CyberSecurity Notes

NOTE: . The main function of the **gateway** is communication protocol transactions, for example, transforming FlexRay format data to CAN data, or transforming CAN data to LIN data.

ISO/SAE 21434

==> Automotive Cybersecurity standarts are defined by `ISO/SAE 21434`

Helped creating common terminology accross the industry.

Helped creating minimum base criteria for cybersecurity in the vehicle.

Creating security assurance level.

It is a reference that regulators point to. In order to enforce a standard, now regulators have a reference point.

The scope of ISO 21424 :

- Risk Management (Assess, monitor, evaluate potential risks.)
- Product Development (security of systems, hardware&software,
- performing TARA(Threat Analysis Risk Assessment))
- VARA (vulnerability analysis risk assessment)
- Operation, maintenance, and processess. Starting from beginning to the end, covers all aspects.
- process overview and interdepencencies.
- An `Assurance Level` is defined with ISO 21424 which is `Cybersecurity Assurance Level` this is like common security assurance level like how secure is the system or how much trust you can put onto it.

CanBus IDS/IPS vs Ethernet IPS/IDS

Comparing Can and Ethernet

CanBus is a Bus topology which means when a message leaves and ECU, it is guaranteed to reach any neighbir ecu on the same bus. Meaning there is no way to stop an attack using software which reside in the gateway since it is just listening to the traffic. Gateway can only stop propogation of the message from one bus to another but not on the same bus.

On the other hand, the ethernet is a star topology meaning on each port there is exactly one device is connected any device that is sent by the device can be inspected by the gateway.

Since ethernet is MUCH more speedy comparing to the Can, any IDS or IPS system needs lots of horsepower to be able to undertake full or partial analysis.

Message length in Can bus is only 8 bytes (60 for Can FD) compared to Ethernet which is 1500 bytes.

In terms of `source identification`, there is no source ID in can bus there is only Message ID. So you cannot tell who sent the message. This can be done using encapsulated protocols like `j1939` but standard does not have message origin identification. Can bus does not have any specific `message destionation` as well, it only has `message id` which is a `multicast` address that anyone can listen to it.

In ethernet, on the other hand, there is `distinct unicast` that means it has source and destination mac addresses, all clear. and etherenet support both unicast and multicast hence, with ethernet it is much easier to detect the source of a message.

Beware that `automotive ethernet` is not necessarily about ethernet, it is about all the networking. For example, on top of ethernet sits MAC, and then on top of that sits IP, on top of that sits TCP/IP or UDP/IP. To the application level, there is DoIP.

Can and Autmotive Ethernet IDS/IPS work process

IDS/IPS is designed to counter an attacker manipulating network traffic via a malicious application, hacked ECU or other controller.

It has 3 steps:

- It `learns` the normal.
- it `monitors` the network traffic and the `Routing` function of gateway rules
- it `Detects` anomalies based on deviation from `normal`. Prevents by discarding frames when possible.

Comparing CAn and Ethernet Rules Generation

- both of the them require recordings to generate rules out of it.
- Both of them use `files` where CAN uses DBC extension, ethernet uses XML.
- Both of them allow user to edit rules manually.
- Rule generation is simple in CAn Bus(but can get complicated if j1939 is used because dynamic adressing comes into play) and is complex in ethernet due to network nature (Mac adresses, DHCP, DoIO etc protocols)
- CAN is static, predictive and deterministic and uses whitelisting. But ethernet is more volatile, it uses both black and white lists simultaneously, also `signature based` detection. Signature based

detection is not implemented in Can Bus.

CAN and Ethernet IDS/IPS Architecture

IDS systems are essentially sdk integrated into ecu. This ECU can be an existing ECU like gateway or a dedicated ECU attached to CAN bus.

Incoming message/frame is `routed` to IDS/IPS to be processed.

the IDS/IPS inspects the message/frame and returns its findings with the associated information.

In case of anomaly, an event is reported.

when configured as IPS, when possible, the anomalous packet is discarded.

Can message is determined to be passing or not based on message ID, if valid, it is passed else, it is discarded. There is a possibility that a compromised ECU can send a message with a valid ID and if it is not a periodic message, it is not possible to determine it was an attack.

Ethernet Packet is filtered based on filtering rules initially. if it is not in white and blacklist, the default rule applies.

Canbus:

In Canbus, these IDPS will most likely be integrated into gateway, in other cases, it can be connected directly into ecus.

If we use IDS not IPS, it just taps into network so there is no need for changes in the vehicle.

Ethernet:

it `definitely` goes into gateway. it serves as automotive backbone.

Message Formality

Canbus:

Relatively easy to validate the format since the message is very short. only exception is CanBus TP(transport protocol) and SAE j1939 TP is used which adds more complexity.

Ethernet:

Etherned side of the message is much harded, needs 7 layer `Deep Packet Inspection(DPI)` to determine individual message format validity. For example:

- Ethernet frame
- IP header
- UDP header
- DoIP PDU header
- UDS PDU header

Protocol Message Flow

Canbus:

Relatively easy to validate flow, with requests and responses, and order of messages.

Impossible to determine ECU origin. Need to add digital signature using 3rd party compression softwares or SecOC.

Ethernet:

To capture a packet flow, you need to follow full OSI 7 layers individually and another problem is many fields are dynamically assigned per instance.

Diagnostics

Canbus:

- Using UDS protocol,
- Many attack options like hacker engaging diagnostics while normal driving.
- Detection and some prevention is possible but somewhat complex since it must be based on a combination of considerations such as vehicle context and usage of transport protocols.

Ethernet:

- based on DoIP and layered UDS protocol.
- Complex mechanism with multiple architectures and implementatitions(like DHCP)
- Many attack options like hacker engaging diagnostics while normal driving.
- sometimes detection is complex, especially if attacker is launched from compromised ECU from within the vehicle.

When running diagnostics in the test workshop using multiple vehicles connected to each other and the test machine, in order to prevent vehicle attacking to another vehicle, the configuration should be set to `multiplexler` not `full-mesh` so central gateway can talk to each vehicle but vehicles can talk to each other.

On media diagnosticsin CAN, the MOST system will not be incpected due to it is `airgapped`and low cyber risk. Also Automotive-Ethernet AVTP(Audio-Video Transfer Protocol) , which is not airgapped and very intense in traffic, will probably not be inspected but can be tested by partial sampling for protocol integrity.

Reporting

Canbus:

CanBus telematics messages. CAN id will send messages to be conveyed to SOC and analyzed there

Other options are possible such as independent SMS over cellular.

Ethernet:

a) `SOME/IP` **messages to telematics, API sending messages to microcontroller.**

it is a Control-communication protocol.

What is SOME/IP?

SOME/IP is a **middleware** where SOME stands for `Scalable service oriented middleware` which creates absraction in automotive and used for standardization of:

middleware used in data exchange that often passes through a network and it is the task of the middleware to ensure that the network itself is transparent to the software components exchanging the data.

it is a Control-communication protocol.

- header format,
- payload serialization rules
- service discovery mechanism
- remote procedure call mechanism(RPC)
- service-based communication
- small footprint
- Scalable,flexible, compatible.

data needs to be serialized. the faster the communication, the more resouce hungry the serialization. so there needs to be a fast and efficient middleware to serialize data. Due to string operations, text-based serialization and deserialization(JSON, XML these are self descriptive(structured) and text based) are very slow so a binay operation neededd. SOME/IP allows you to build the most efficient hi-speed performance system,formatting is optimized for low resources and high speed directly works on binary data. SOME/IP is like XML and JSOn, which are very slow. SOME/IP works on binary data and non-descriptive. it is fastest due to `zero-copy` and nearly as fast as Raw struct which is not serialization.

Advantages of SOME/IP Protocol SOME/IP protocol has many advantages compared to traditional automotive protocols like CAN, LIN, and MOST. Some of the worth-noting advantages of SOME/IP are the following:

1. Some/IP is license Free.
2. SOME/IP designed automotive use cases in mind, scales very well.
3. SOME/IP provides large bandwidth for data communication in the range of 100Mbps and takes care of not wasting the bandwidth at all by providing all data communications in a client-server configuration.
4. SOME/IP is supported by `AUTOSAR`, very fast in serialization, has `built in service discovery`
5. The data from the server ECU can be communicated to client ECU via unicast, multicast, and broadcast.
6. Being a middleware, it is suitable even for CPU-intensive applications, and OS-agnostic.

The size of the SOME/IP payload field depends on the transport protocol used. For UDP, the SOME/IP payload can contain 0–1400 bytes. The decision to limit the payload length to 1400 bytes was taken in

order to allow for future changes to the protocol stack, such as using IPv6 or adding security protocols. Since TCP supports the segmentation of payloads, larger payload sizes are automatically supported.

b) **Syslog format for direct interface to SIEM at the SOC.**

c) **other proprietary formats over IP are possible.**

Some TakeAways

- threats and risks are much more severe in automotive ethernet than in CAN bus.
- several areas are similar so some R&D effort on CAN can be used for ethernet.
- IDPS for ethernet is much more complex than for CAN.
- designing IDPS for autmotife ethernet requires expertise both in automoive field and in IT networking.
- For network security, SSL/TLS, VPN, IPSec protocols are for backend connectivity, MacSEc can protect all multicast, unicast, broadcast messages at line-speed. SecOC allows application layer protection for selected use cases.
- for access control, Ethernet access Control and SOME/IP is used

SecOC

SecOC = Secure OnBoard Communication.

SecOC protects CAN, ethernet, Flexray etc. communications. Adds security to the onboard communication.

normally Can bus , when reached, can be tapped into and tampered or even through `SOTA`(Software Over The Air) can be used to change the firmware.

This adds integrity and authenticity to the communication by being attached to the outgoing message.

in-vehicular communication vulnerabilities can be prevented by using SecOC in the comunication networdk. SecOC adds security to the `outgoing` message to achieve `integrity and authenticity` ofthe message.

It is specified to check the authenticity of a single transmitted PDU (Protocol Data Unit) in order to detect attacks such as replay attacks, message tampering, and denial of service attacks.

With SecOC implementation, **the attacker HAS TO know the sender's secret key in order to spoof a message.**

on the **sender ECU**, the SecOC module is sitting between PDURouter, CryptoService Manager and the counter.

1- PDU Router receives a PDU from an upper layer.

2- passes message to SecOC module to add authentication.

3- SecOC obtains `Freshness Value` from the counter.

4- SECOC then generates `authenticator` using services from CryptoService Manager.

This authenticator is `freshness value + secret key` using secret key

5- SecOC then attaches the freshness value and authenticator to the `PDU Frame` and returns it to the PDU router.

6- PDU router then passes it to the destination interface.

CSM module provides cryptographic services for SecOC.!

authenticator is also known as `MAC, Message Authentication Code` if keys are symmetrical, if asymmetric, then it is called as `signature`.

on the **receiver ECU**,

1- PDU router receives a secured PDU with authentication added from network interface.

2- this frame is sent to SecOC module for authentication

3- SecOC module strips authenticator and freshness value from the secure frame and freshness value is compared with the current counter value in the counter module.

4- if the received frame is not a fresh one then it is considered a `replay attack` and discarded.

5- through CSM,new authenticator is generated using the secret key and freshness value for comparison with the received authenticator value.

since the frehness value and secret key are the same, the keys are expected to be the same.

6- if auth is successful, then PDU is sent back to PDUR for further processing.

==

CAN bus has the limitation. Classical CAN frames provide a payloadsize of only 8 bytes. the PDU along with freshness value and authenticator cannt be supported in one frame. This leaves 2 options:

1. Truncate these fields to shorten the lenghts.
2. Send authenticator in another frame.(available in Autosar)

Between two, the tradeoff are the `security level` and `impact on busload`.

Networks like CAN FD, Flexray and Ethernet do not have payload limits.

==========

Vehicle Key Management Systems

Automotive CyberSecurity Threat Scenarios

- Do not forget that we have remote access to the vehicles.
- Also you have customer data collected and maybe stored.
- Now we have internal networks which means easy internal attack traversal (lateral movement.)
- Access to ECU-application meanis theft of intellectual properties.
- We have more software, meaning better attack surface.

Some Ways to Mitigate these.

- **Debug Access Protection**. This may help to prevent gain access to car network. (especially JTAG and UART)
- **Reverse Engineering Protection**. (through encryption and/or obfuscation of Data-at-rest)
- Add **Secure Boot** for preventing malicious software from being flashed over ethernet. (checking signatures)
- use **SecOC**.
- Common approach should be **Distributed Trust**.
- **Code Signing**

Root of Trusts are : Symmetrical Keys, Asymmetrical Keys, Password Protection, certification.

Distributed trust boils down to Vehicle Key Management Systems. This means secure generation, storage, distribution and destruction of keys.

The Goals of Vehicle Key Management Systems are:

- ***Create*** cryptographic material:
  - which can be symmetric, asymmetric, password protected, or certificate protected.
  - Cyryptographic material can be platform, product, or even ECU specific(most complex but most secure).
  - *Good Enough* Randomness is hard to achieve. (random bit generation)
- ***Distribute*** cryptographic material:
  - these keys should be accessible to control units.
  - Keys must be stored in the control units in manifacturing.
  - ECUs are mostly manufactured on-premies at the supplier-sites but they belong to OEMs so secure onboard comms is something OEM has to make sure it works.
  - in the final assembly of the card, all keys distributed MUST fit together.
- ***Store*** cryptographic material:
  - access to locked control units must be possible after shipping the car.
  - cryptographic material must be stored **securely on-chip**
  - Software supported hardware security.

- ***Additional Requirements***:

- - **Security**: the material must be protected.
- - **Availability**: Manufacturer must continue development and support 24/7.
- - **Scalability**: huge amount of keys/passwords/certs must be managed.

==========

Architecture is divided into 3 layers:

1. **Backend + Middleware**:

main component is the database which is used for creating and storing the cryptographic material.

It is usually set for `High Availability` in mind. Also, for the better security, **Hardware Security Modules** (HSM) are used alongside database.

Database are used because HSM are small in memory to keep all these data.

HSM is in charge of data-at-rest encryption and randomness generation.

Key Disstribution and Management is taken care by attached services/applications.

different interfaces are used like REST API, KMS-interfaces, OPENPGP for eMAil, proprietary interfaces etc.

interfaces may implement server-side services like encryption and signing.

2. **FrontEnd**:

frontend directly talks to the backend to undertake tasks like:

- - getting keys and flash them to ECUs in manufacturing.
- - Caching OEM-keys for asynchronous supplier-side manufacturing.
- - retrieve unit-specific passwords for root-cause analysis.

you want to store them in the backend but use it on the frontend for the key security.

depending on the use case, the frontend side application can be manual GUI, another machine ,or some library which is used by another software.

Sometimes even offline use-cases have to be supported by the VKMS like in-line flashing verification checks.

3. **In-Vehicle(network and on-chip)**

Processor/CPU + Rom/Flash + HSM.

- - Every crypto material managed by the VKMS will eventually be distributed into an ECU for in-vehicle purposes like SecOC, SecureBoot, IP-Protection.
- - Certain modules will always be present on chip for security and practicality reasons. these are Secure way of storing (Secure Hardware Extension(SHE), Software based HSM), A library/stack of hardware providing cryptographic services.(AUTOSAR Crypto Stack, Cryptoprocessor).

HSM on the chip is required for the protection of the keys stored on the chip so they are not extracted or extracted easily. SHE is specifically designed for this purpose.

Like if not stored on chip, how will it execute secure boot check, or signature checks.

- - Depending on the VKMS implementation, on-chip software might be in or out of scope.

but in any case you need to address as holistic problem.

===

Exceprts from YouTube Videos

1.

End to End (E2E) does not mean `Application to Application` only. They all sit on a system. Thinking of the security measurements, many of them go below the application layer which are `Stack` layer and `Hardware` layer. Of these, there are MacSEc, IPSec,SecOC none of them are Application to Application.

Although TLS is an Application layer security, in AUTOSAR implementation, it is a `stack layer` security. Although in Linux, TLS can be linked to Application, when acceleration is needed for cryptography, which you need in Automotive industry, TLS is a shared resource between the application and OS levels.

2.

In automotive security, unlike IT , IP spoofing, port Spoofing etc can be prevented if the network security design works. Like hop-by-hop MACsec combined with strong filtering on Ethernet Switches causes enforcing VLANS and IPS, allows effective firewall on hosts and in the network and this allows implementing strong Access Control at many places because all these information are now trusted.

3.

Do every protocol need security build in?

for example DoIP(lives in OSI 5-7) reaches out of the vehicle so it makes sense to secure it. You can enforce security within or support it with TLS below it, or the combination of both.

But say, do we need to secure UDP Network Management(UDPnm) for network management by building security into the protocol? Technica does not think so. 😄

Main idea is, if two sides of the bridge are secure, you dont need a security for the bridge. Because if you implement MacSec, which protects everything sitting on top of it, an external attacker cannot modify antyhing.

for internal attacker, who is controlling a compromised ECU, say Head Unit, you can use ACL, or sort of firewalls. But you dont put them on TCP layer, but below it to protect it. So , you DONT need built in secuirty for everyhing as long as you think secuirty first and design things accordingly.**Make sure you have access control and filtering.**

4.

Are IDPS the most important security mechanism?

There is a hype in IDS and IPS systems. but instead of going through IDS, you can stgart with createing the parth to transport `events` from vehicles to your backend and then incrementally add `events` and look at your data.

IPS on the other hand , has **actively** intervening mechanisms which can cause vehicle(especially autonomus cars) to go out of control, not-desirable.

IDS is JUST a tool in your secuirty toolbox, and you need to craete a strategy for it.

5.

Security does not allow testing.

This myth comes from the encryption of the data. However that is not correct. Think of this way:

- SecOC is authentication only and there is no encryption.
- IPSec, TLS and MACsec support authentication only modes.

How to handle `auth` does not let me change anything problem?

- add secure process to turn secuity on ECU on and off.
- add a secure process to share keys between test system and ECU.

As long as you know your tool chains and you design your process, testing is possible.

==========***==========

Standard ethernet cannot be used in cars , the EMC emission is not automotive compliant. automotive ethernet(100 and 1000 BASE T1) made some changes on phy layer to meet the requirements:

- full duplex comms
- avaliability and low cost
- reliable and fast link establishment (100 miliseconds is the upper bound)
- very low bit-rate error
- appropriate EMI and EMC missions.

Testing Considerations:

1- Define test levels according to the **V-Model**.

V stands for verification and validation

On the left side of the V-Model there is system model from more abstract to more concrete.

On the right side of the V-Model there is the integration and test levels.

- User Requirements --> Vehicle Test

entire product test, on test/prototype vehicles.focus is bring car into prod maturity.

- System Requirements --> System Test

testing behavior of `whole system` black box, HIL and SIL (software in the loop)

- System Architectural Design --> System Integration Test

test on target, performed onDUT/HIL, mainly black box testing. focus is integration and stability.

- Software REquirements --> Component TEst

Component test which is Grey Box, functionalk requirements testing

- SoftwareDesign + Software integration

CI. automated tests, high-frequency commits and PRs

2. Design approprioate Test concept and specifications.

3. Implementation Levels: Component,EoL,Partial Network,Full Network and vehicle test solutions.

You need to consider the multitude of protocols. Physical and layer testing. individual components test(HIL). ethernet test suite, ethernet network integration test.

4. Ensure reproducibility of test results.

==========

Verification and Validation

ISO 26262-6 is the main functional safety standard with respect to software development for high-integrity in-vehicle applications. It defines requirements and constraints for software development, verification and validation processes.

ISO 26262 guide automotive engineers to identify hazards, risks, and resulting safety goals. The check on how well the design meets those goals comes through verification and validation.

In **verification**, systems and supporting software undergo quality audits, testing, and expert reviews to confirm that all are designed to specification. During **validation**, product tests – including full operation of systems and components – ensure the end product will function as intended and confirm the adequacy of the safety goals.

ISO 26262 (—Road Vehicles - Functional Safety‖) is a standard for the design and verification of automotive systems and a buzzword!

Verification process includes checking of documents, design, code and program whereas Validation process includes testing and validation of the actual product.

Verification does not involve code execution while Validation involves code execution.

Verification uses methods like reviews, walkthroughs, inspections and desk-checking whereas Validation uses methods like black box testing, white box testing and non-functional testing.

Verification checks whether the software confirms a specification whereas Validation checks whether the software meets the requirements and expectations.

Verification finds the bugs early in the development cycle whereas Validation finds the bugs that verification can not catch.

Comparing validation and verification in software testing, Verification process targets on software architecture, design, database, etc. while Validation process targets the actual software product.

Verification is done by the QA team while Validation is done by the involvement of testing team with QA team.

Comparing Verification vs Validation testing, Verification process comes before validation whereas Validation process comes after verification.

General E/E Knowledge

DONT FORGET THAT CAN/FLEXRAY/LIN/ETHERNET ETC ARE NOT ECU! THEY CONNECT TO ECU and ECU were BEFORE these network systems!!!!

ONCE ECU vardi (ABS, fuel-injection gibi) sonra CAN etc geldi.

E/E stands for **'Electrical/Electronic'**

There were no electrrical/electronic parts in the vehicles in the 70s, maybe some small parts to toggle the lights.

Then came remote door unlock. and gradually put on.

These small electrical systems made use of `Electronic Control Units` (ECU) to apply rules or perform calculations which were wired to sensors, motors, actuators etc.

each of these components means more **cost** and **weight** to the vehicle.

what if we wire ECUs to each other? that reduces the wire and costs.

This means, creation of `network` in the vehicle so ECU can send / receive data.!

also, per regulations, ECUs are asked to undertake `Diagnostics` , aka OBD.

OEMs wanted to reduce the cost by finding ways to use single ECU in different (but similar) vehicles. This led them developing ECUs than can be tuned (calibrated) to match the vehicles.

With the advent of technology, we increased the thinking power of ECUS from single core 8 bit to multi core 32 bit, even adding AI.

With being connected to GPS or Web servers, the vehicles became a part of greater network. This also helped offboard calculations to be made instead of putting heavy loads on the ECU. (yani disardan aliyor veriyi mesela bazen, kendi hesaplamiyor.)

A new generation of ECU, called HCP (High-Performance Computing Platform) is coming up. Embedded systems perform single,specific job and have specialized Embedded Software.

In vehicles, these embedded systems are ECU. These provide the control functions.(logic functions, if this happens, do this.) also they provide diagnostics, calibration etc.

Unlinke microprocessors, these microcontrollers that ECUs use come with a dedicated ram and storage.

Networks allow ECUs to share information with a reduced risk of undetected errors.

the networks used between ECU's within vehicle E/E system is called `serial communications` where serial means single stream of communication.

Sequence of voltages are applied to the network, that often directly represent binary digits (bits.)

So each bit correspods to a voltage.

Video and other `heavy` data also used in `Advanced Driver Assistance Systems` (ADAS)

Can , Lin, Flexray are `Signal oriented architectures` whereas Ethernet also allows vehicle E/E systems to use `Service-Orientated architecture` (SOA).

In some defect situations, ECU can accept diagnostic requests and send diagnostic responses. On borad Diagnostics(OBD) standardises requests, responses related to emissions control systems, their structures and contents.

If there is a problem in the ECU, like fault memory, this is logged and also sent in form of DTC for diagnosis.

Main ECU software are :

```
BSW (Basic Software)

RTE (Runtime Environment)

Application Software

Flash Bootloader
```

Everytime a ECU boots, flash bootloader runs and checks the main ECU software is valid or not.

it is also used when software update is in progress. the ECU software is stored in `non-volatile memory` (NVM) meaning on reboot, memory is not erased.

the FBL provides diagnostic services that allow the whole NVM to be erased and re-flashed by a network connection. also can verify the data.

what about **HCP**?

as stated above, HCP is a new generation of ECU . High-Performance Computing Platform.

nowadays, vehicles use Ethernet networks to move high volume data and they evaulate this data to build a picture of their environment to make predictions as to what happens next and how to behave or how to determine them precisely.

this requires more horse-power than ECU can provide. HCPs are primarily used for `Data-centric processing` (DCP)

What is `Data-centric processing`?

some common vehicle environments require very fast data processing mesela otonom suruculu araclari dusun, yerleri surekli degisen objeleri takip edip ona gore haber vermesi lazim.

for this end, we need `high-bandwith` network and large amount of memory. also we need `GPU` for processing the data because they split tasks into thousands of small pieces and execute them in parallel thanks to their hundreds of cores.

while ECUS use single stream of signal data, HCP needs dynamic, `service oriented` communication because single stream is not enough.

to meet the modern needs (like HCP), AUTOSAR developed a new set of specifications for the `Autosar Adaptive Platform` that runs on a POSIX operating system. (like linux.)

modern vehicles need both hardware-centric (ECU) platforms and software-centric (HCP) platforms together.

What do future vehicles need alongside `predictability`?

Autosar adaptive platform (for HCP) provides access to the computing power needed by future vehicles which are ACES functions.

ACES (Advanced Driver Assistance System) is a new generation of `Autosar Adaptive Platform` (ADAP) that is designed to meet the needs of future vehicles.

A ==> Automation

C ==> Connectivity

E ==> Electrification (less carbon emission)

S ==> Sharing.

HCP can provide all these ACES functions!

While ECU programs depend on signals, they cannot perceive Objects so languages like C are used. But HCP can and has to rely on objects so OOP is used, like C++.

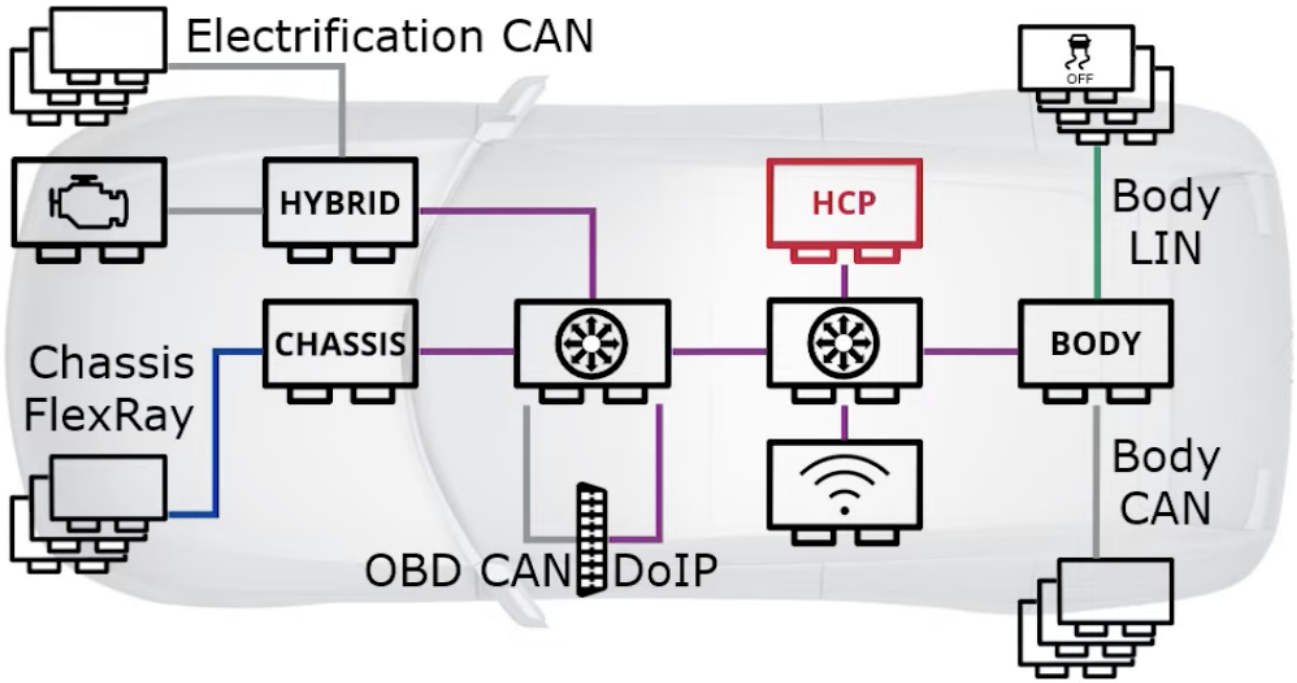
While ECUs rely on `hard real-time` behaviors (ince ayarlanmis deadlines), HCP rely on POSIX operating systems and ACES functions.

with the 2000's there has been an ECU explosion. there were lots of ECUS in the vehicles but this had some bottleneck. because there is a limit for how many ECUs can be used in a CAN Bus.

to overcome this, **`GATEWAYS`** were introduced. Gateway is a `router`,central hub in the vehicle that securely interconnects and processes data across heterogeneous vehicle networks to and from functional domains such as powertrain, chassis and safety, body control, infotainment, telematics and advanced driving assistance systems.

later on, with the advent of Automotive Ethernet, `Functional Domain Controller` (FD-C) was introduced. they were connected via AE to ethernet SWITCHES and aggreate functions that were originally hosted by ECUs.

HCP runs Infotainments, and can also run `Predictive Traction Control`.



to problem was, there is lots lots of cabling and wiring. to adress this, we can divide a vehicle into topological zones, each with a `Zone Domain Controller` (ZDC) with busses to `IO Nodes`. that gather inputs and provide outputs relying on little amount of ECUS. still ethernet and switches are used here in interconnectedness.

this HCP, ZDC, and IO nodes enable `Software-defined vehicles`.

So what is **AUTOSAR**?

for ECU ==> AUTOSAR Classic Platform

for HCP ==> AUTOSAR Adaptive Platform

in classical times, there were little or no ECU specification standardisation, so vehicle manufacturers had to specify everything else themselves while the suppliers had to cope with many similar but different specifications.

Because of lack of standardisation, there was no eaasy way to add new features to ECUS or move functions. This caused adding more and more ECUS for even small stuff to be handled.

Hence both manufacturers and suppliers agreed on the necessity of a standardisation. The result was

Automotive Open System Architecture (AUTOSAR) which standardized bthe specifications for ECU software y dividing the architecture into the stacks and modules with well-defined responsiblities.

it not only standarddized but also set `methodology` and `file formats`(Autosar XML) to be used by software tools.

AUTOSAR `foundation` specification helps Autosar classic and adaptive platforms to be used together.

at the heart of AUTOSAR architecture is the `Run-Time Environment` (RTE) which is a software stack that provides the basic services for the ECU.

AUTOSAR RTE is the Run-Time Environment (RTE) that is the heart of the AUTOSAR ECU architecture. It provides the infrastructure services that enable communication between AUTOSAR software components. It is acting as the means by which AUTOSAR software-components access basic software modules including the OS and communication service.

Autosar adaptive platform (for HCP) provides access to the computing power needed by future vehicles which are ACES functions.

DTC and Flashing and V-Model

DTC

DTC stands for **Diagnostic Trouble Code**.

DTC is a code used to diagnose malfunctions in a vehicle or heavy equipment. While the malfunction indicator lamp (MIL)—also known as the check engine light—simply alerts drivers that there is an issue, a DTC identifies what and where the issue is. DTCs are also called engine vehicle fault codes codes, and can be read with a scanner that plugs directly into the port of a vehicle.

DTC is used by a vehicle's onboard diagnostics (OBD) system to alert when a malfunctuion is detected.

When the vehicle’s OBD system detects a problem, it generates a specific DTC code and transmits the alert to the vehicle’s instrument panel as a warning light. In vehicles equipped with a telematics system, the alert can be delivered directly to the fleet. The system can be set up to deliver the alert right to the maintenance department.

These codes were created by the Society of Automotive Engineers (SAE) to help vehicles comply with emission regulations. SAE is now called SAE International, and it is the professional organization that develops standards for automotive engineers.

How do DTC codes work?

Diagnostic Trouble Codes or OBD-II (in light-duty vehicles) or J1939 (in heavy-duty vehicles) trouble codes are codes that the vehicle’s OBD system uses to notify you about a problem. Each code corresponds to a fault detected in the vehicle. When the vehicle’s computer detects an issue that requires attention, it will activate the corresponding trouble code. What does a DTC mean in a vehicle?

The engine control module (ECM) functions as the main computer on all newer model vehicles. The ECM is also commonly referred to as the engine control unit (ECU) or powertrain control module (PCM).

When your vehicle's ECM is directly connected to your company via telematics, app, or gateway, you can find out in real-time from your desk in the home office what's going on with the vehicle. KeepTruckin's vehicle diagnostics automatically monitors fault codes through its direct connection to on-board vehicle diagnostics.

DTC operates on **UDS** (Unified Diagnostic Services) protocol which is an ISO standard that defines structures of diagnostic requests and responses.

Unified Diagnostic Services (UDS) is an automotive protocol that lets the diagnostic systems communicate with the ECUs to diagnose faults and reprogram the ECUs accordingly (if required). It is called unified because it combines and consolidates all the standards like KWP 2000, ISO 15765 and others.

Flashing ECU

updating the ECU is done through UDS protocol.

Flashing = Tuning.

Flashing means embedding the software into the chip just like flashing a ROM into your phone.

One of the major reasons to consider while designing the Flash Bootloader is the need for compliance with **ISO 26262 Standard for functional safety**.

You can undertake flashing:

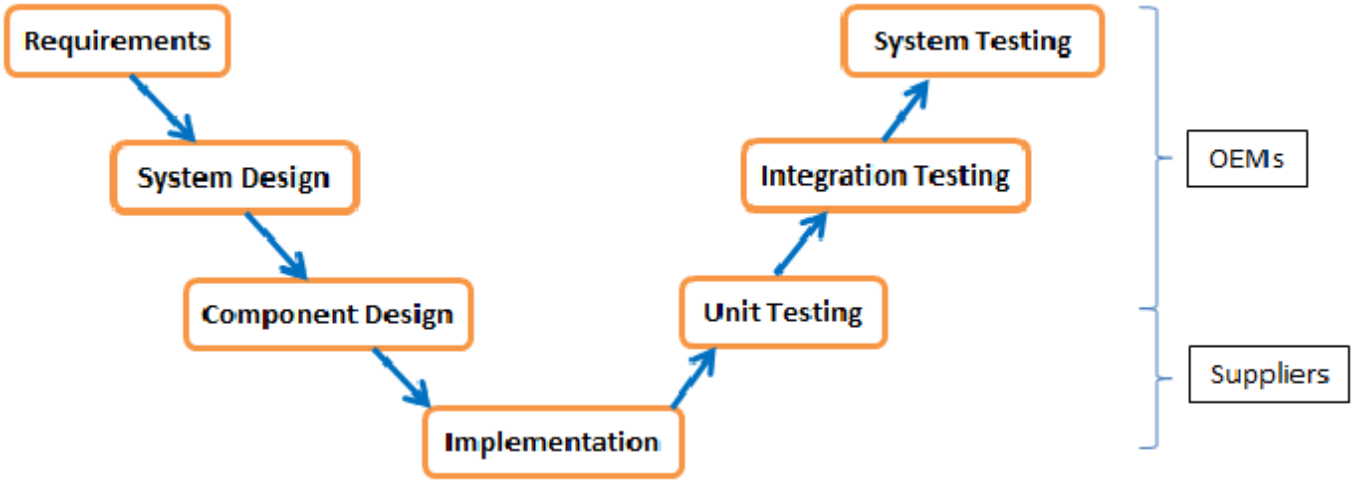
1. via external hardware.
2. application protocols.(UDS) ==> to the flash bootloader(FBL)
3. FOTA (Flash Over The Air). ==> communicate over ETHERNET.

ECU flash can be segmented into 2 section:

1. Program Flash (PFlash) : flashing into where the entire code resides,and is responsible for control logics like entire car program or changing visual commander to voice commander etc.
2. Data Flash (DFlash) : flashing into where the data resides, and targets the variables (constants, maps, curves) and referred when the software needs like re-flashing for setting maximum speed or uploading new map.

V-Model

To put it simply, V-Model (where V stands for verification and validation) splits the development process into two parts – the left arm of the V consists of requirement analysis, function design, and software development while the right arm concentrates on the verification and validation activities followed by the release. The V-model is an extension of the waterfall methodology. V-Model emphasizes testing, particularly the need for early test planning.



Quick Swithc/Hub/Router Notes

Very Good Explanation!

Quick notes on 3 of them:

Hubs and switches are used to **CREATE** networks while routers are used to **CONNECT** networks.

- **Switch:**

has multiple ports. but unlike hub, a switch is intelligent. swtich can learn phtical addresses (MAC) and stores them in switch table. Hence, when a packet comes, it only goes to the intended device. so switches are far more efficient and secure than hubs.

like hubs, switches are also used for local area network, meaning internal networks.

They dont exchange data out of their own nwtworks.

Swtiches are generally **IP blind**, hence they cannot handle external network traffic like internet

- **Hub:**

Connects devices on an **internal** network. has multiple ethenet ports. not intelligent because does not filter data.

only thing a hub knows is whether a device is connected or not.

when a data packet arrives to one of the ports, data is **copied** to all the other ports so all devices see that data packet. this is why it creates security and bandwith issues.

it acts as a broadcaster.

hubs too are generally **IP blind**, hence they cannot handle external network traffic like internet

- **Router:**

forwards data based on **IP address** hence can externally communicate.

When a packet comes in, it inspects the IP address and understands whether it is for its own network or for external network. If intended for local network, it receives it ; if for external network, it forwards it to another network.

the router is a gateway of a network.

So you can build 2 interconnected networks, being interconnected by a shared router and for each network, a swich handles their respective internal traffic handling.

More On Ethernet

Quality of Service(QoS) Protocols:

One of the reasons for the automotive industry to adopt Ethernet-based communication as an in-vehicle networking system is the chance for synergies, i.e., the possibility of reusing protocols that have been developed and tested in other industries.

NOTE: AVB (Audio Video Bridging) is officially renamed to TSN(Time Sensitive Networking.)

The implementation of AVB requires that the underlying Ethernet network runs at least at 100 Mbps full-duplex, that the Ethernet payload does not exceed the maximum size of 1500 bytes

AVB offers three key capabilities that depend on each other:

- Bandwidth-reservation that depends on
- Traffic-prioritization and
- Time-synchronization

TSN is another collection of standards designed to **extend** AVB. It also leverages traffic prioritization, time-synchronization, and can be used with both the credit shaper and strict priority schedulers.

Aside from improving performance and provided performance guarantees, TSN adds two new capabilities:

- Time-scheduled traffic
- Frame-preemption

Switches and Virtual LANs (VLANs)

In an Ethernet network there are two important properties that can have an impact on the network's robustness: (a) Communication can be flooded and (b) there is no default control in an Ethernet network on how much traffic a network participant can transmit.

A powerful way to structure an Ethernet network is virtualization. On top of the physical Ethernet network, different virtual networks may be created, each with possibly different subset topologies and QoS configurations. For Ethernet this feature is called **Virtual LANs** (VLANs) as defined in IEEE 802.1Q

VLANs are often used to not only limit broadcast domains but also to isolate traffic. Depending on the design, the isolation can be between critical/uncritical traffic, internal/external traffic, or it can isolate the traffic flows of different application areas or security zones.

Two additional aspects to consider with respect to VLANs are the following:

- Data logging and testing:

VLANs provide flexibility in relating ECUs to network segments, independent of the physical location of the units. This will have increasing importance for data logging and analysis in growing Automotive Ethernet networks.

- Performance:

Certain communication may be assigned to a specific VLAN and this VLAN can, in turn, be prioritized within the switches.

===

Switches:

The main task of a switch is to look at the address fields of a received packet and to forward it to the transmit port(s) via which the destination endpoint can be found.

To support this fundamental behavior the switch maintains a **forwarding table**. During normal operation, a layer two switch **“learns” MAC addresses** (MAC LEARNING) by observing the MAC source addresses of received packets and associating those MAC source addresses with the ports via which the packets were received. Thus, when the same MAC address is later observed as a packet’s destination address, the switch can readily identify the port to which the packet must be forwarded. If a packet is received whose MAC destination address has not yet been learned, the switch floods the packet to every transmit port except the port via which the packet was received. This ensures that the packet makes it to its intended destination regardless of whether or not its destination address is known to the switch.

Sending broadcast or unknown multicast packets has a similar flooding effect. To prevent the flooding of packets with unknown unicast addresses, static or semi-static configuration of unicast addresses is often recommended. In this solution, the Ethernet Switch is configured with all unicast addresses in the network or they may be learned in very limited circumstances, e.g., just once with the first start in the factory. In order to limit the flooding of unknown addresses, the Ethernet switches would then need to be configured to discard all packets with unknown source or destination addresses. Unfortunately, the usefulness of this in a car is limited, despite the network being preconfigured and more or less static in its configuration. It would make testing and replacing units during service more complex, since, e.g., the tester addresses cannot be known a-priori and partial networking becomes difficult.

For multicast traffic, configuration of multicast addresses is very helpful. Instead of flooding the multicast packets within the packet’s VLAN, each Ethernet Switch is configured to only forward these packets to the specified outgoing switch ports. This is especially helpful if VLANs are used to define domains and not traffic types. In addition, configuring Ethernet switches to discard packets with unknown multicast destination addresses is highly recommended. This feature is already common for multicast addresses configured by SRP. Another important option is to configure ingress policing for Multicast and Broadcast traffic. Additionally, in some Ethernet switches a specific variant for this is present as Denial of Service (DoS) prevention.

Routing vs Switching:

Routing versus Switching Ethernet-based communication generally provides for two ways to pass packets through a network: **Using the MAC addresses in the Ethernet packet header at ISO/ OSI layer two, or**

using the IP addresses provided in the header of the IP packet (that can be found in the payload of the Ethernet packet) at layer three. The first method is called switching (throughout this book), the second routing.

When switching is possible, it seems to be the simpler choice. With switching, packets are forwarded using the addresses of a packet's outermost header (Ethernet), whereas routing must examine the IP header that follows the Ethernet header. Switching can thus offer somewhat lower latencies and is performed in hardware within dedicated switch semiconductors that are currently readily available to the automotive industry. Routing is used to interconnect separate layer two networks (i.e., LANs). This is done to minimize the scope of a LAN's "broadcast domain," to allow a mix of different layer two technologies (i.e., something other than Ethernet), and to build networks of massive scale (e.g., the global Internet).

Routing is also well established and commonly used in the IT world and in data centers. These networks benefit from the efficiency and stability of layer three, for which well-proven methods and protocols exist.23,24 As the in-vehicle network is always limited in size, as routers seem more complex and expensive (software), and as dedicated automotive router chips are not (yet) available, the car industry has, until now, shied away from using routing instead of switching. However, the following list gives examples of three well-established features and methods enabled by IP that might be worthwhile to investigate for automotive deployment: 282 Protocols for Automotive Ethernet

The **Time-To-Live (TTL)** field of IP ensures that any single packet cannot make it further than all of the way across the network. If a packet's TTL expires, the packet is discarded; guarding the network against infinite loops. This feature safeguards a network that may have many redundant routes. While the automotive network sees little redundancy today, it might play a larger role when the concept of networks is truly realized. A feature such as TTL will be of use in the automotive industry then, too.

The **Explicit Congestion Notification (ECN)** is useful when best-effort traffic is mixed with time-critical and/or synchronous data. It functions as follows: When a router determines that a packet was present in its queues longer than some specific time limit, the router sets the ECN field to reflect this. This allows the recipient to notify the source about the issue, which, in turn, can reduce its transmission rate and, consequently, reduce congestion. Reducing congestion has the beneficial effect of reducing network latency (no queued packets means no delays). And, if congestion is allowed to grow to excess, packets may be dropped. These dropped packets must be dealt with by a higher-layer protocol such as TCP (or even the application itself). This, unfortunately, leads to greatly reduced network throughput. Remember that TSN reserves QoS for prioritized traffic (see Section 7.1). As cars adopt more distributed computing over time, providing some minimum guarantees and greater efficiencies for best-effort traffic may also become relevant.

Remote Direct Memory Access (RDMA) over Converged Ethernet (RoCE) v2 is an important tool for distributed computing that runs on top of Ethernet, IP, and UDP [81]. It implements a large part of the protocol stack in hardware and thereby saves power and time as well as frees up CPU load. It may be beneficial, when software is virtualized in cars and arbitrarily distributed among the ECUs.

OOP and Programming stuff

OOP

Object-oriented programming has four basic concepts: encapsulation, abstraction, inheritance, and polymorphism. Even if these concepts seem incredibly complex, understanding the general framework of how they work will help you understand the basics of an OOP computer program. Below, we outline these four basic principles and what they entail:

Encapsulation Abstraction Inheritance Polymorphism

1. Encapsulation

The word, “encapsulate,” means to enclose something. Just like a pill "encapsulates" or contains the medication inside of its coating, the principle of encapsulation works in a similar way in OOP: by forming a protective barrier around the information contained within a class from the rest of the code.

In OOP, we encapsulate by binding the data and functions which operate on that data into a single unit, the class. By doing so, we can hide private details of a class from the outside world and only expose functionality that is important for interfacing with it. When a class does not allow calling code access to its private data directly, we say that it is well encapsulated.

Example: Elaborating on the person class example from earlier, we might have private data in the class, such as "socialSecurityNumber," that should not be exposed to other objects in the program. By encapsulating this data member as a private variable in the class, outside code would not have direct access to it, and it would remain safe within that person’s object.

If a method is written in the person class to perform, say, a bank transaction called "bankTransaction()," that function could then access the "socialSecurityNumber" variable as necessary. The person’s private data would be well encapsulated in such a class.

2. Abstraction

Often, it’s easier to reason and design a program when you can separate the interface of a class from its implementation, and focus on the interface. This is akin to treating a system as a “black box,” where it’s not important to understand the gory inner workings in order to reap the benefits of using it.

This process is called “abstraction” in OOP, because we are abstracting away the gory implementation details of a class and only presenting a clean and easy-to-use interface via the class’ member functions. Carefully used, abstraction helps isolate the impact of changes made to the code, so that if something goes wrong, the change will only affect the implementation details of a class and not the outside code.

Example: Think of a stereo system as an object with a complex logic board on the inside. It has buttons on the outside to allow for interaction with the object. When you press any of the buttons, you're not thinking about what happens on the inside because you can't see it. Even though you can't see the logic board completing these functions as a result of pressing a button, it's still performing them., albeit hidden to you.

This is the concept of abstraction, which is incredibly useful in all areas of engineering and also applied to great effect in object-oriented programming.

Example: In OOP, we might have a class defined to represent the human body. One might define some functions as part of its publicly facing interface such as “walk()” or “eatFood().” Calling code could call these functions and remain completely oblivious to the complex inner workings of the human body and its necessary functions to perform the act of walking or eating. These details are completely hidden in the implementation of the walk() and eatFood() body functions and are, therefore, us abstracted away from the

end user. In these cases, it’s not important for calling code to understand how the brain coordinates walking or how the stomach manages digesting the food, but rather simply that a human walked or ate.

3. Inheritance

Object-oriented languages that support classes almost always support the notion of “inheritance.” Classes can be organized into hierarchies, where a class might have one or more parent or child classes. If a class has a parent class, we say it is derived or inherited from the parent class and it represents an “IS-A” type relationship. That is to say, the child class “IS-A” type of the parent class.

Therefore, if a class inherits from another class, it automatically obtains a lot of the same functionality and properties from that class and can be extended to contain separate code and data. A nice feature of inheritance is that it often leads to good code reuse since a parent class’ functions don’t need to be re-defined in any of its child classes.

Consider two classes: one being the superclass—or parent—and the other being the subclass—or child. The child class will inherit the properties of the parent class, possibly modifying or extending its behavior. Programmers applying the technique of inheritance arrange these classes into what is called an “IS-A” type of relationship.

Example: For instance, in the animal world, an insect could be represented by an Insect superclass. All insects share similar properties, such as having six legs and an exoskeleton. Subclasses might be defined for grasshoppers and ants. Because they inherit or are derived from the Insect class, they automatically share all insect properties.

2. Polymorphism

In OOP, polymorphism allows for the uniform treatment of classes in a hierarchy. Therefore, calling code only needs to be written to handle objects from the root of the hierarchy, and any object instantiated by any child class in the hierarchy will be handled in the same way.

Because derived objects share the same interface as their parents, the calling code can call any function in that class’ interface. At run-time, the appropriate function will be called depending on the type of object passed leading to possibly different behaviors.

Example: Suppose we have a class called, “Animal” and two child classes, “Cat,” and “Dog.” If the Animal class has a method to make a noise, called, “makeNoise,” then, we can override the "makeNoise" function that is inherited by the sub-classes, "Cat" and "Dog," to be “meow” and “bark,” respectively. Another function can, then, be written that accepts any Animal object as a parameter and invokes its "makeNoise" member function. The noise will be different: either a “meow” or a “bark” depending on the type of animal object that was actually passed to the function.

=======

Software Design Patterns

Design patterns are typical solutions to common problems in software design. Each pattern is like a blueprint that you can customize to solve a particular design problem in your code.

Design patterns differ by their complexity, level of detail and scale of applicability to the entire system being designed. I like the analogy to road construction: you can make an intersection safer by either installing

some traffic lights or building an entire multi-level interchange with underground passages for pedestrians.

The most basic and low-level patterns are often called idioms. They usually apply only to a single programming language.

The most universal and high-level patterns are architectural patterns. Developers can implement these patterns in virtually any language. Unlike other patterns, they can be used to design the architecture of an entire application.

In addition, all patterns can be categorized by their intent, or purpose. This book covers three main groups of patterns:

- **Creational patterns** provide object creation mechanisms that increase flexibility and reuse of existing code. *How objects are created*
    - **Singleton** ensures that a class *has only one instance* and provides a global point of access to it. Mesela settings class gibi. bir kere yaziyorsun.
    - **Factory** provides an interface for creating objects without specifying their concrete classes. bir suru if-else check yerine, bir factory class method yazip bizim yerimize o karar veriyor if elselere gerek kalmadan.
    - **Builder** allows you to construct complex objects step by step. for building an object, instead of using a constructor with full of parameters, you create them step by step and assigning each property to this functions.
    - **Prototype** (clone) allows you to create objects without using the new operator. It is an alternative to `inheritance`, you dont inherit from a class but from a prototype or object. using **proto** is a way for it in Javascript for example. instead of using new keyword you use *Object.Create(prototype, properties)*

- **Structural patterns** explain how to assemble objects and classes into larger structures, while keeping these structures flexible and efficient. *How objects relate to each other*
    - **Facade** provides a unified interface to a set of interfaces in a subsystem. it is a simplified API for reaching other properties in the codebase. Bunun sebebi, complex code logici bir API arkasina saklamak (API burda bir class olabilir) ihtiyac olunca bu API call ile islem yapilir. Butun npm packegeler bir facade aslinda.
    - **Proxy** provides a surrogate or placeholder for another object to control access to it. interacting with a substitute object instead of the original object. A proxy controls access to the original object, allowing you to perform something either before or after the request gets through to the original object. Generally used with databases or big data. A credit card is a proxy for a bank account and cash in it.

- **Behavioral patterns** take care of effective communication and the assignment of responsibilities between objects. *How objects interact and communicate*

- **Chain of responsibility** allows for the passing of requests to a chain of objects until one of them handles the request.
- **Observer** allows for the notification of observers when an object changes. one-tom-many relationship. lets you define a subscription mechanism to notify multiple objects about any events that happen to the object they're observing.
- **Iterator** allows for the traversal of a collection of objects without exposing its underlying representation. For loop mesela is an iterator. or range. Depth-first or breadth-first traversal is also an iterator.
- **Mediator** allows for the communication between objects without exposing their implementation details and exposing their chaotic dependencies. for example with the profile editing form, the dialog class itself may act as the mediator. Most likely, the dialog class is already aware of all of its sub-elements, so you won't even need to introduce new dependencies into this class.
- **State** allows for the encapsulation of an object's internal state and its behavior when this state changes. and lets an object alter its behavior when its internal state changes. It appears as if the object changed its class. Mesela telefondaki tuslar, context dependent olarak farkli fonksiyonlara gore hareket ederler.

Crypto

1. Symmetric Encryption

Symmetric encryption gathers plain-text data and then shuffles it to make it unreadable. And just before reaching the required party, it re-arranges the data again. Symmetric types of encryption are the fastest of other encryption processes. The viable part to remember here is that the encrypter and decrypter party both need to have the same key to intercept the data. The bad part about the symmetric key is that even if your data is encrypted, the software readily needs the unencrypted data to match the password and not the encrypted one. This indirectly proves that the software itself is compromised. The only to protect yourself is to design the software so that the data remains encrypted when the user logs out of the system and leaves the key only in an unreadable encrypted format which is actually tough, to begin with.

2. Asymmetric Encryption (public key encryption)

Asymmetric encryption, similar to symmetric ones, also gathers plain-text, shuffles it, and re-arranges it again at the other end. Still, here multiple variable keys are used for each end. Users and decrypters use public key and private key to shuffle and re-arrange the data. The only problem with a public key is to make sure you trust the public key you hold. If the public-key is somewhat compromised, then everything is. A simple Man-in-the-middle attack is an easy way to compromise it.

two different keys are used for public-key encryption. One key is used for the encryption process, and another key is used for the decryption process. Once the key is decided for encryption and decryption, no other key will be used. One key is called a public key from these two keys, and another one is called a private key.

Let's assume that you want to communicate with friends over the internet, to start the communication securely; you need to obtain both public and private key. The private key is a secret key; you should keep it

as a secret. If the private key is disposed to another party, there is a chance of attack through the third party.

If X wants to communicate with Y securely, both X and Y should have a public key and private key.

- X should keep her private key secret.
- X should inform her public key to Y.
- Y should keep her private key secret.
- Y should inform her public key to X.

whoever wants to decrypt, use their private keys.

3. Hashing

Nowadays, when you hear the term encryption process, it’s actually hashing what is happening in the background. Hashing is not a pure form of the encryption process, though. Remember the example I gave previously about email security?