

**CRYPTOGRAPHY IN ADAPTIVE AUTOSAR**

Contents

[**Cryptography 3**](#_Toc146614747)

[**Definitions of Cryptography Keywords 3**](#_Toc146614748)

[**Adaptive Architecture of cryptography 3**](#_Toc146614749)

[**cryptography Architecture 4**](#_Toc146614750)

[**Three main sub division of cryptography 4**](#_Toc146614751)

[**Crypto provider 4**](#_Toc146614752)

[** Random number Generator 4**](#_Toc146614753)

[** Hashing 5**](#_Toc146614754)

[** Key Derivation function 6**](#_Toc146614755)

[** Message authentication code 8**](#_Toc146614756)

[** Authenticated Encryption 10**](#_Toc146614757)

[** Symmetric Encryption 11**](#_Toc146614758)

[** Asymmetric encryption 15**](#_Toc146614759)

[** Key wrapping 16**](#_Toc146614760)

[** Key Encapsulation mechanism 17**](#_Toc146614761)

[** key Agreement 18**](#_Toc146614762)

[** Digital Signature 19**](#_Toc146614763)

[**Key Storage 20**](#_Toc146614764)

[**Public Key 20**](#_Toc146614765)

[**Private Key 20**](#_Toc146614766)

[**Difference between the Public Key and Private key 21**](#_Toc146614767)

[**X.509 certificates 22**](#_Toc146614768)

[**Cryptography Algorithms 24**](#_Toc146614769)

[**Why use cryptography in automotive? 24**](#_Toc146614770)

[**About Adaptive Cryptography Project 25**](#_Toc146614771)

[**Multiple Testing’s of Adaptive Autosar source code: 27**](#_Toc146614772)

[**Functional Testing 27**](#_Toc146614773)

[**Unit Testing 28**](#_Toc146614774)

[**Integration Testing 30**](#_Toc146614775)

[**Adaptive AUTOSAR Configuration** 31](#_Toc146614776)

[**1.** **Application Design** 31](#_Toc146614777)

[**2.** **Service Instance Manifest** 32](#_Toc146614778)

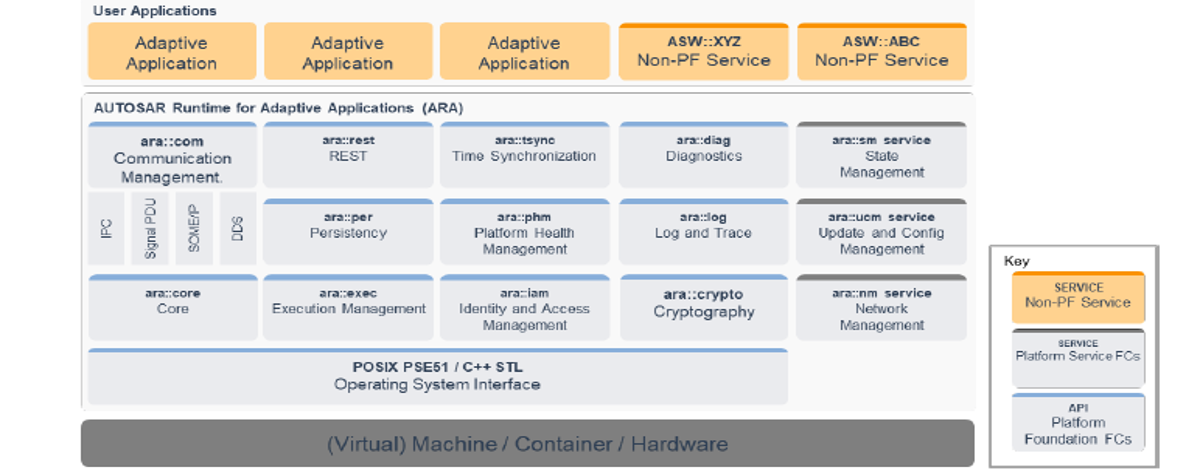
# **Cryptography**

**cryptography** is a converting a plain readable data into an unreadable data and again retransforming that message into its original form and It provides Confidentiality, Integrity, and Accuracy.

## **Definitions of Cryptography Keywords**

* **Encryption:** The process of changing the plaintext into the ciphertext is referred to as encryption.
* **Decryption:** The process of changing the ciphertext to the plaintext that process is known as decryption.
* **Plain Text:** This is the message which is readable or understandable. This message is given to the Encryption algorithm as an input.
* **Cipher Text:** The cipher text is produced as an output of Encryption algorithm. We cannot simply understand this message.
* **Encryption Algorithm:** The encryption algorithm is used to convert plain text into cipher text.
* **Decryption Algorithm:** It accepts the cipher text as input and the matching key (Private Key or Public key) and produces the original plain text.

## **Adaptive Architecture of cryptography**



## **cryptography Architecture**

****

## **Three main sub division of cryptography**

1. **Crypto provider** – All encryption and decryption algorithm functionalities
2. **Key Storage** – retrieving and storing of all crypto keys
3. **Certificates** – X.509 certificates parsing, verification and retrieval of public key.

## **Crypto provider**

functionalities of Crypto providers

### **Random number Generator**

In cryptography, RNG stands for Random Number Generator. An RNG is a fundamental component used to **generate random and unpredictable numbers for various cryptographic purposes**. These random numbers play a critical role in ensuring the security and confidentiality of cryptographic algorithms and protocol.

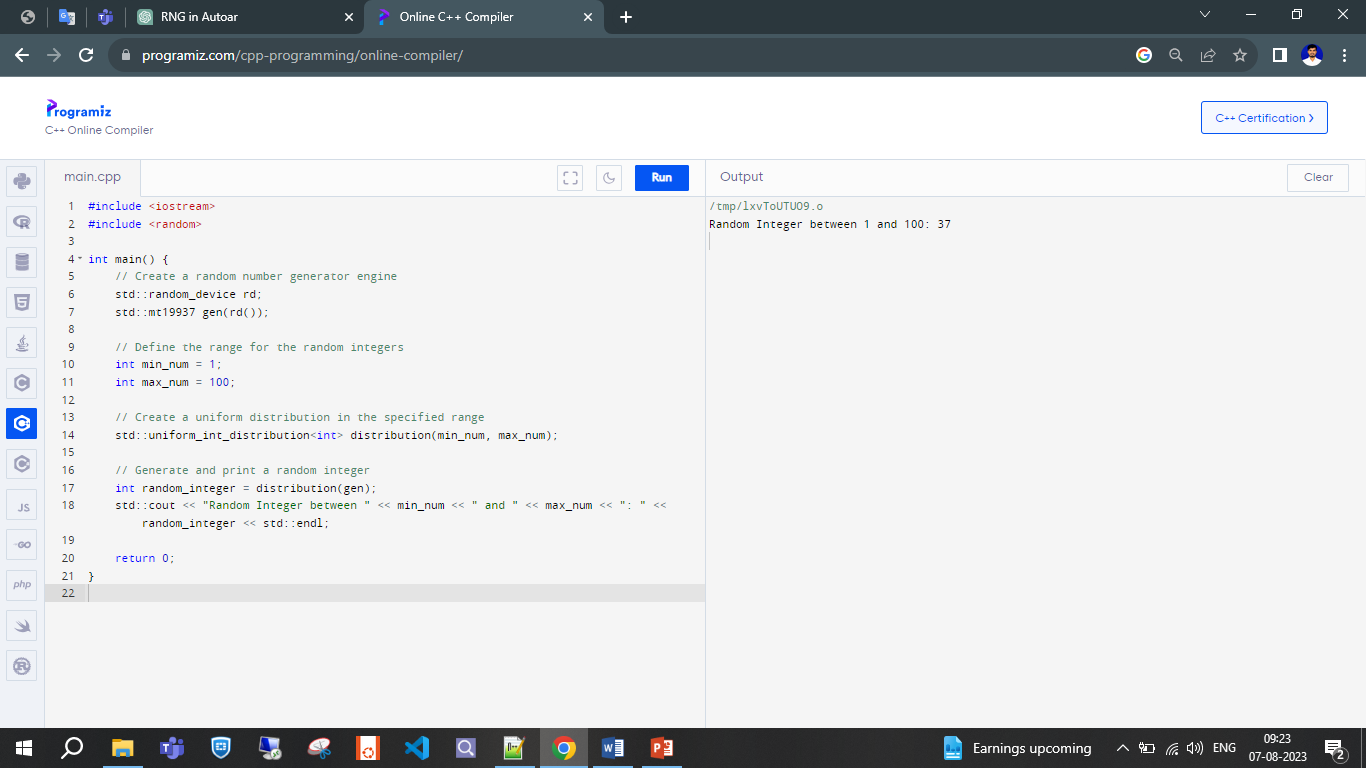
**Cryptographic RNGs are used in a variety of applications within cryptography, including:**

* Generating cryptographic keys for encryption and decryption.
* Creating Initialization Vectors (IVs) for symmetric encryption modes.
* Generating Nonce for secure communications and cryptographic protocols.
* Creating random salts for password hashing and storage.
* Producing random values for digital signatures and secure authentication.

**RNG Interface methods:**

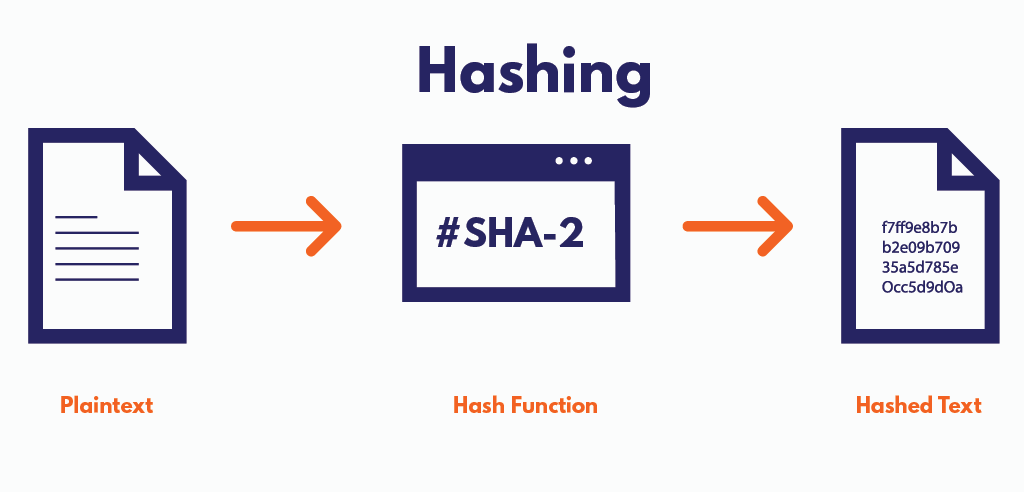
1. **bool Seed (ReadOnlyMemRegion seed):** set the internal state of the RNG using the provided seed(ROMR).
2. **bool Seed (const SecretSeed &seed):** set the internal state of the RNG using the provided seed(ss).
3. **bool SetKey (const SymmetricKey& key):** a SymmetricKey with the key used as seed value.
4. **bool AddEntropy (ReadOnlyMemRegion entropy):** Update the internal state of the RNG by mixing it with the provided additional entropy.

**Let’s see example of RNG:**



### **Hashing**

* A hash function is a mathematical function **that converts a numerical input value into another compressed numerical value.** The input to the hash function is of arbitrary length but output is always of fixed length.it is return hash value.
* A hash algorithm is used for various purposes, including secure communication, data integrity verification, password storages and authentication within automotive electronic systems. Hash algorithms are crucial for ensuring the security and reliability of communication between different components and ECUs (Electronic Control Units) in a vehicle's network.
* Some commonly used hash algorithms are **SHA-1** (Secure Hash Algorithm 1), **SHA-256** (Secure Hash Algorithm 256) and **SHA-3** (Secure Hash Algorithm 3).



**Hashing interface methods:**

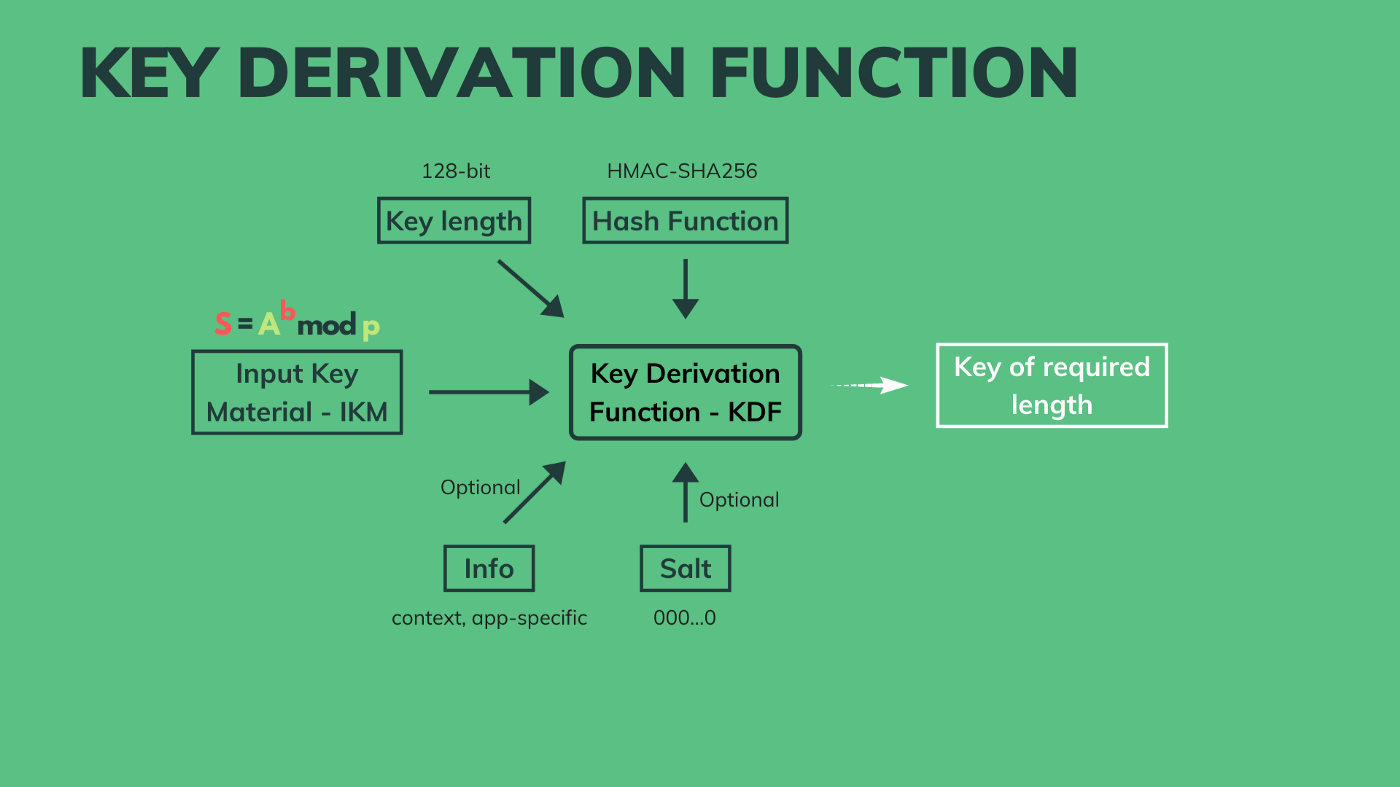
1. **Void Start (ReadOnlyMemRegion iv):** Initialize the context for a new data stream generation.
2. **Void Update (ReadOnlyMemRegion in) :** Update the digest calculation context by a new part of the message.
3. **Byte Finish () :** Finish the digest calculation and optionally produce the "signature" object.
4. **Byte GetDigest (std::size\_t offset = 0):**it is calculation the total size of hash value and return.

**Let’s see example of hash:**



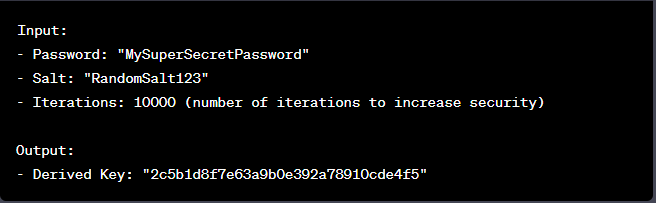
### **Key Derivation function**

* KDF stands for "Key Derivation Function." It is a cryptographic function used **to derive one or more secret keys from a single secret value or a password**. KDFs are particularly useful when you need to generate cryptographic keys from a user's password or other low-entropy secret, while also incorporating additional security measures like salting and iterating to defend against brute-force and dictionary attacks.
* One common scenario where KDFs are used is in password-based encryption or authentication. In such cases, you don't want to directly use a user's password as an encryption or authentication key because passwords are often not strong enough and can be susceptible to various attacks. Instead, a KDF is applied to the password to derive a more secure key.
* Key Derivation Function (KDF) algorithms are used in cryptography **PBKDF2** (Password-Based Key Derivation Function 2) and **HKDF** (HMAC-based Key Derivation Function).

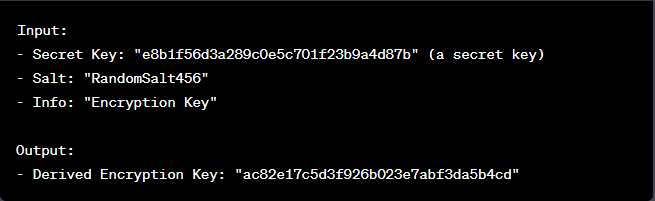


**Let’s see simple examples of KDF algorithms:**

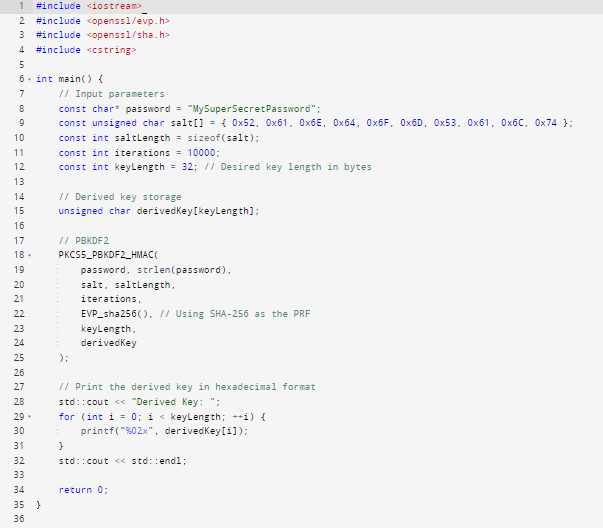
Here's example using the PBKDF2 (Password-Based Key Derivation Function 2) algorithm:



Here's another example using the HKDF (HMAC-based Key Derivation Function) algorithm:



**Let’s see example of KDF:**



### **Message authentication code**

A Message Authentication Code (MAC) is a cryptographic technique used to verify the integrity and authenticity of a message.

MAC algorithm is a symmetric key (k) cryptographic technique to provide message authentication for establishing MAC process, the same symmetric key (K) is shared between sender and receiver.

A Message Authentication Code (MAC) algorithms are used in cryptography **HMAC** (Hash-based Message Authentication Code) and **CMAC** (Cipher-based Message Authentication Code).

**Purpose of MAC:**

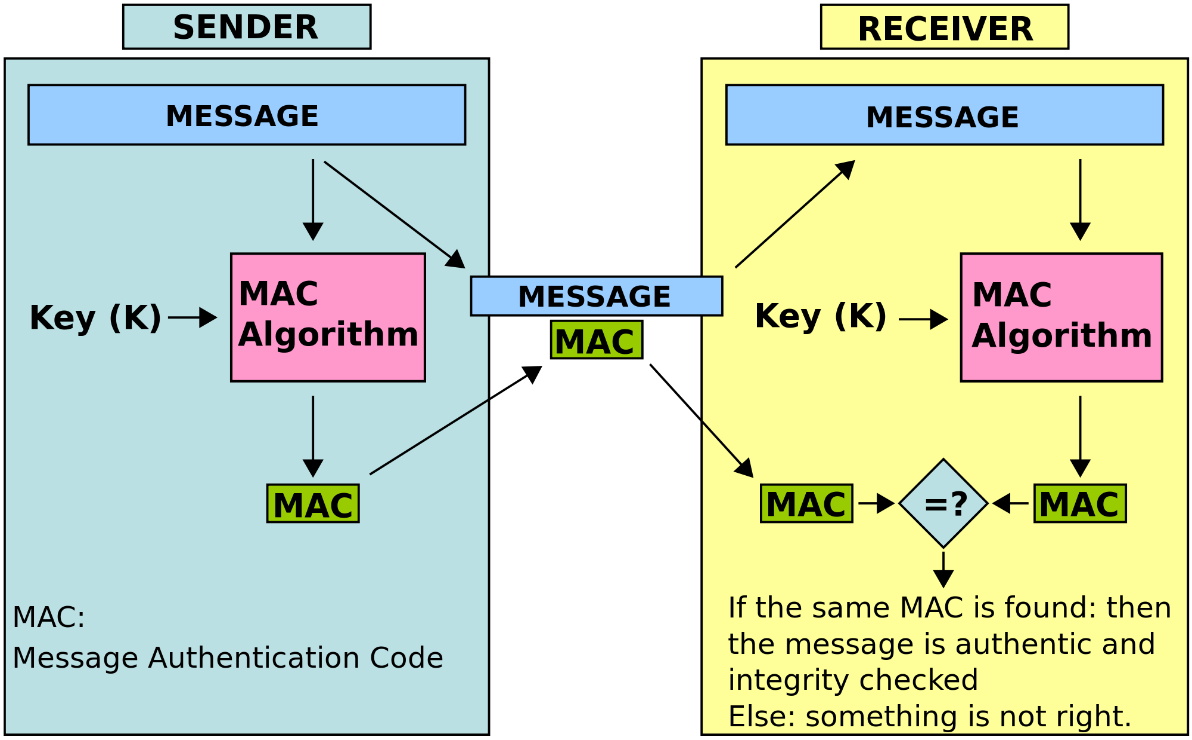
The primary purpose of using MACs is to ensure the following:

* **Integrity:** MACs detect any changes or alterations made to the message during transmission. If the message or data has been tampered with, the MAC verification process will fail.
* **Authenticity:** MACs provide a means to confirm that the message originated from the claimed sender. Since the MAC is generated using a shared symmetric key, only someone with the key can generate a valid MAC for a given message.

**How MACs Work:**

The process of using a MAC involves the following steps:

* **Key Setup**: Both the sender and the receiver must share a secret key that is known only to them.
* **MAC Generation:** The sender uses the symmetric key and the message content as inputs to a MAC generation algorithm. This algorithm produces a fixed-length MAC code.
* **Transmitting:** The sender sends both the original message and the MAC code to the recipient.
* **MAC Verification:** The recipient uses the same symmetric key and the received message to generate a new MAC code. It then compares the calculated MAC with the received MAC. If they match, the recipient knows that the message is both intact and from an authentic sender.



**Use Cases of MAC:**

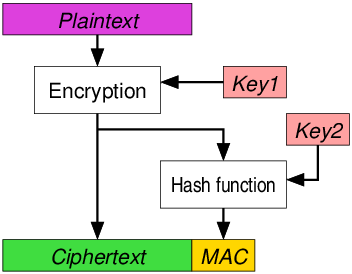
MACs are used in various security protocols and applications, such as:

* Network communication to ensure data integrity.
* Secure messaging to prevent tampering.
* Digital signatures to prove the authenticity of documents.
* Secure storage to verify data integrity over time.

### **Authenticated Encryption**

**Authenticated Encryption** (AE) is a cryptographic technique that combines both encryption and authentication in a single operation. It ensures both the confidentiality of data (encryption) and the integrity/authenticity of the data (authentication).

Authenticated Encryption (AE) algorithms are used in cryptography **AES-GCM** (Advanced Encryption Standard in Galois/Counter Mode) and **AES-CCM** (Advanced Encryption Standard with Counter with CBC-MAC).



**Purpose of AE:**

**Confidentiality:** The plaintext message is transformed into ciphertext, ensuring that unauthorized parties cannot read the original content.

**Integrity/Authenticity:** The ciphertext includes additional information (called authentication tag or MAC - Message Authentication Code) that helps verify the integrity and authenticity of the data. This ensures that the ciphertext hasn't been tampered with or modified by unauthorized parties.

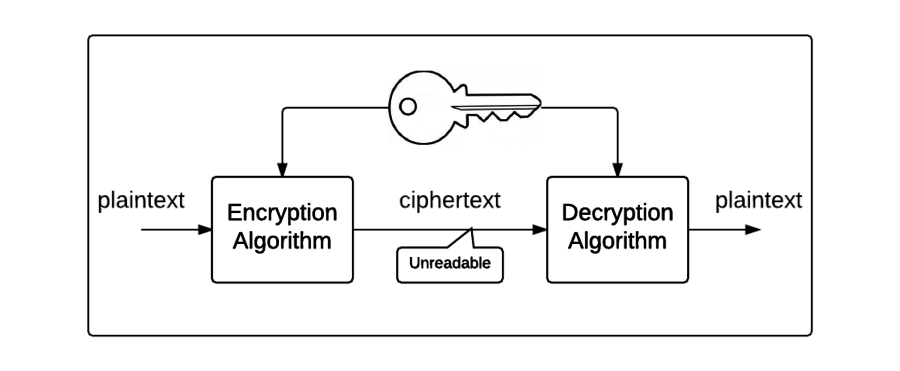
**Use Cases of MAC:**

* Wireless Communication
* Secure Communication Protocols
* Secure Messaging Applications
* Disk Encryption
* VPN (Virtual Private Network)
* Cloud Storage
* Message Authentication
* Digital Signatures

### **Symmetric Encryption**

**Symmetric ciphers** use symmetric algorithms to encrypt and decrypt data. uses the same key to encrypt data and decrypt data.

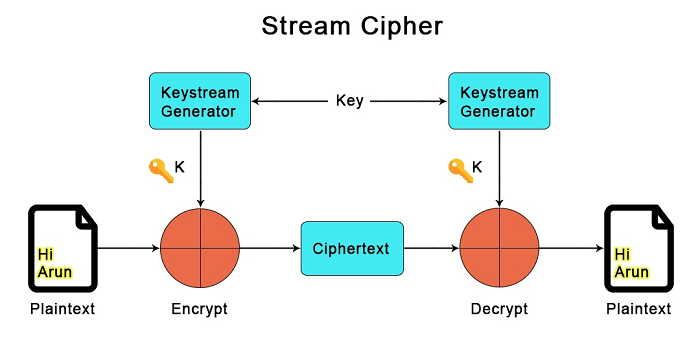
**Example:** Private key.



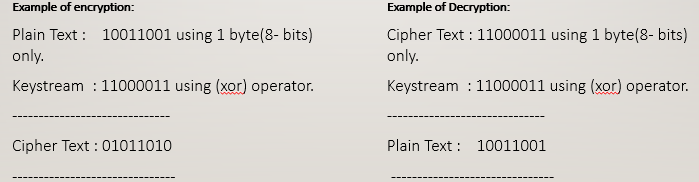
**Sub Functionalities of Symmetric Cipher:**

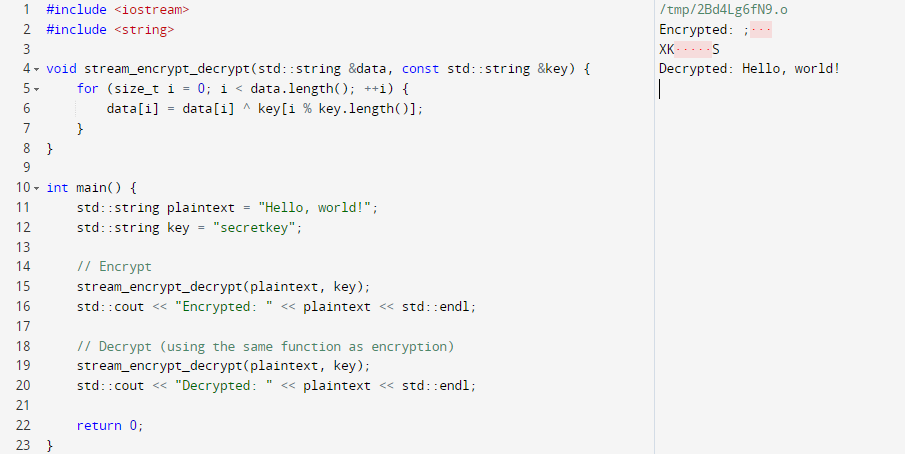
#### **Stream Cipher**

* A **Stream Cipher** is used for Symmetric Key cryptography, same key is used to encrypt and decrypt data.
* In stream cipher, one byte (8 Bites) is encrypted at a time. Initially, a key(k) will be supplied as input to pseudorandom bit generator and then it produces a random 8-bit output which is treated as keystream. The Plaintext will undergo XOR operation with keystream bit-by-bit and produces the Cipher Text.
* Commonly used to protect data in motion, such as encrypting data on the network.
* **Stream Cipher** (SC) algorithms are used in cryptography **CTR-AES-128.**



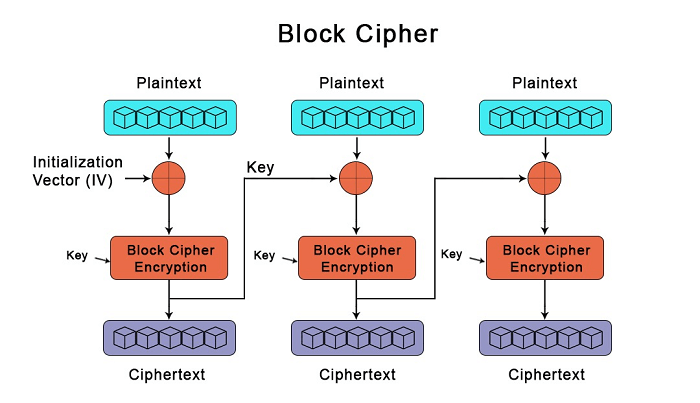
**Let’s see example of Stream Cipher:**



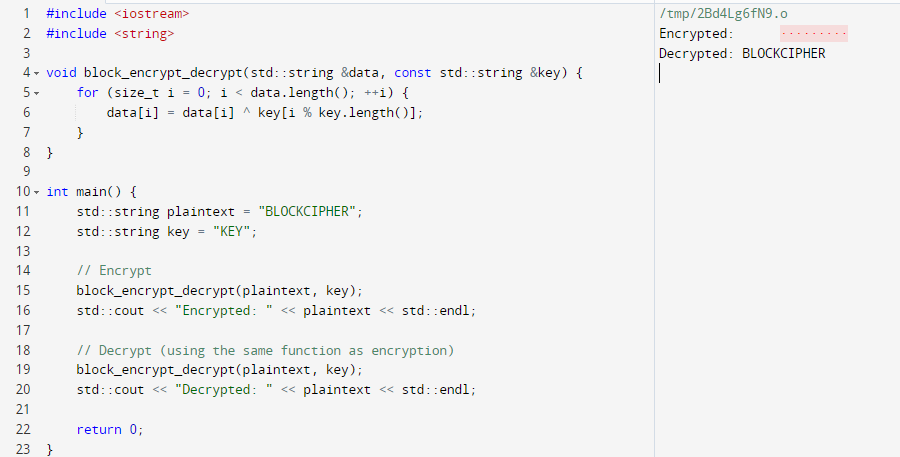


#### **Block Cipher**

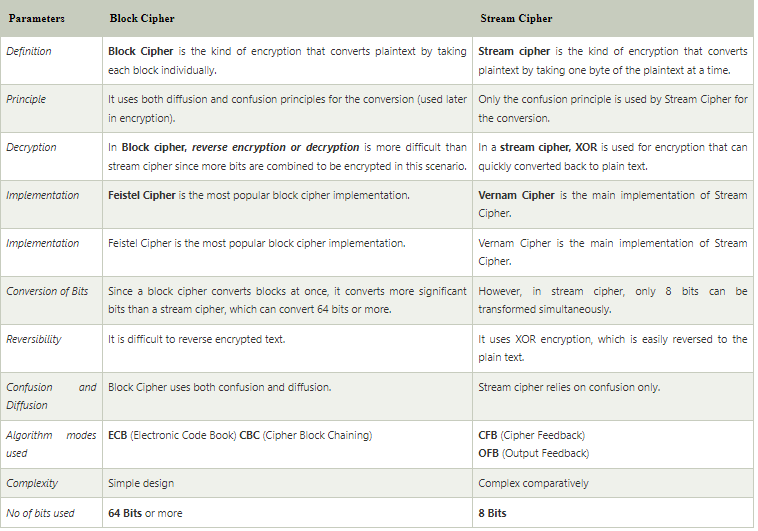
* **Block ciphers** convert data in plaintext into ciphertext in fixed-size blocks.
* The block size generally depends on the encryption scheme and is usually in octaves (64-bit or 128-bit blocks).
* If the plaintext length is not a multiple of 8, the encryption scheme uses **padding** to ensure complete blocks.
* Block ciphers commonly used to protect data at rest, such as on file systems.
* **Stream Cipher** (SC) algorithms are used in cryptography **CBC-AES-128.**



**Let’s see example of Block Cipher:**



#### **Difference between the Stream and Block Cipher:**

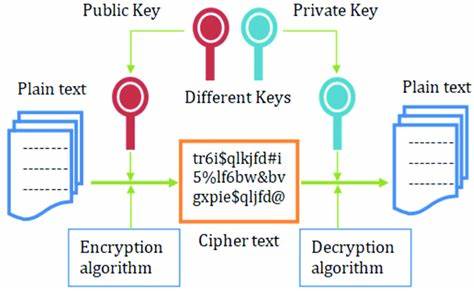


### **Asymmetric encryption**

**Asymmetric encryption**, also known as public-key cryptography, is a cryptographic method that uses a pair of keys to secure communication. These keys consist of a public key and a private key. The public key is widely shared and used for encryption, while the private key is kept secret and used for decryption.

Example: Public Key and Private Key.

Asymmetric encryption algorithms are used in cryptography **RSA-SHA1-PKCSV1\_5.**

****

**How MACs Work:**

Here's how the process generally works.

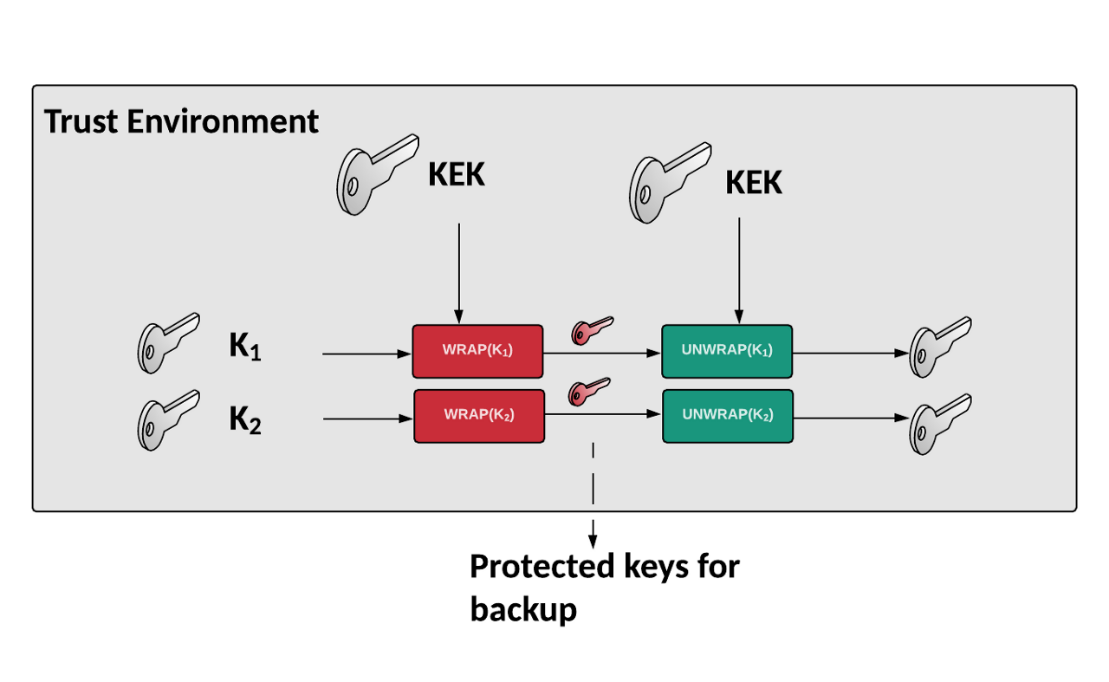
* **Key Generation:** A user generates a key pair: a public key and a private key. The public key is shared openly, while the private key is kept secret.
* **Encryption:** When User A wants to send a confidential message to User B, User A uses User B's public key to encrypt the message. Only User B, who possesses the corresponding private key, can decrypt and read the message.
* **Decryption:** User B uses their private key to decrypt the received encrypted message.
* **Digital Signatures (optional):** Asymmetric encryption can also be used to create digital signatures to verify the authenticity of messages. User A can use their private key to sign a message, and anyone with User A's public key can verify that the message indeed came from User A.

**Use Cases of MAC:**

* Secure Email Communication
* Secure Online Transactions
* Digital Signatures
* Secure File Sharing

### **Key wrapping**

Key wrapping is the process of encrypting one key using another key, in order to securely store it or transmit it over an untrusted channel. Key wrapping may depend on either symmetric or asymmetric cryptography.



Here's a simplified example using a common key wrapping algorithm called the RSA Key Wrapping algorithm:

**Key Wrapping (Encryption):** Let's say we have a data encryption key (DEK) that needs to be securely transmitted. We also have an RSA public key (used for wrapping) and a wrapping key (used to encrypt the DEK). The process Involves**:**

* Generate a random Data Encryption Key (DEK).
* Use the wrapping key to encrypt the DEK.
* Transmit the encrypted DEK along with any necessary metadata (e.g., algorithm identifiers).

**Key Unwrapping (Decryption):** The recipient uses the corresponding RSA private key to decrypt the encrypted DEK:

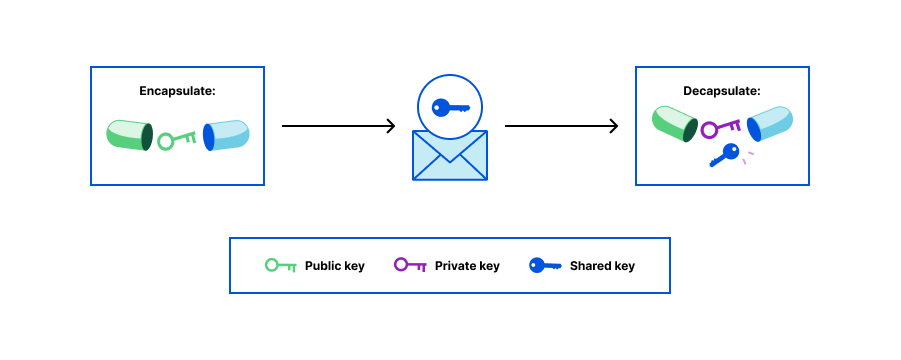
* Use the recipient's private key to decrypt the encrypted DEK.
* Obtain the original DEK for further use in data encryption/decryption.

**Use Cases of Key Wrapping:**

* Key Management and Distribution
* Secure Key Storage
* Key Transport
* Key Migration

### **Key Encapsulation mechanism**

**key encapsulation mechanism** (KEM) is used to secure [symmetric](https://en.wikipedia.org/wiki/Symmetric_cryptography) key material for transmission using [asymmetric](https://en.wikipedia.org/wiki/Asymmetric_cryptography) (public-key) algorithms. It is commonly used in [hybrid cryptosystems](https://en.wikipedia.org/wiki/Hybrid_cryptosystem).



Here's how a typical KEM works:

**Key Generation:**

Each party generates a pair of cryptographic keys: a public key and a private key.

**Key Encapsulation:**

* One party (often referred to as the sender or initiator) generates a random symmetric secret key (session key).
* The session key is used to encrypt the actual message or data.
* The sender also generates a unique value, known as a random seed, for the key encapsulation process.
* The session key is then encrypted using the recipient's public key, and the random seed is combined with the encrypted session key to create a ciphertext.

**Key Decapsulation:**

* The recipient (receiver) of the ciphertext uses their private key to decrypt the encrypted session key and recover the random seed.
* With the random seed, the recipient derives the same session key that the sender had generated.

**Secure Key Exchange:**

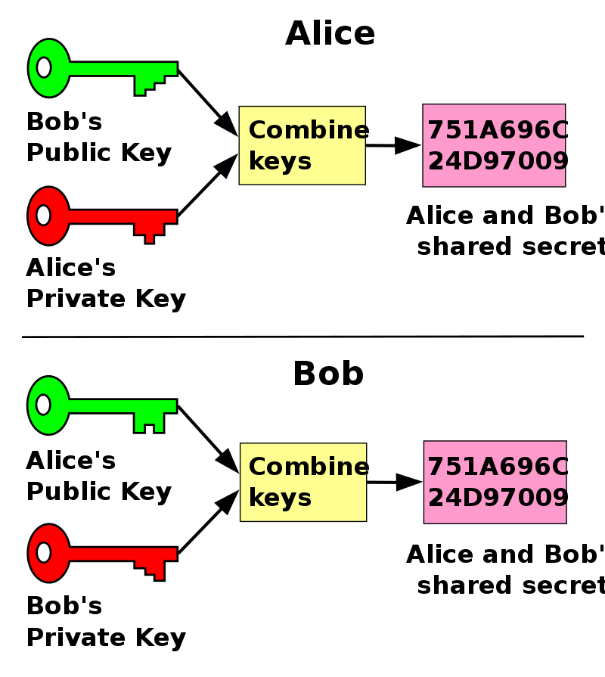
* Both parties now share the same session key without ever transmitting it directly over the insecure channel.
* This session key can be used for subsequent cryptographic operations, such as encrypting and decrypting messages.

**Use Cases of key encapsulation mechanism**:

* Secure Remote Access
* Secure Communication
* Transport Layer Security (TLS)
* File and Data Encryption
* Secure Voice and Video Communication

### **key Agreement**

* Key agreement, also known as key exchange or key establishment. In the presence of untrusted communication networks and dynamic connections, need to avoid exposure and misuse of cryptographic keys. In such situations the **Diffie-Hellman** key exchange scheme is the common used key agreement mechanism.
* **DHKE** is one of the foundational key agreement protocols. Two parties, Alice and Bob, independently generate their public and private key pairs. They then exchange their public keys over the insecure channel. Using their own private keys and the received public keys, both parties compute a shared secret key that is the same at both ends. The security of DHKE relies on the discrete logarithm problem.



**Use Cases of Key agreement:**

* Secure Messaging Applications
* E-commerce and Online Shopping
* Secure File Sharing
* Online Banking and Transactions
* Government and Military Communication

### **Digital Signature**

A **digital signature** is a cryptographic technique used to verify the authenticity, integrity, and non-repudiation of digital messages or documents. It involves using a private key to sign a message or document, creating a unique digital signature that can be verified using the corresponding public key. Digital signatures are essential for ensuring the security of online communication and transactions.

**A digital signature consists of three algorithms:**

**1. Key generation algorithm**

The key generation algorithm selects private key randomly from a set of possible private keys. This algorithm provides the private key and its corresponding public key.

**2. Signing algorithm**

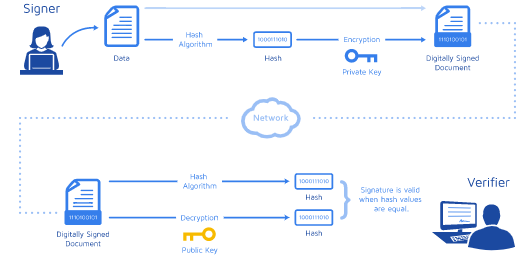
A signing algorithm produces a signature for the document.

**3. Signature verifying algorithm**

A signature verifying algorithm either accepts or rejects the document's authenticity.

**The steps followed in creating digital signature are:**

* Message digest is computed by applying hash function on the message and then message digest is encrypted using private key of sender to form the digital signature. (digital signature = encryption (private key of sender, message digest) and message digest = message digest algorithm(message)).
* Digital signature is then transmitted with the message. (message + digital signature is transmitted)
* Receiver decrypts the digital signature using the public key of sender. (This assures authenticity, as only sender has his private key so only sender can encrypt using his private key which can thus be decrypted by sender’s public key).
* The receiver now has the message digest.
* The receiver can compute the message digest from the message (actual message is sent with the digital signature).
* The message digest computed by receiver and the message digest (got by decryption on digital signature) need to be same for ensuring integrity.



**Use Cases of Digital Signature:**

* Legal documents and contracts
* Sales contracts
* Financial Documents
* Health Data
* Shipping Documents
* Securing Email Communication

## **Key Storage**

The Key Storage Provider is responsible for secure (confidential and or authentic) storage of different type key material (Public key, Private key).

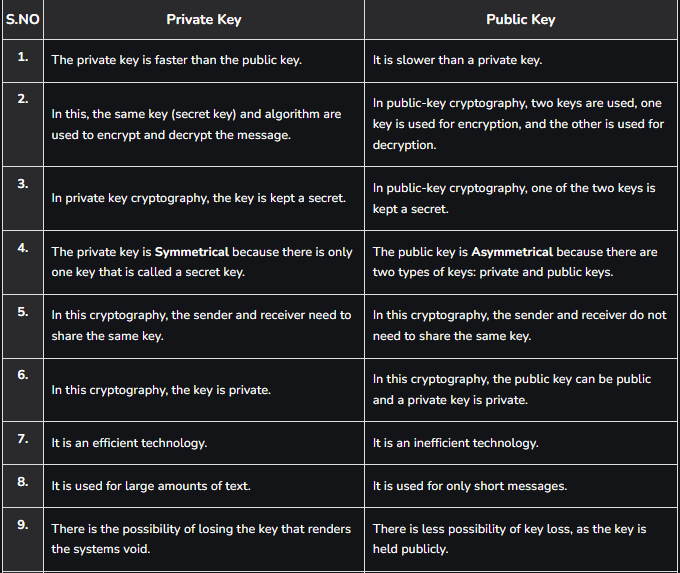
### **Public Key**

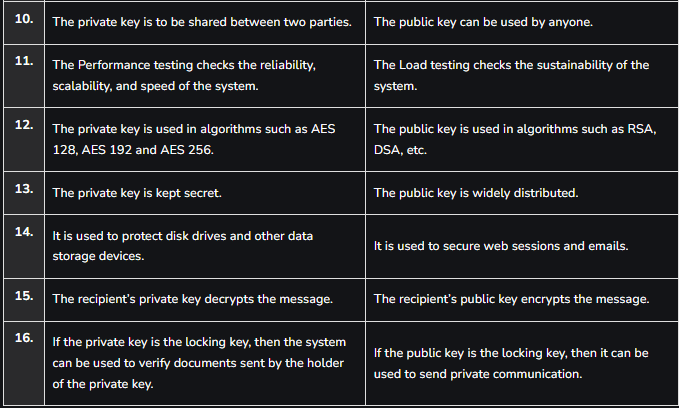
In a Public key, two keys are used one key is used for encryption and another key is used for decryption. In this key is Asymmetric because the One key (public key) is used to encrypt the plain text to convert it into cipher text and another key (private key) is used by the receiver to decrypt the cipher text to read the message.

### **Private Key**

In the Private key, the same key (secret key) is used for encryption and decryption. In this key is symmetric because the only key is copied or shared by another party to decrypt the cipher text. It is faster than public-key cryptography.

### **Difference between the Public Key and Private key**





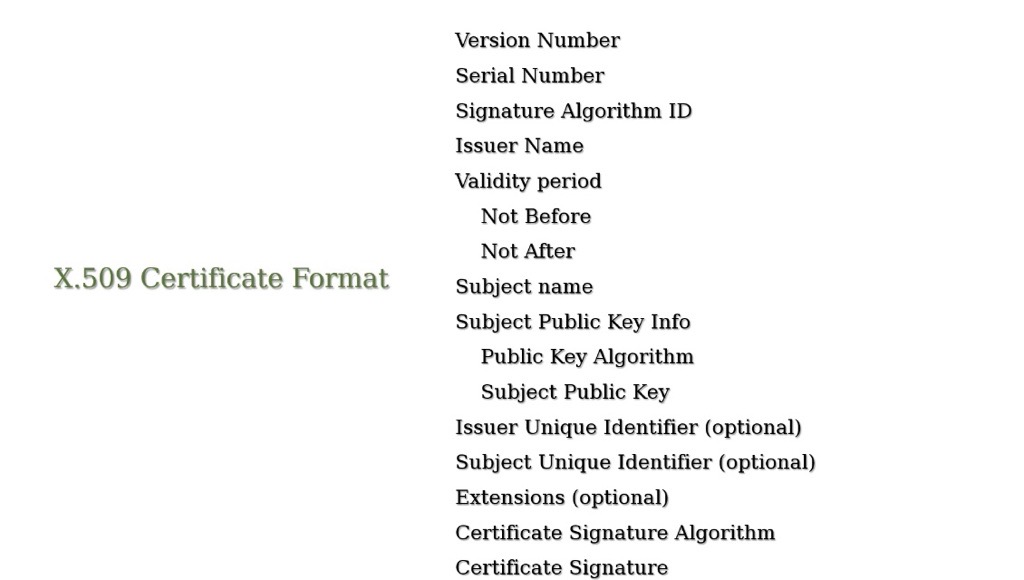
## **X.509 certificates**

* X.509 is a standard that defines the format for public key certificates.
* Encrypted connection like TLS/SSL uses public key encryption to authenticate server and client.
* when the communication starts, the server sends the digital certificate to the client.
* Here digital certificate contains the public key of the certificate authority.
* Digital certificate is electronic document the proves the ownership of a public key.

**The digital certificate contain varies information such as:**

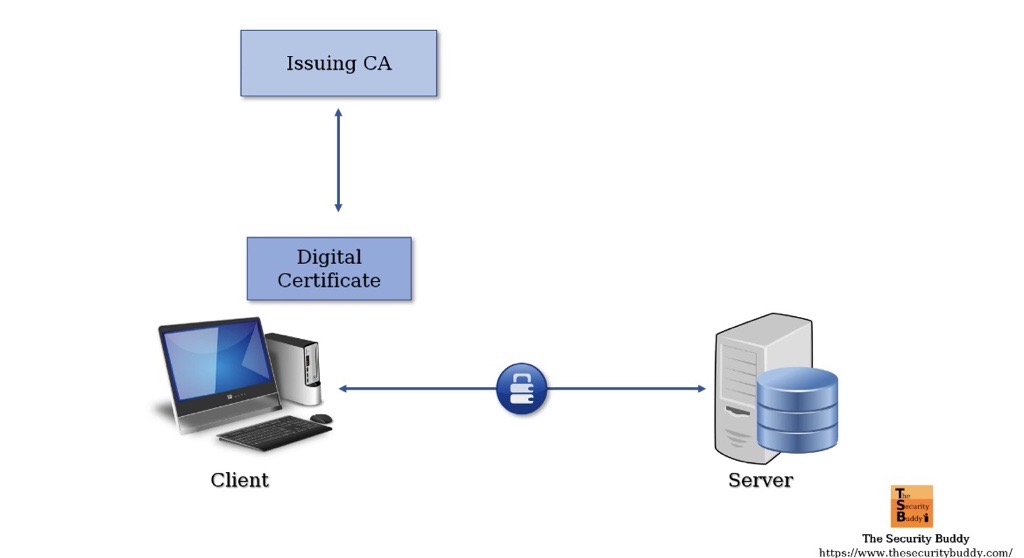


**Flow specific structure of certificate such as:**

****



* Certificate authority issue the digital certificate and the digital certificate contain the digital signature of the certificate authority(CA).
* Server sent the digital certificate to client; the client verifies the digital certificate with issuing CA. If the both public keys successful verification, clients go to the establish the connection.



## **Cryptography Algorithms**

**Sr. No** **Crypto Functionality ctx** **Algorithm Name supported by ctx**

1 RNG CTR-AES+DRBG-RNG

2 Hash SHA-256

3 KDF HMAC-SHA256+PBKDF2

4 MAC HMAC-SHA256

5 Symmetric Block Cipher CBC-AES-128

6 Symmetric Stream Cipher CTR-AES-128

7 Authentication Cipher GCM-AES-128

8 Symmetric Key Wrapper KEY-WRAP-AES-128

9 Asymmetric Encryption RSA-SHA1-PKCSV1\_5

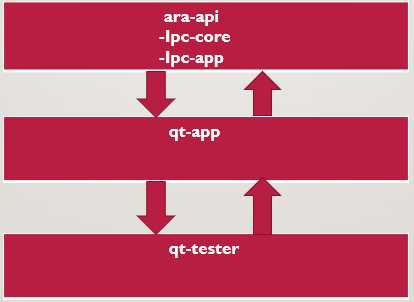
10 Digital Signature ECDSA-SHA256

## **Why use cryptography in automotive?**

* Cryptographic techniques can be used to support the security goals such as authentication, confidentiality, integrity assurance etc., in V2X communications, where a vehicle is communicating with other vehicles (V2V) or roadside infrastructure (V2I).
* Many service providing companies exchange information with vehicles to facilitate the use in terms of location services or other helpful applications. All these authentications are carried out by cryptographic algorithms to ensure the identity of sender and receiver.
* Cybersecurity, within the context of road vehicles, is the protection of automotive electronic systems, communication networks, control algorithms, software, users, and underlying data from malicious attacks, damage, unauthorized access, or manipulation.

## **About Adaptive Cryptography Project**

**Crypto Project block diagram:**



**Working flow of code:**

**1. qt-tester:** we passed the values into function call and it will invoke the function from the qt-app.

**2. qt-app:** it is wrapper application used for function calling to call the actual function from the stack. if stack is not accessible for the everyone. here we are passing arguments in function calling and it reaches the ipc-app side.

**3. Ipc-app:** here communicate the both qt-app and ipc-app in form of ipc mechanism. Which verifies the function and its arguments configure in the qt-app. if the verification is successful call reaches to ipc-core.

**4. Ipc-core:** here actual function implementation and crypto libs are present. In ipc-core checking the values provided in qt-tester to the arguments of respective function and process the return value and communicate to the ipc-app.

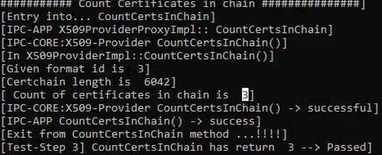
**5. Ipc-app:** return from the ipc-core is verified and share to the qt-app.

**6. qt-app:** return from the ipc-app it verifies and share to the qt-tester.

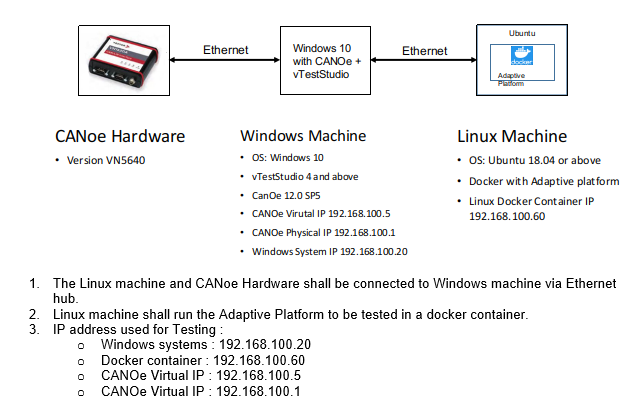
**7. qt-tester:** return value returned from the qt-app is compared with the expected value, if the value is correct it is return the true.

**Example of working flow:**

**Code Flow:**

****

**Canoe Testing:**

****

## **Multiple Testing’s of Adaptive Autosar source code:**

### Functional Testing

Functional testing is a type of software testing that focuses on verifying that a software application or system functions correctly according to its specified requirements. The primary goal of functional testing is to ensure that the software performs its intended functions and features in a way that meets user expectations and business requirements.

**Key points about functional testing include:**

**Requirement Validation:** Functional testing involves comparing the software's actual behavior with its documented requirements, specifications, and user stories to ensure they align.

**Black-Box Testing:** Testers do not need to know the internal code or design of the software. They interact with the application as an external user, input various inputs, and observe the outputs.

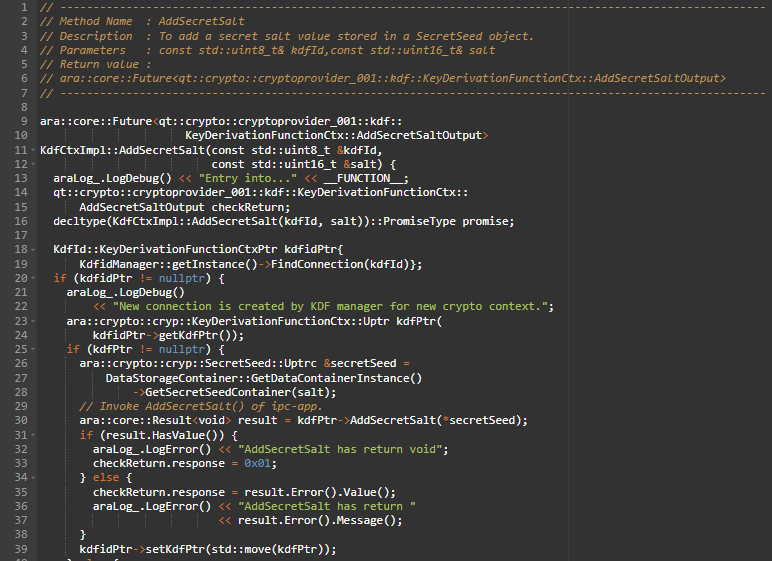
**Test Cases:** Test cases are designed based on functional requirements and cover various scenarios, such as positive and negative test cases, boundary cases, and typical user interactions.

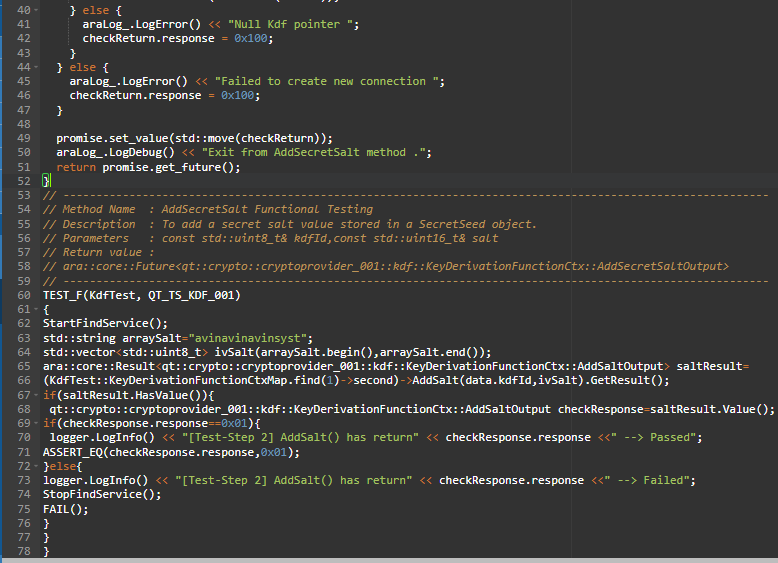
**Types of Functional Testing:** Functional testing can be divided into several subtypes, including smoke testing, sanity testing, regression testing, integration testing, and user acceptance testing (UAT).

**Test Environment:** Functional testing is typically performed in a controlled test environment that replicates the production environment as closely as possible.

**Verification of Features:** Testers verify that each feature, component, or module of the software performs its intended function accurately. This includes testing user interfaces, data processing, calculations, and interactions between different parts of the software.

Let’s see example of Functional Testing.

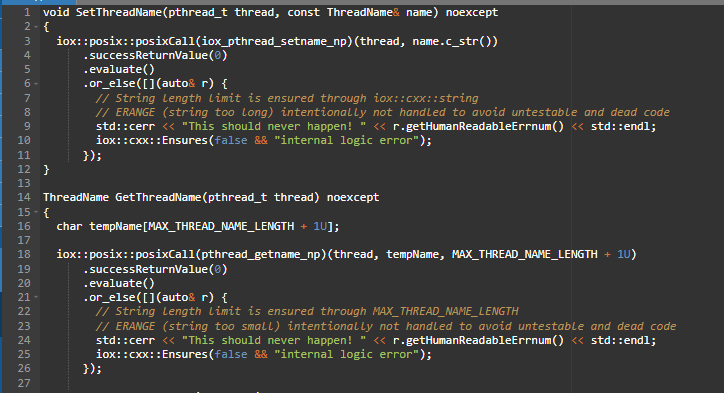


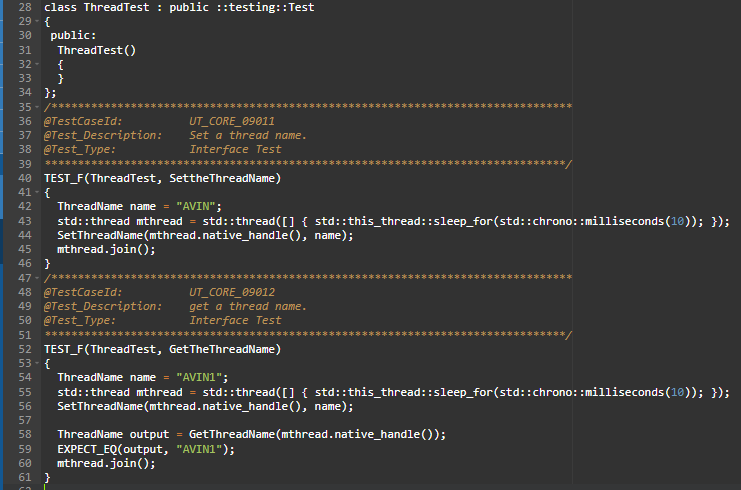


### Unit Testing

Unit testing is a software testing technique where individual components or units of a software application are tested in isolation to ensure that they function correctly. In unit testing, a "unit" typically refers to the smallest testable part of the software, such as a function, method, or class. The primary purpose of unit testing is to validate that each unit of code performs its intended function accurately and independently.

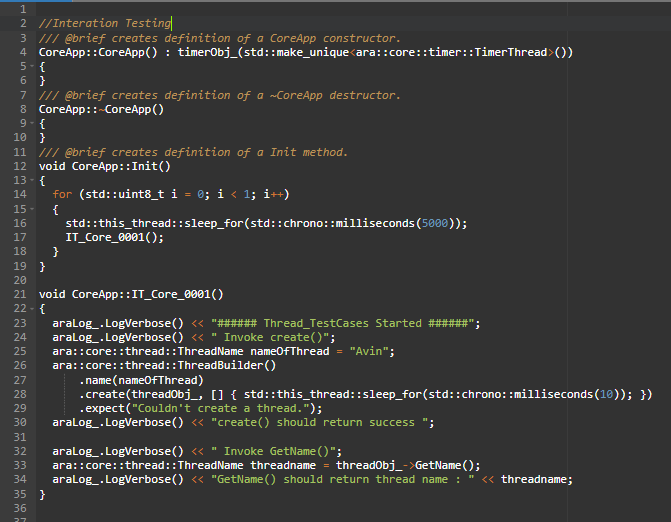
Let’s see example of Unit Testing.

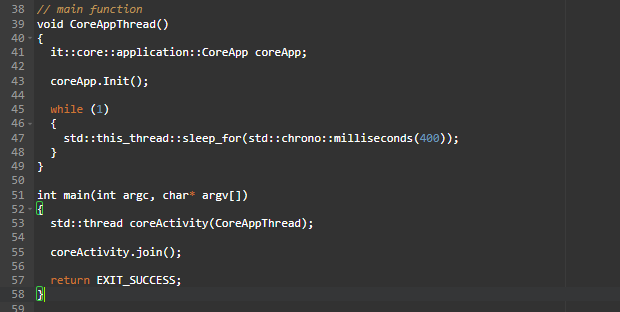




### Integration Testing

Integration testing is a type of software testing that focuses on verifying the interactions and interfaces between different components, modules, or subsystems within a software application. The primary goal of integration testing is to ensure that these integrated parts work together as intended when they are combined.





## Adaptive AUTOSAR Configuration

Here, need to configure qt-app and qt-tester.

**qt-app:**

* qt-app is a sample application, it is providing the p-port.
* qt-app mainly using for canoe testing as well as qt-tester.
* P-Port provide the offering the services.
* Basically R-port not used in canoe, we are using only P-Port based on P-Port inputs canoe generates the R-ports.

**qt-tester:**

* qt-tester is a testing application, qt-tester required the R-port.
* qt-test mainly testing the qt-app applications.
* R-Port provide the finding the services.

if we want to communicate qt-app to qt-tester as well as qt-app to canoe need to configure in ASTRA tool.

**Here, mainly configure 2 containers below given:**

### **Application Design**

* **data types:** configured the all C++ data types. Like **int, float, double, char** etc.
* **application errors:** application errors configurated the all error types. **Ex: invalid input size** and **uninitialized context**, etc.
* **port interface:**

**ARA com interface:**

**service interface:** Here, need to create interface like below**.**

**let 's see interface name was Calculator.**

**Interface name:** **Calculator**

**admin data:** no need to configure

**event:** no need to configure

**fields:** no need to configure

**methods:** Here, we have to configure inside **Calculator** methods like **add, multiple, minus**, etc.

**short name:** **add**

**possible application errors:** **invalid input size**

**arguments:** **variable name** (a, b, c), **type** (int, char, float), **direction**(IN/OUT)

**Ex:** a int IN **and** response int OUT.

**Note:** We can create n number of variables like above.

* **SWC (Adaptive Applications):**

**qt-app-instance:** Inside need to create **P-Port** and this port connect to the **service interface**.

**short name:** **PportCalculator**

**provided interface:** **serviceinterface/Calculator**

**test-application:** Inside need to create **R-Port** and this port connect to the **service interface**.

**short name:** **RportCalculator**

**provided interface:** **serviceinterface/Calculator**

* **Executable:**

Inside executable need to add software components(SWC) like above.

**qt-crypto-instance/Executable**

**short name:** **qt-app-instance**

**application type:** **qt-app-instance**

**test-application/Executable**

**short name:** **test-application**

**application type:** **test-application**

### **Service Instance Manifest**

* **Deployment:** Here using **Someip** protocol in ara::com module having communicate between the

qt-app and qt-tester.

**Someip service interface Deployment:**

**Short name:** **Calculator\_Someip**

**service interface:** **serviceinterface/Calculator**

**service interface ID:** **1** (should be unique)

**someip method deployment:**

Here need to map all calculator interface methods whatever configure in **port interface**.

**short name:** **add**

**method:** **serviceinterface/add**

**method id:** **1**

**transport protocol:** **UDP**

**someip service version:**

**major: 0**

**minor:** **0**

Note: This versions decide by the ara::com module.

* **Service Instance:** Here inside need to create **provided someip** and **required someip**.

**provided someip:** (mainly used for qt-app)

qt-app support the provided someip because of it is offering the services for the qt-tester.

**short name:** **Calculator\_provided**

**Someip service interface Deployment:** **Calculator\_Someip**

**provided service instance ID:** 1 (using this only need to communicate qt-app and qt-tester if both side number diff communication won't happen).

**server config:** **need to configure server services**.

**Required someip:** (mainly used for qt-tester)

qt-tester support the required someip because of it is find services for the qt-app.

**short name:** **Calculator\_required**

**Someip service interface Deployment:** **Calculator\_Someip**

**Required service instance ID:** 1

**client config:** **need to configure client services.**

* **Mapping:** Here **service instance to machine** mapping and **service instance to port** mapping

**service instance to machine mapping:**

**short name:** **CalculatorToMachinemapping**

**communication connector:** **Ethernet** (This is present to **execute manifest**)

**service instance:** **Calculator** (service interface)

**UDP port:** **1000** (should be unique)

**service instance to port mapping:**

**short name:** **CalculatorToportMapping**

**process design:** **ProcessDesign/qt-app-instance**

**process:** **Process/qt-app-instance**

**service instance: Calculator** (service interface)

**target prototype port:** **PportCalculator**

**short name:** **testapplicationCalculatorToportMapping**

**process design:** **ProcessDesign/test-application**

**process:** **Process/test-application**

**service instance: Calculator** (service interface)

**target prototype port:** **RportCalculator**