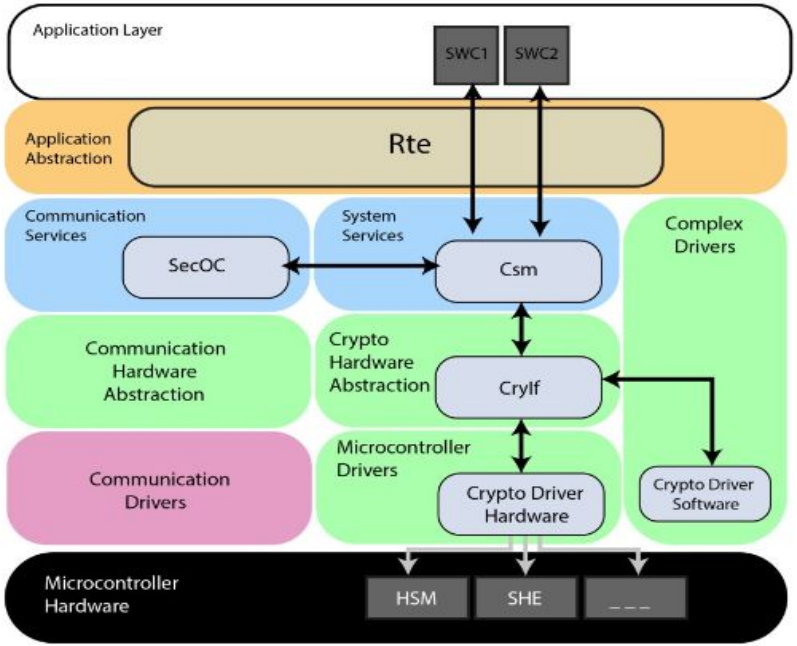
- The FVM generates a random value for each message at the sender's side.
- When an ECU prepares a message to be sent, the FVM appends the generated freshness value to the message.
- Upon receiving a message, the receiving ECU extracts the freshness value.
- The FVM checks this freshness value against its internal records to verify if it has been previously processed.
- If the freshness value is new, The message is deemed valid and is processed further.
- If the freshness value is already used The message is rejected as a potential replay attack.

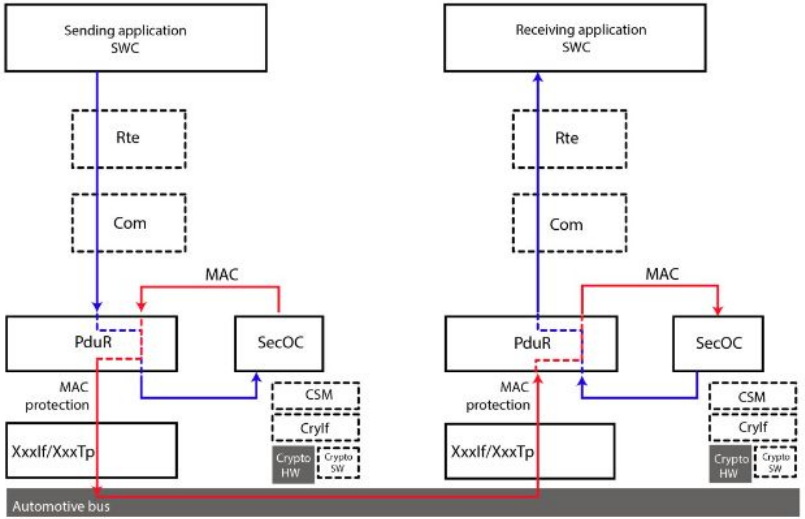


- The FVM maintains a sliding window of recent freshness values to efficiently track and manage what has been seen recently.
- The FVM can define a maximum age for freshness values, which adds an additional layer of security by ensuring that even if a value is unique, it must still be within an acceptable time frame.
- Each freshness value shall have an expiration time allowing too old messages to be rejected, further mitigating risks associated with time-based attacks.

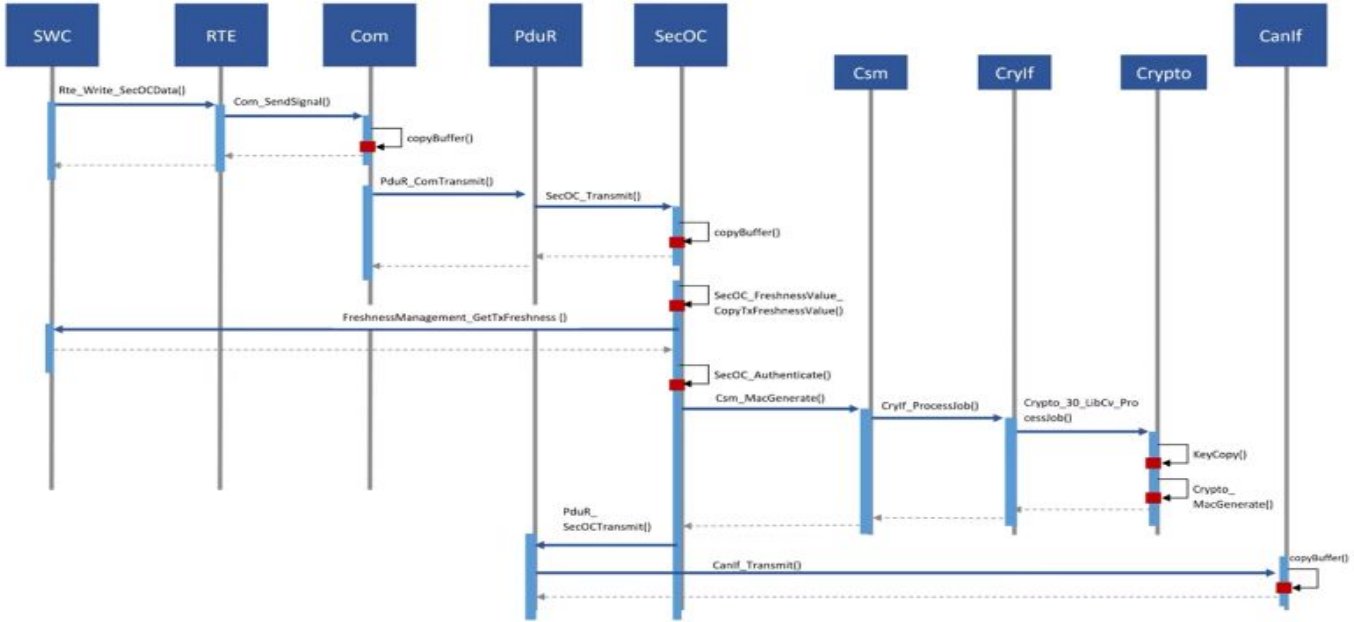


SecOC Architecture Overview in Autosar





SecOc Message Transmission



Secure Diagnostics

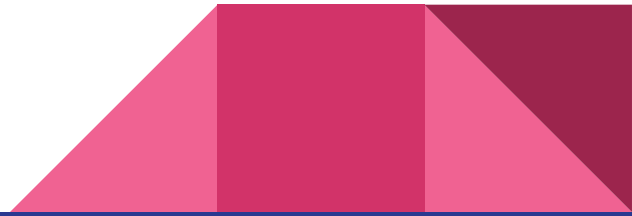
- Secure diagnostics is a way to establish a secure session over the UDS protocol.
- It is meant to ensure the authentication of the diagnostic tool that is interacting with the system.
- Before providing sensitive services, the system shall ensure that the UDS device that it interacts with is authenticated.
- The system shall only accept requests from authorized UDS devices.
- Following UDS Services can be used for Secure Diagnostics.
  - Secure Access UDS Service 0x27
  - Secure Authentication UDS Service 0x29

Secure Access in Automotive Diagnostics

- **What is Security Access?**
  - Security Access, is a diagnostic service used in the automotive industry to ensure secure access to restricted ECU (Electronic Control Unit) functionalities.
  - It uses cryptographic operations for secure authentication between the Diagnostic Tool (Tester) and the ECU.
  - The service involves the exchange of cryptographically signed messages to ensure that only authorized tools or users can access sensitive operations.

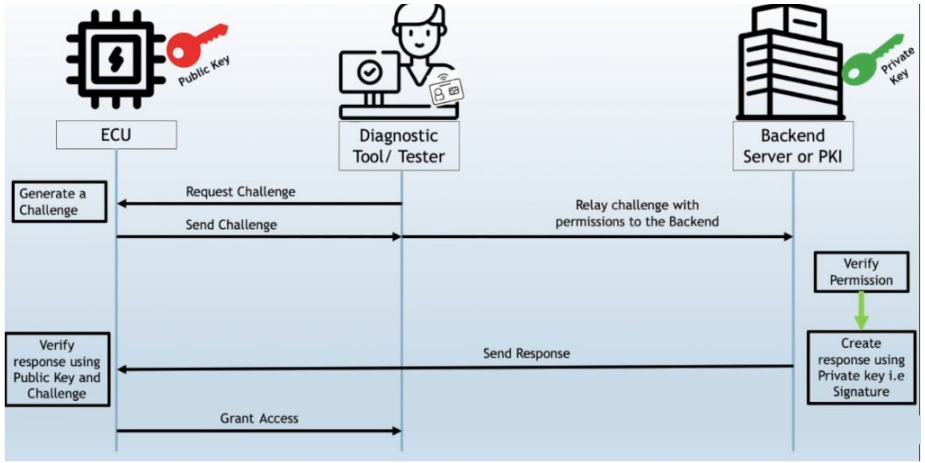
Purpose of Security Access

- Ensures Confidentiality and Integrity:
  - By using cryptographic signatures and challenges, it ensures that communications between the ECU and the diagnostic tool cannot be intercepted or tampered with.
- Prevents Unauthorized Access:
  - The service helps prevent malicious actors or unauthorized diagnostic tools from accessing or modifying ECU data or settings.
- Secures Sensitive Operations:
  - It is essential for operations that involve access to restricted or sensitive vehicle information, such as programming, parameter adjustments, and more.



Key Process in UDS Service 27

- **ECU Generates Challenge:**
  - The ECU generates a random challenge that is sent to the diagnostic tool or tester when requested.
- **Relay Challenge to Backend:**
  - The diagnostic tool forwards the challenge to the backend server or PKI (Public Key Infrastructure), which holds the Private Key.



- **Backend Verifies Permission:**
  - The backend server verifies whether the diagnostic tool has permission for access, ensuring that only authorized tools are allowed to interact with the ECU.
- **Create Response Using Private Key:**
  - If permissions are valid, the backend server uses its Private Key to encrypt the challenge, generating a digital signature or response.
- **Send Response:**
  - The signed response is sent back to the ECU via the tester.
- **Verify Response:**
  - The ECU verifies the response using the Public Key corresponding to the backend's private key and the original challenge.
  - If the response is verified successfully,
    - Access is granted to the restricted ECU functionalities.

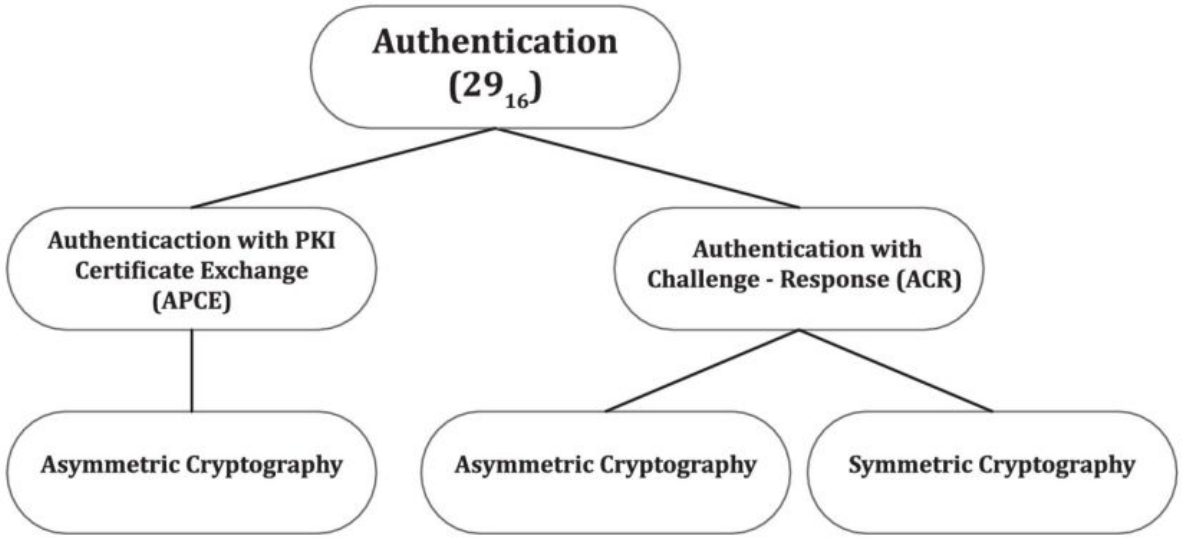
Secure Authentication in Automotive Diagnostics

- **Purpose**
  - A means for the client to prove its identity.
  - Allows a client to access data or diagnostic services, which have restricted access due to security or safety reasons.
- **Use Cases**
  - When using diagnostic services for downloading/uploading routines or data into a server.
  - When reading specific memory locations from a server.
- **Risk of not having Authentication**
  - Improper routines or data downloaded into a server could potentially damage the electronics or other vehicle components.
  - Risk the vehicle's compliance to emission, safety, or security standards.
  - Data security might be violated when retrieving data from a server.

Security Concepts supported by Authentication Service

- **Authentication with PKI Certificate Exchange (APCE)**
  - Uses asymmetric cryptography i.e different sets of certificates and corresponding keys shall be available at both sides client and server.
  - Uses a certificate format
- **Authentication with Challenge-Response (ACR)**
  - Uses Either Asymmetric Cryptography or Symmetric Cryptography
  - In case of using asymmetric cryptography, the client private key shall be present in the client and the client public key shall be present in the server.
  - In case of using symmetric cryptography, a symmetric key shall exist and shall be pre-shared between the client and the server.
  - Uses software authentication tokens.

Overview over the Authentication service security concepts



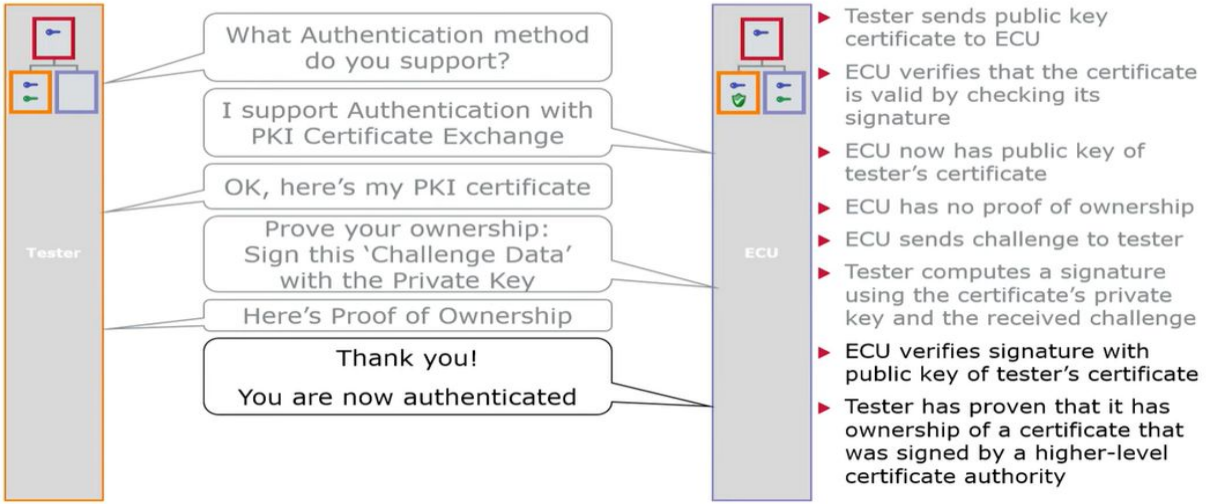
Authentication with PKI Certificate Exchange (APCE)

- Variant 1: Unidirectional authentication
  - Client sends its certificate client.
  - Server verifies the certificate client.
  - Server creates a challenge server.
  - Server sends the challenge server.
  - Client calculates proof of ownership client by building an appropriate authentication token whose content shall be signed.
  - Client sends the proof of ownership client.
  - Server verifies the proof of ownership client with the public key from the received certificate client.
  - The server grants access to diagnostic objects according to access rights.
  - The server responds that the authentication was successful.

Authentication with PKI Certificate Exchange (APCE)...

Authentication

Service 29_{16}: Authentication with PKI certificate exchange



- Tester sends public key certificate to ECU
- ECU verifies that the certificate is valid by checking its signature
- ECU now has public key of tester's certificate
- ECU has no proof of ownership
- ECU sends challenge to tester
- Tester computes a signature using the certificate's private key and the received challenge
- ECU verifies signature with public key of tester's certificate
- Tester has proven that it has ownership of a certificate that was signed by a higher-level certificate authority

Secure Access Vs Secure Authentication

Aspect	Security Access(Service 27)	Authentication(Service 29)
Authorization	Based on security levels (e.g., 1, 3, 5, …)	Based on user roles (e.g., supplier, development)
Cryptography	Symmetric cryptography (can be simple)	PKI (Public Key Infrastructure) certificate-based
Protected Services	Only critical services (e.g., Write, IO Control, Software Update) are protected	Most services are protected except basic ones (e.g., session change, Read OBD)
AUTOSAR State Machine	Single Security Access Level state machine for all physical connections	Authentication state machine exists per physical layer connection (e.g., CAN or Ethernet)
Connection Type	Independent of communication medium (CAN or DoIP)	Authentication varies per connection type

Why Authentication is needed when we have Secure Access?

- **Drawback of Secure Access**
  - Service 27 operates with a single security access level state machine that applies to all physical connections (like CAN, Ethernet, etc.).
  - This means that the same security protocol or authentication mechanism is used for all connections, regardless of the underlying physical layer or communication channel.
- **Potential Risks:**
  - Limited Flexibility:
    - Different physical layers (CAN, Ethernet, etc.) may have different security requirements, but a single state machine treats them uniformly.
  - Higher Risk of Compromise:
    - An attack on one layer could potentially compromise the entire security.

- **Benefits of Service 29's Approach:**
  - Having a separate state machine for each layer allows for tailored security measures for each physical connections.
  - If one layer (e.g., CAN) is compromised, the authentication mechanism on other layers (e.g., Ethernet) remains secure.

Secure Update

- **What is Secure Update?**
  - A secure update refers to the process of safely transmitting and applying software updates to a system or device while ensuring the integrity, authenticity, and confidentiality of the data involved.
  - The goal of a secure update is to prevent unauthorized access, alterations, or rollbacks to older, less secure software versions.
  - Secure Update will verify the received software image is whether it’s authenticated.
  - Secure Update ensures that only authorized SW can be run on the ECU.
- **Importance of Secure Update**
  - Security Patching
  - Preventing Malware and Attacks
  - Maintaining Functionality
  - Ensuring Trust

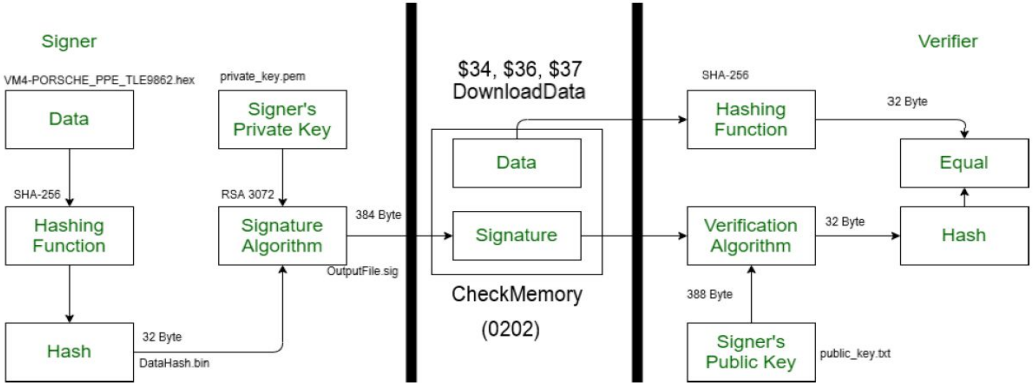
Key components of Secure Update

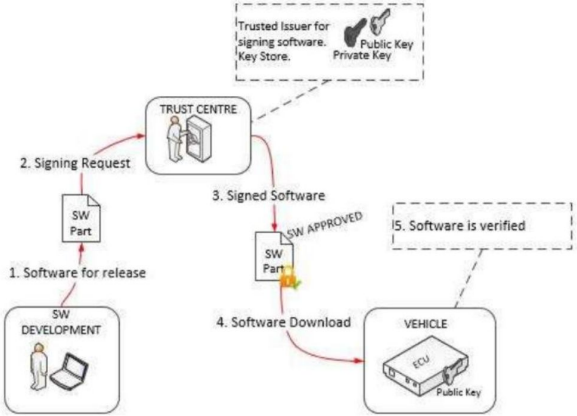
- **Confidentiality**
  - Updates are encrypted using secure algorithms and only decrypted in the secure environment of the receiving device.
- **Integrity**
  - Ensures that the update has not been tampered with during transmission using cryptographic hash functions.
- **Authenticity**
  - Ensures that the update comes from a trusted source using digital signatures where the update package is signed using a private key of a trusted entity and verified on the device using the corresponding public key.

Key components of Secure Update

- **Authorization**
  - Ensures that only devices or users with the correct permissions can initiate or apply updates.
  - Often involves checking permissions or roles before allowing the update process to proceed
- **Secure Transmission**
  - Ensures that the channels used for transmitting updates are secure.
  - Typically uses transport security mechanisms like TLS (Transport Layer Security) to protect the data in transit.
- **Version Control and Rollback Protection**
  - Prevents the re-installation of older firmware versions that may contain known vulnerabilities.
  - Involves mechanisms to check and verify the version of the update to ensure it is the latest and most secure.

Secure Update Model





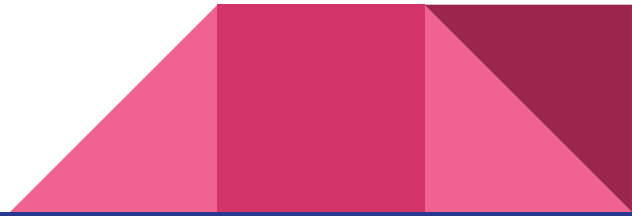
Secure Boot

- **Purpose**
  - Secure boot ensures that any code executed by the CPU is authentic, thereby preventing the CPU from executing unwanted programs (i.e., malicious or integrity-less instructions) thereby protecting the vehicle's systems from potential security threats.
  - Secure Boot aims to guarantee the Authenticity and Integrity of the software by verifying them every time the system boots up.
- **Root of Trust (RoT)**
  - A root of trust consists of code, usually implemented in hardware or immutable software, that are used to perform the initial firmware verification steps.
  - The root of trust shall also consist of a cryptographic key used to verify the software components at startup.
  - RoTs are usually stored in the non rewritable area or the secure area.
  - secure area refers to access prohibited without authentication.

Steps involved in Secure Boot implementation

- **Power On:**
  - The ECU powers on, initiating the boot process.
- **Bootloader Execution:**
  - The bootloader begins execution immediately after power-up.
- **Initialize Root of Trust:**
  - The bootloader initializes the Root of Trust (RoT) and any secure hardware components necessary for secure operations.
- **Firmware Signature and Hashing:**
  - The manufacturer signs the hashed firmware image using a private key. This digital signature is stored alongside the firmware in the ECU's flash memory.
  - A cryptographic hash of the entire firmware image is generated.
  - Firmware image is generated using a hash function like SHA-256
  - Firmware image is then stored in a secure area of the ECU's memory.

- **Retrieve Public Key:**
  - The bootloader retrieves the public key from secure storage within the ECU, ensuring it is used for verification.
- **Signature Verification:**
  - The bootloader extracts the digital signature from the firmware image.
  - It verifies this signature using the retrieved public key to ensure the authenticity of the firmware image.
- **Compute Hash of Firmware:**
  - The bootloader computes a hash of the firmware image using the same cryptographic hash function (e.g., SHA-256).
- **Integrity Check:**
  - The bootloader compares the computed hash with the known good hash stored in secure memory, ensuring the integrity of the firmware image.



- **Decision Point:**
  - If both checks (signature verification and integrity check) pass:
    - The bootloader proceeds to load the validated firmware into RAM and transfers control to it.
  - If either check fails:
    - The bootloader can enter a safe mode or execute recovery procedures to prevent compromised firmware from running.
- **Error Logging:**
  - An error for the failed secure boot process can be logged for diagnostics.
  - The Device Identifier (DID) can be used to read the cause of failure for further analysis.
- **NOTE:**
  - Asymmetric Secure Boot
    - Employs off-board private keys and integrity-protected public keys on the ECU.
    - Uses signature and hashing algorithms for verification.
  - Symmetric Secure Boot
    - Employs securely stored symmetric keys
    - Uses MAC verification for firmware integrity.

Benefits of Secure Boot:

- Enhanced Security:
  - Blocks unauthorized software, safeguarding against cyber threats.
- Safety Assurance:
  - Ensures only tested, certified software runs, maintaining system reliability.
- Regulatory Compliance:
  - Helps manufacturers adhere to industry standards for automotive cybersecurity.
- Rollback Protection:
  - Prevents the installation of older, potentially vulnerable firmware versions.
- Runtime Monitoring:
  - Continuously checks firmware integrity to detect unauthorized changes.
- Tamper Resistance:
  - Uses secure hardware to protect sensitive keys and firmware from attacks.

Secure Storage

- **What is Secure Storage?**
  - Secure storage refers to methods and technologies designed to protect sensitive data and cryptographic materials from unauthorized access, manipulation, and exploitation.
  - It ensures that data remains confidential, authentic, and integral throughout its lifecycle.
- **Key Objectives of Secure Storage**
  - Confidentiality:
    - Protects data from unauthorized read operations, ensuring that sensitive information remains private and accessible only to authorized entities.
  - Integrity and Authenticity:
    - Safeguards against unauthorized modifications, such as overwriting or swapping data. This ensures that stored data remains unchanged and verifiable as legitimate.

- Data Freshness:
  - Protects against replay attacks by ensuring that the system always returns the most recent version of the data, preventing the use of outdated or stale information.
- Protection of Cryptographic Materials:
  - Ensures that cryptographic algorithms and their operations are stored and executed securely within a Hardware Security Module (HSM) or a secure enclave, preventing exposure to threats.
- Protection of Cryptographic Materials:
  - Ensures that cryptographic algorithms and their operations are stored and executed securely within a Hardware Security Module (HSM) or a secure enclave, preventing exposure to threats.

SECURE DEBUG

- **What is Secure Debug?**
  - Secure debug refers to a set of security measures designed to control and protect access to debug features in microcontrollers and embedded systems.
  - These measures help ensure the integrity and confidentiality of data stored within the microcontroller while preventing unauthorized access that could lead to exploitation.
- **Key Objectives of Secure Debug**
  - Access Control:
    - Ensures that only authorized personnel or systems can access debug features, protecting sensitive information from unauthorized manipulation or extraction.
  - Data Integrity and Confidentiality:
    - Maintains the integrity of the microcontroller's data.
    - Prevents unauthorized disclosure.
    - Ensures that debugging doesn’t compromise sensitive information.

- Password Unlock:
  - The JTAG (Joint Test Action Group) interface, commonly used for debugging, is protected by a unique 256-bit password specific to the device. This prevents unauthorized users from accessing debug functions.
- Signed Unlock:
  - Secure debug can be facilitated through a commercially signed debug certificate.
  - This certificate verifies the legitimacy of the request to unlock debug features.
  - The unlocking process can be tied to the ECU's unique hardware ID, ensuring that only the correct hardware can be debugged.
- Challenge Unlock:
  - A challenge/response protocol is implemented to authenticate access to debug features:
    - Challenge Seed Generation: The ECU uses a True Random Number Generator (TRNG) to generate a challenge seed.
    - Response Generation: Using the AES-128 encryption algorithm, the debugger generates a response based on the challenge seed and a shared public key.
    - This private key is known only to the ECU and the secure server.
    - Authentication: If calculated response matches the expected value,
      - Access to the debug features is authenticated.
      - The hardware debug connection is unlocked.

SECURE HARDWARE PROTECT

- **What is Secure Hardware Protect?**
  - To protect the MCU pins from tampering to prevent physical attacks, access shall be disabled either by physically removing them from the PCB or by logically controlling their access.
  - For Example: Unused I/O Pins, SPI lines, etc.
- **Measures for Ensuring Hardware Protection**
  - JTAG Port Concealment:
    - Ensure that no traces of JTAG ports or probing points are exposed on the PCB.
  - Disable Unused Communication Interfaces:
    - Configure the PCB to disable any communication interfaces that are not utilized in the design.
  - Sensitive Data Line Protection:
    - Integrate sensitive data lines into deeper layers of a multilayer PCB.
    - This enhances protection against probing.

- JTAG Port Concealment:
  - Ensure that no traces of JTAG ports or probing points are exposed on the PCB.
- Disable Unused Communication Interfaces:
  - Configure the PCB to disable any communication interfaces that are not utilized in the design.
- Sensitive Data Line Protection:
  - Integrate sensitive data lines into deeper layers of a multilayer PCB.
  - This enhances protection against probing.
- Disable Unused Connectors:
  - Physically remove unused connectors or connector footprints from the PCB to prevent unauthorized access.
- Logical Disabling of Unused Ports
  - Disable any unused I/O ports and communication interfaces.
  - This can be done through software or firmware settings.

SECURE SOFTWARE DEVELOPMENT

- Secure coding practices helps in eliminating the known software vulnerabilities like buffer overflow, uninitialized vector, SQL injection, etc,
- Secure coding best practices shall be implemented irrespective of the coding language.
- The software developer shall follow secure coding guidelines like MISRA, Cert C coding, etc. to reduce vulnerabilities in the software running in the system.
- Static code analysis, vulnerability scanning and warnings shall be fixed.
- All unnecessary software packages including any development binaries shall be removed on production.

References

https://www.autosar.org/fileadmin/standards/R22-11/CP/AUTOSAR_SRS_CryptoStack.pdf

https://www.autosar.org/fileadmin/standards/R22-11/CP/AUTOSAR_SWS_CryptoDriver.pdf

https://www.autosar.org/fileadmin/standards/R22-11/CP/AUTOSAR_SWS_CryptoServiceManager.pdf